CHAPTER 3

TYPES AND MEANING COMPOSITION

This chapter introduces some of the elementary mathematical techniques in formal semantics. We systematize models by organizing denotations in domains of different types. This general type system allows models to describe sets, as well as relations and other operators with multiple arguments. Denotations of complex expressions are compositionally derived by a uniform semantic operation of function application. The resulting semantic framework is demonstrated by treating modified noun phrases (a tall man), reflexive pronouns (herself) and coordinations between different phrases. We avoid excess notation by defining denotations set-theoretically and introducing lambda-style shorthand when convenient.

This chapter systematically explains the way in which models allow linguistic expressions to denote abstract objects. This will give us a better insight into our theory of meaning and its relations with syntactic forms. The first step is to describe how denotations are organized in a model. Throughout Chapter 2, we used models freely to describe different mathematical objects. For sentences we used truth-values, for names like Tina we used entities, and for adjectives like tall we used sets of entities. In addition we used the membership operator for is, the intersection function for and, and the complement function for not. Using various mathematical objects in this manner was useful for expository purposes. However, in general it makes our compositionality principle hard to obtain. With each new mathematical notion we introduce, we need to see how it composes with other denotations. Too much mathematical freedom in the design of the denotations makes it hard to describe how they operate in different natural language expressions. The model structure that we introduce in this chapter helps us to make semantic distinctions between
language expressions within well-defined boundaries. In this way we gain a better understanding of denotations in general, and see more clearly how they interact with syntactic forms and with each other.

Some of the foundational themes in this chapter may seem intimidating at first glance. However, none of them is especially hard. To help you follow this chapter with ease, it is divided into four parts. Each of these parts covers a general topic that leads naturally to the topic of the next one. If you are a novice to the field, it is a good idea to solve the exercises referred to at the end of each part before reading on.

- Part 1 (‘Types and domains’) classifies denotations in models into different domains with different types. An important tool will be functions that characterize sets.
- Part 2 (‘Denotations at work’) elaborates on the composition of denotations and on how typed denotations in a compositional setting are translated to other set-theoretical concepts. An important tool here is functions that operate on other functions.
- Part 3 (‘Using lambda notation’) introduces a short notation for functions by using so-called ‘lambda-terms’. This helps us to define and use denotations in our analyses.
- Part 4 (‘Restricting denotations’) is about denotations that are systematically restricted by our models.

The formal system that is developed throughout this chapter is foundational to many works in formal semantics. For this reason, a technical summary of this chapter is included as an appendix to this book (page 239). This appendix gives readers a global overview of some of the most basic technical assumptions in formal semantics.

**PART 1: TYPES AND DOMAINS**

One of the main goals of this book is to systematically describe semantic distinctions between expressions as they are manifested in entailments. In Chapter 2, the basic difference was between entity denotations of names and truth-value denotations of sentences. Further, we used different functions as the denotations of the words *is*, *and* and *not*. Now we would like to analyze denotations of many more expressions. Therefore, it is high time to introduce some discipline into our semantic framework. In order to deal with denotations in a more
systematic way, we will formally specify the kind of mathematical objects that our models contain. In technical terms, such a specification is referred to as a type system.

CHARACTERISTIC FUNCTIONS

Many of the denotations in formal semantics are functions of different types. To illustrate a simple type of function in formal semantics, we start with a maximally simple sentence:

(3.1) Tina smiled.

Intuitively, the intransitive verb smile should denote a set of entities, just like the adjectives tall and thin in Chapter 2. We conceive of the denotation of the word smiled in (3.1) as the set of entities that smiled at a given moment in the past. For convenience, we often ignore the tense in our discussion, and refer to the verb smiled in (3.1) as being associated with “the set of smilers”. But now, how can sentence (3.1) denote a truth-value? Unlike the sentence Tina is tall, sentence (3.1) contains no word like is that may express the membership function. Thus, the set for smiled and the entity for Tina do not immediately give a truth-value in (3.1). To allow for their easy composition, we should change perspectives slightly. We still use sets of entities for describing denotations of intransitive verbs, but we do that indirectly using functions. For example, suppose that we want to describe a model with three entities: a, b and c. Suppose further that in the situation that the model describes, entities a and c smiled and entity b did not. In this case the set of smilers in the model, S, is the set \{a, c\}. Instead of defining the denotation of the verb smile to be the set S itself, we let it be a function that indirectly describes S. This function, which we denote \(\chi_S\), is a function from entities to truth-values. For each of the two elements in S, entities a and c, the function \(\chi_S\) returns the truth-value 1. For entity b, which is not in S, we let \(\chi_S\) return 0. Thus, \(\chi_S\) is the following function:

(3.2) \(\chi_S : a \mapsto 1 \quad b \mapsto 0 \quad c \mapsto 1\)

The function \(\chi_S\) is called the characteristic function of the set \{a, c\} over the set of entities \{a, b, c\}. In general, we define characteristic functions
as follows:

Let $A$ be a subset of $E$. A function $\chi_A$ from $E$ to the set $\{0, 1\}$ is called the characteristic function of $A$ in $E$ if it satisfies for every $x \in E$:

$$
\chi_A(x) = \begin{cases} 
1 & \text{if } x \in A \\
0 & \text{if } x \notin A 
\end{cases}
$$

For every element $x$ of $E$, the truth-value $\chi_A(x)$ indicates whether $x$ is in $A$ or not. Thus, $\chi_A$ uniquely describes a subset of $E$. The converse is also true: for every subset of $E$ there is a unique characteristic function. This means that sets of entities and their characteristic functions encode precisely the same information. For this reason we often interchangeably talk about subsets of $E$ or the functions that characterize them. Specifically, in Chapter 4, we will refer by ‘$f^*$’ to the set characterized by a function $f$. Further, we will see that functions can also be used for characterizing subsets of other domains besides $E$.

For the time being, for the sake of brevity, we use the general term ‘characteristic functions’ when referring exclusively to functions that characterize sets of entities in $E$. With this notion of characteristic functions, we can easily describe how the composition process in sentence (3.1) works. We assume that the denotation of the verb $\text{smile}$ is an arbitrary characteristic function. This corresponds to our assumption that the verb $\text{smile}$ can be associated with any set of entities. Suppose that the denotation of $\text{smile}$ is the function $\text{smile}$. In our analysis of sentence (3.1), this function applies to the entity denotation $\text{tina}$. In a formula, sentence (3.1) is analyzed as follows:

(3.3) \(\text{smile}(\text{tina})\)

The expression in (3.3) describes the truth-value that the function $\text{smile}$ assigns to the entity $\text{tina}$. For example, in the model we described above, the denotation $\text{smile}$ is the function $\chi_S$ in (3.2). Suppose that in the same model, the denotation $\text{tina}$ is the entity $a$. As a result, the denotation (3.3) of sentence (3.1) is $\chi_S(a)$, which equals 1. If $\text{tina}$ is the entity $b$, the denotation (3.3) equals 0. This way, by letting the denotation of the verb $\text{smile}$ characterize a set, we directly obtain a truth-value denotation for the sentence Tina smiled.
In our analysis of sentence (3.1), we have been using three denotations: an entity, a characteristic function and a truth-value. Each of these denotations has a different ‘nature’, which we distinguish by letting each of them come from a different domain. In Chapter 2, we already let every model \( M \) contain a domain of entities \( E^M \) and a domain of truth-values \( \{0, 1\} \). Now it is time to also introduce domains for characteristic functions and other denotations. Each domain we introduce comes with a label that we call a type. We use the letter \( e \) as the type for the domain \( E^M \) in a given model \( M \). Since we want to make an explicit connection between types and their respective domains, we also use the notation ‘\( D_e^M \)’ (the \( e \) Domain in \( M \)) as an alternative name for \( E^M \). As usual, when the model \( M \) is clear from the context, we write ‘\( D_e \)’ rather than ‘\( D_e^M \)’. The letter \( t \) is used as the type for the domain of truth-values. Accordingly, this domain is denoted ‘\( D_t \)’. Since we fix \( D_t \) as the set \( \{0, 1\} \) in all models, we do not mention the model when referring to this domain. In our example above, the name Tina takes its denotation from \( D_e \), and the sentence Tina smiled takes its denotation from \( D_t \). In short, we say that proper names are of type \( e \) and sentences are of type \( t \). We refer to the types \( e \) and \( t \) as the basic types of our type system. The domains for these types, \( D_e \) and \( D_t \), have been specified with no relation to other domains. For this reason, we refer to them as the basic domains in every model.

Now, we also want to define a type and a domain for characteristic functions like the denotation of smile. These are defined on the basis of the types \( e \) and \( t \), and the domains \( D_e \) and \( D_t \). Specifically, a characteristic function in a model \( M \) is a function from the entities in \( M \) to truth-values. Accordingly, we define the domain of characteristic functions as follows:

\[
(3.4) \text{The domain of characteristic functions in a model } M \text{ is the set of all the functions from } D_e^M \text{ to } D_t.
\]

This domain is assigned the type ‘\( (et) \)’. We often omit outermost parentheses, and refer to the same type as ‘\( et \)’. The corresponding domain is accordingly referred to as ‘\( D_{et}^M \)’, or simply ‘\( D_{et} \)’.

In set theory, there is a common way to refer to the set of all functions from a set \( A \) to a set \( B \). Formally, we use ‘\( B^A \)’ when referring to this set of functions. Thus, the definition of the domain \( D_{et} \) in (3.4)
Table 3.1: Subsets of $D_e$ and their characteristic functions in $D_{et}$.

<table>
<thead>
<tr>
<th>Subset of $D_e$</th>
<th>Characteristic function in $D_{et}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$f_1: a \mapsto 0 \quad b \mapsto 0 \quad c \mapsto 0$</td>
</tr>
<tr>
<td>${a}$</td>
<td>$f_2: a \mapsto 1 \quad b \mapsto 0 \quad c \mapsto 0$</td>
</tr>
<tr>
<td>${b}$</td>
<td>$f_3: a \mapsto 0 \quad b \mapsto 1 \quad c \mapsto 0$</td>
</tr>
<tr>
<td>${c}$</td>
<td>$f_4: a \mapsto 0 \quad b \mapsto 0 \quad c \mapsto 1$</td>
</tr>
<tr>
<td>${a, b}$</td>
<td>$f_5: a \mapsto 1 \quad b \mapsto 1 \quad c \mapsto 0$</td>
</tr>
<tr>
<td>${a, c}$</td>
<td>$f_6: a \mapsto 1 \quad b \mapsto 0 \quad c \mapsto 1$</td>
</tr>
<tr>
<td>${b, c}$</td>
<td>$f_7: a \mapsto 0 \quad b \mapsto 1 \quad c \mapsto 1$</td>
</tr>
<tr>
<td>${a, b, c}$</td>
<td>$f_8: a \mapsto 1 \quad b \mapsto 1 \quad c \mapsto 1$</td>
</tr>
</tbody>
</table>

can be formally written as follows:

$$D_{et} = D_t^{D_e}$$

Functions in the $D_{et}$ domain, as well as expressions of type $et$, are often referred to as *one-place predicates* over entities. Common alternative notations for the type of one-place predicates are $\langle e, t \rangle$ and $e \rightarrow t$. In this book we will stick to the shorter notation $et$.

All intransitive verbs like *smile*, *dance*, *run* etc. are assigned the type $et$. In a given model, each of these verbs may have a different denotation. For example, in the model we described above, the verb *smile* denotes the function $\chi_S$, which characterizes the set $S = \{a, c\}$. Other verbs like *dance* and *run* may be associated with different sets in the same model, and hence denote different characteristic functions. For this reason, the domain $D_{et}^M$ in a given model $M$ includes all the functions from $D_e^M$ to $D_t$. To see which functions these are in our example model, we first note that the domain $D_e$ has eight subsets in that model. These are: the empty set $\emptyset$; the singleton sets $\{a\}$, $\{b\}$ and $\{c\}$; the doubletons $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$; and the whole set $D_e$, i.e. $\{a, b, c\}$. The domain $D_{et}$ includes the eight functions that characterize these sets, as shown in Table 3.1. In such models, where the entities are $a$, $b$ and $c$, intransitive verbs like *smile* and *run* must denote one of the eight functions in $D_{et}$. Specifically, the function $f_6$ in Table 3.1 is the same function $\chi_S$ that we assumed as the denotation of the verb *smile* in our example.
MANY TYPES, MANY DOMAINS

We have seen how to define the domain $D_{et}$ on the basis of $D_e$ and $D_t$. As we consider more expressions besides intransitive verbs, we will need more types and domains. The methods for defining them are similar to the definition of $D_{et}$. The construction of the type $et$ and the domain $D_{et}$ illustrates the general principle that we use for defining new types and domains. We have defined the type $(et)$ as the parenthesized concatenation of the basic types $e$ and $t$. The corresponding domain $D_{et}$ was defined as the set of functions from the domain $D_e$ to the domain $D_t$. We employ the same method for defining more new types and domains. Once two types $\tau$ and $\sigma$ and domains $D_\tau$ and $D_\sigma$ are defined, they are used for defining another type $(\tau\sigma)$ and a domain $D_{\tau\sigma}$, which consists of all the functions from $D_\tau$ to $D_\sigma$. More types and domains are defined in a similar way, with no upper limit on their complexity. Since the same method works for defining all types and domains from the basic ones, we refer to it as an inductive procedure. Specifically, types $e$, $t$ and $et$, and their respective domains, are used inductively for defining new types and domains. For instance, when using type $e$ twice, we get the type $ee$. The respective domain, $D_{ee}$, contains all the functions from $D_e$ to $D_e$. Further, combining the types $e$ and $et$, we get the type $e(et)$. The corresponding domain $D_{e(et)}$ is the set of functions from $D_e$ to $D_{et}$. As we will see below, this $e(et)$ domain is useful for denotations of transitive verbs, i.e. verbs that have both a subject and a direct object.

Definition 1 below formally summarizes our inductive method for specifying types:

**Definition 1.** The set of types over the basic types $e$ and $t$ is the smallest set $T$ that satisfies:

- (i) $\{e, t\} \subseteq T$
- (ii) If $\tau$ and $\sigma$ are types in $T$ then $(\tau\sigma)$ is also a type in $T$.

This inductive definition specifies the set of types as an infinite set $T$, including, among others, the types given in Figure 3.1.

For every model $M$, we specify a domain for each of the types in the set $T$. Let us summarize how this is done. The domain $D^M_e$ is directly specified by the model. The domain $D_t$ is fixed as $\{0, 1\}$ for
all models. Domains for other types are inductively defined, as formally summarized in Definition 2 below:

**Definition 2.** For all types \( \tau \) and \( \sigma \) in \( \mathcal{T} \), the domain \( D_{\tau \sigma} \) of the type \( (\tau \sigma) \) is the set \( D_{\sigma}^{D_{\tau}} \) – the set of functions from \( D_{\tau} \) to \( D_{\sigma} \).

The induction in Definition 2, together with our stipulated basic domains \( D_e \) and \( D_t \), specify the domains for all the types derived from Definition 1. We have already discussed the domain \( D_{et} \) of characteristic functions. Let us now consider in more detail the definition of the domain \( D_{e(et)} \). When unfolding Definition 2, we see that it derives the following definition:

\[
(3.5) \quad D_{e(et)} = \text{the set of functions from } D_{e} \text{ to } D_{et} = \text{the functions from entities to the functions from } D_{e} \text{ to } D_{t} = \text{the functions from entities to the functions from entities to truth-values.}
\]

Thus, functions of type \( e(et) \) return functions (of type \( et \)) as their result. This is a result of our inductive definitions. Our definitions above also make another situation possible: functions that take functions as their *arguments*. For instance, the type \( (et)e \) describes functions that map \( et \) functions to entities. Further, Definitions 1 and 2 also allow functions that take function arguments and map them to function results. Consider for instance the type \( (et)(et) \). The corresponding domain, \( D_{(et)(et)} \), contains the functions that map characteristic functions to characteristic functions. For instance, suppose that \( F \) is a function in \( D_{(et)(et)} \). This means that \( F \) can receive any characteristic function \( g \) in \( D_{et} \) and return a characteristic function \( h \) in \( D_{et} \), possibly
different from $g$. We describe this situation by writing $F(g) = h$. The functions $g$ and $h$ characterize sets of entities. Thus, we can view functions like $F$, of type $(et)(et)$, as mapping sets of entities to sets of entities. We already used one such function in Chapter 2, when we let the denotation of the word *not* map sets of entities to their complement. Functions from sets of entities to sets of entities, in their new guise as $(et)(et)$ functions, will often reappear in the rest of this book.

*You are now advised to solve Exercises 1, 2 and 3 at the end of this chapter.*

**PART 2: DENOTATIONS AT WORK**

Semantic types and their corresponding domains give us a powerful tool for analyzing natural language meanings: one that is empirically rich, and yet systematically constrained. Equipped with our expressive tool for describing denotations, it is high time to start using it for analyzing linguistic examples. In order to do that, we have to explain what principles allow denotations to combine with each other compositionally. One elementary principle lets functions apply to their arguments. As we will see, functions, and the rule of *function application*, allow us to encode many useful intuitions about meanings, using the technique known as *currying*. After introducing this technique of using functions, we will see how to develop systematic analyses by solving *type equations*. This will allow us to look back at what we did in Chapter 2, and systematically treat the copula *be* and predicate negation as part of our uniform type system.

**FUNCTION APPLICATION**

Types provide us with a record of the way denotations combine with each other. In our analysis of the simple example *Tina smiled* we saw how an *et* function combines with an entity (type $e$) to derive a truth-value (type $t$). We write it as follows:

$$(et) + e = t.$$

The rule we used for combining denotations in the sentence *Tina smiled* is *function application*: we applied a function *smile* from entities to truth-values to an entity *tina*, and got a truth-value
smile(tina). In terms of denotations, we write it as follows:

\[ \text{smile}_{et} + \text{tina}_e = \text{smile}(\text{tina}) : t \]

By the notation ‘\text{smile}_{et}’, we refer to the denotation of the verb smile, and state that it is of type \( et \). Similarly for ‘\text{tina}_e’. An alternative notation is ‘\text{smile} : et’ and ‘\text{tina} : e’. This colon becomes more convenient when we wish to state the type \( t \) of the result smile(tina), and write ‘\text{smile}(\text{tina}) : t’. Following standard mathematical practice, we let the function smile appear to the left of its argument tina. However, English verbs like smile normally follow the subject, as is the case in the sentence Tina smiled. The workings of function application are not affected by this. So we also assume:

\[ e + (et) = t. \]

Thus, when wishing to highlight the syntactic ordering, we also describe the composition of denotations in the sentence Tina smiled as follows:

\[ \text{tina}_e + \text{smile}_{et} = \text{smile}(\text{tina}) : t. \]

In more general terms, our type-based rule of function application is given below:

**Function application with typed denotations:** Applying a function \( f \) of type \( \tau\sigma \) to an object \( x \) of type \( \tau \) gives an object \( f(x) \) of type \( \sigma \).

**In short:**

- Types: \( (\tau\sigma) + \tau = \tau + (\tau\sigma) = \sigma \)
- Denotations: \( f_{\tau\sigma} + x_{\tau} = x_{\tau} + f_{\tau\sigma} = f(x) : \sigma \)

The equations that we gave above describe how types are combined with each other. For each type combination, there is a corresponding operation between denotations in the corresponding domains: function application. Such a system, which combines types and denotations, is called a *type calculus*. The type calculus above, which deals with function application, is known as the **Ajdukiewicz Calculus** (after K. Ajdukiewicz). In Chapter 5 we will return to type calculi, and extend their usages for other operations besides function application.
For now, let us see some more examples of the way we use Ajdukiewicz’s calculus:

\[ e + ee = e \quad \text{applying a function } g_{ee} \text{ to an entity } x_e \text{ gives an entity } g(x) \]
\[ e(et) + e = et \quad \text{applying a function } h_{e(et)} \text{ to an entity } x_e \text{ gives an et function } h(x) \]
\[ et + (et)(et) = et \quad \text{applying a function } F_{(et)(et)} \text{ to a function } k_{et} \text{ gives an et function } F(k) \]

These equations each contain two types and their combination using function application. However, for many pairs of types, function application cannot work. For instance, function application cannot combine a function \( f \) of type \( t(et) \) with a function \( g \) of type \( et \). The reason is twofold. First, the function \( f \) cannot apply to \( g \), since \( f \) takes truth-values as its argument, and \( g \) is not a truth-value. Second, the function \( g \) cannot apply to \( f \), since \( g \) takes entities as its argument, and \( f \) is not an entity. Such situations, where the type calculus does not produce any result, are referred to as a \textit{type mismatch}.

**TRANSITIVE VERBS**

Now that we have seen how typed denotations are systematically combined with each other, let us consider the following sentence:

(3.6) Tina [praised Mary]

Sentences like (3.6), which contain both a subject and a direct object, are referred to as \textit{transitive sentences}. In (3.6) we standardly assume that a transitive verb (\textit{praise}) forms a constituent with the object noun phrase (\textit{Mary}). This means that, in order to compositionally derive a truth-value for sentence (3.6), we first need to derive a denotation for the verb phrase \textit{praised Mary}. To do that, we follow our treatment of intransitive verbs. In the same way that the denotation of the verb \textit{smiled} characterizes the set of entities that smiled, we now want the denotation of the verb phrase \textit{praised Mary} to characterize the set of entities that praised Mary. This is the function that sends every entity that praised Mary to 1, and any other entity to 0. How do we derive such an \textit{et} function from the denotations of the words \textit{praised}.
Figure 3.2 Syntactic structure and semantic interpretation for Tina praised Mary.

and Mary? The key to doing so is to assume that the verb praise denotes a function of type $e(et)$. As we have seen in (3.5) above, functions of type $e(et)$ map entities to $et$ functions. Thus, we let the verb praise denote an $e(et)$ function $\text{praise}$. Applying this function to the entity $\text{mary}$, we get a denotation of type $et$ for the verb phrase praised $\text{Mary}$. This is formally written as follows:

$$ (3.7) \quad \text{praise}_{e(et)} + \text{mary}_e = \text{praise(mary)} : et $$

Further, in the compositional analysis of the whole sentence (3.6), the $et$ function in (3.7) applies to the entity $\text{tina}$. What we get is the following truth-value:

$$ (3.8) \quad \text{praise(mary)} + \text{tina}_e = (\text{praise(mary)})(\text{tina}) : t $$

In words: when the $et$ function $\text{praise(mary)}$ applies to the entity $\text{tina}$, the result is a truth-value. This truth-value is the denotation that our model assigns to the sentence (3.6). To increase readability, we often omit obvious types and parentheses, and write this truth-value as:

$$ (3.9) \quad \text{praise(mary)}(\text{tina}) $$

To summarize the compositional process in our analysis of sentence (3.6), we repeat it in Figure 3.2 using tree notation.

Figure 3.2A is the tree notation of the structure we assumed in (3.6). Figure 3.2B is the same tree, but the nodes are now decorated with their
types and their denotations. We refer to tree diagrams like Figure 3.2B as *semantically interpreted structures*. In this interpreted structure, the nodes for the words *Tina*, *praised* and *Mary* are decorated by their lexical types and denotations. In addition, we have two nodes for the constituents assumed in the binary structure: the verb phrase *praised Mary* and the whole sentence. These nodes are decorated by their types and denotations, which are compositionally derived by function application.

Any $e(et)$ function can combine with two entities, one entity at a time, returning a truth-value. Having seen how $e(et)$ functions allow us to derive truth-values for transitive sentences like (3.6), we may still feel that functions that return functions as their result are excessively intricate when analyzing such simple sentences. Fortunately, there is an equivalent way of looking at $e(et)$ denotations of transitive verbs, which better reflects their intuitive simplicity. Intuitively, denotations of verbs like *praised* can be viewed as two-place relations, aka *binary relations*. Such relations are sets of pairs of entities. In our example, we may view the denotation of the verb *praised* as the set of pairs of entities $\langle x, y \rangle$ that satisfy the condition $x$ praised $y$. For instance, suppose that in our model, the entities $t$, $j$ and $m$ are the denotations of the respective names *Tina*, *John* and *Mary*. When the domain $D_e$ is $\{t, j, m\}$, we may describe who praised who by the following binary relation $U$:

$$U = \{(t, m), (m, t), (m, j), (m, m)\}$$

The relation $U$ is useful for describing a situation with three people, where Tina only praised Mary, John praised no one, and Mary praised everybody, including herself. In this way, the relation $U$ provides full answers to the following questions:

\begin{align*}
(3.10) \quad & a. \quad \text{Who praised Tina?} \quad \text{Answer: only Mary.} \\
& b. \quad \text{Who praised John?} \quad \text{Answer: only Mary.} \\
& c. \quad \text{Who praised Mary?} \quad \text{Answer: only Tina and Mary herself.}
\end{align*}

Conversely: anybody who gives the same answers as in (3.11) will have implicitly described the binary relation $U$.

Now we can get back to $e(et)$ functions, and observe that they give us the same information as binary relations like $U$. In particular, the
The function $\chi_U$ maps each of the three entities in $D_e$ to an $et$ function. More specifically:

- $\chi_U$ maps the entity $t$ to the function characterizing the set $\{m\}$.
- $\chi_U$ maps the entity $j$ to the function characterizing the same set, $\{m\}$.
- $\chi_U$ maps the entity $m$ to the function characterizing the set $\{t, m\}$.

Note the parallelism between this specification of $\chi_U$ and the question–answer pairs in (3.11). When the denotation *praise* is $\chi_U$, our model describes the same situation that (3.11) describes in words, which is the same information described by the binary relation $U$. More generally, we conclude that $e(et)$ functions encode the same information as binary relations over entities. This is similar to how characteristic functions of type $et$ encode the same information as sets of entities. In mathematical terms, we say that the domain of $e(et)$ functions is isomorphic to the set of binary relations over $D_e$. Because $e(et)$ functions take two entities before returning a truth-value, we sometimes also refer to them as *two-place predicates*.

### CURRYING

There is a general lesson to be learned from our treatment of transitive sentences and $e(et)$ predicates. We have seen how a situation that is naturally described by a binary relation can equally be described by a function that returns functions. The general idea is useful in many other circumstances in formal semantics (as well as in computer science). A binary relation between entities is a set containing pairs of entities. We can characterize such a set by a function that takes pairs of
entities and returns “true” or “false”. Such _two-place functions_ occur very often in mathematics. As another example, let us consider one of the most familiar two-place functions: number addition. This function takes two numbers, \( x \) and \( y \), and returns their sum, which we normally denote ‘\( x + y \)’. To highlight the fact that number addition is a two-place function, let us denote it using the function symbol \( \text{sum} \). Thus, we use the following notation:

\[
\text{sum}(x, y) = x + y
\]

Now, let us use the letter ‘\( n \)’ as the type for natural numbers. Using this type, we will now see how we can also encode addition as a function of type \( n(n) \): a function from numbers to functions from numbers to numbers. Let us refer to this function as \( \text{ADD} \). To define \( \text{ADD} \), we need to define the result that it assigns to any given number. This result is an \( nn \) function: a function from numbers to numbers. Therefore, to define \( \text{ADD} \) we will now specify the \( nn \) function that \( \text{ADD} \) assigns to any number. For every number \( y \), we define:

(3.13) The \( nn \) function \( \text{ADD}(y) \) sends every number \( x \) to the number \( \text{sum}(x, y) \).

As we expect from number addition, the function \( \text{ADD} \) takes two numbers and returns their sum. But it does it step by step: it first takes one number, \( y \), it returns a function \( \text{ADD}(y) \), and this function applies to another number \( x \) and returns the sum \( \text{sum}(x, y) \), or more simply: \( x + y \). As a result, for every two numbers \( x \) and \( y \), we get:

\[
\text{ADD}(y)(x) = \text{sum}(x, y) = x + y
\]

For example, let us consider how we calculate the sum of 1 and 5 using the function \( \text{ADD} \). We first give \( \text{ADD} \) the number 1 as an argument, and get the function \( \text{ADD}(1) \) as the result. This resulting function can take any number and return its sum with 1. Now, we choose to give the function \( \text{ADD}(1) \) the argument 5. Unsurprisingly, the result of calculating \( (\text{ADD}(1))(5) \) is \( 5 + 1 \), or 6. Have we gained anything from reaching this obvious result in such a roundabout way? As strange as it may sound, we have! While calculating the sum of 5 and 1, we generated the function \( \text{ADD}(1) \). This is the _successor_ function: the function that sends every natural number to the number that
follows it. This function is of course useful for other purposes besides applying it to the number 5.

These different ways of looking at number addition are quite similar to what we saw in our syntactic-semantic analysis of sentence (3.6). In that analysis, we equivalently encoded situations either using binary relations like $U$ or using $e(et)$ functions. Because we adopt the latter method, we got an intermediate result by applying the $e(et)$ denotation of the verb *praise* to the entity denotation of the object *Mary*. This is the $et$ denotation of the verb phrase *praised Mary*. Having such a denotation for the verb phrase is compositionally useful. As we will see later in this chapter, it gives us a natural treatment of conjunctive sentences like *Tina smiled and praised Mary*, where the verb phrase *praised Mary* does not combine directly with the subject *Tina*.

The kind of maneuver we saw above will also be useful for treating many other phenomena besides transitive sentences. In its full generality, the idea is known as Currying (after H. Curry) or, less commonly, as Schönfinke‌lization (after M. Schönfinkel). In technical slang we often say that a one-place function like $ADD$ is a Curried version of the two-place addition operator $sum$. Conversely, we say that the addition function $sum$ is an unCurried (or ‘deCurried’) version of $ADD$. For more on Currying, see the suggested further reading at the end of this chapter.

SOLVING TYPE EQUATIONS

Using Currying, we now present a revised treatment of the copular sentences with adjectives from Chapter 2. Reconsider the following sentence:

(3.14) Tina [ is tall ]

In Chapter 2 we analyzed the verb *is* in (3.14) as the membership function. This two-place function sends pairs, of entities and sets of entities, to truth-values. However, now we no longer have two-place functions and sets of entities in our models: we have replaced them by Curried functions and characteristic functions, respectively. Therefore, we need to revise our analysis of (3.14) along these lines. First, as for intransitive verbs, we let adjectives characterize sets of entities. Thus, in our analysis of sentence (3.14) we assume that the adjective *tall* denotes an $et$ function. In many other languages
besides English, this allows us to treat sentences of the form *Tina tall*. However, English requires the copula *be* to appear in such sentences. How should we now analyze the semantic role of the English copula? Let us first consider its type. Compositional interpretation of the structure in (3.14) means that the denotation of the constituent *is tall* has to be determined before we determine the denotation of the sentence. To be able to combine with the denotation of the subject *Tina*, the constituent *is tall* has to denote a function that applies to the entity *tina* and derives a truth-value. Therefore, the expression *is tall* should have the same type *et* as the adjective *tall*. We conclude that the denotation of *is* has to be a function of type (*et*)(*et*): a function from *et* functions to *et* functions. When such an (*et*)(*et*) denotation for the word *is* applies to an *et* function like *tall*, it gives a function of the same type, *et*, which we will use as the denotation for the constituent *is tall*.

What we have done above is a kind of puzzle solving. We assumed solutions to some parts of the puzzle: the types of the words *Tina* and *tall*, and the type *t* of the whole sentence. Using the sentence’s structure, we found a suitable type for the word *is* that allows function application to compositionally derive for the sentence a *t*-type denotation. The puzzle can be summarized as follows, with *X* and *Y* as the unknown types:

\[(3.15) \ [ \text{Tina}_e \ [ \text{is}_Y \ \text{tall}_{et} ]_X ]_t \]

In our solution, we found that *X* = *et* and *Y* = (*et*)(*et*). The solution process itself is described in Figure 3.3. This figure contains two type equations. One equation is:

**Eq. 1:** \[ e + X = t \]

In words: which type(s) *X* combines with type *e* and derives type *t*?

By solving this equation, we see that the type for the constituent *is tall* must be *et*. This leads us to another equation in the compositional analysis of the sentence, which helps us to see how we can derive the type *et* for this constituent:

**Eq. 2:** \[ Y + et = et \]

In words: which type(s) *Y* combines with type *et* and derives type *et*?
Figure 3.3  Solving type equations for the sentence Tina is tall.

Solving this equation as well, we see that the type for the copula is must be \((et)(et)\).

BACK TO COPULAS AND PREDICATE NEGATION

Now, after determining the type of the copula is, we want its denotation to preserve the welcome results of our analysis in Chapter 2. To do that, we let the constituent is tall denote the same et function as the adjective tall. This is because, intuitively, we still want the sentence Tina is tall to be interpreted as a membership assertion, claiming that the entity tina is in the set that the function tall,et characterizes. Thus, we assume that the word is denotes the identity function for et functions: the function of type \((et)(et)\) that maps any et function to itself. Formally, we define the denotation is for the word is as the following \((et)(et)\) function:

\[
(3.16) \quad \text{is is the function sending every function } g \text{ of type } et \text{ to } g \text{ itself.}
\]
Our compositional, type-theoretical analysis is summarized in Figure 3.4.

Because the denotation of the copula *is* is now defined as the identity function, the truth-value that we get for sentence (3.14) can now be analyzed as follows in (3.17):

(3.17) a. \( \text{IS}(\text{tall}) = \text{tall} \) (by definition of \( \text{IS} \) in (3.16))

b. \( (\text{IS}(\text{tall}))(\text{tina}) = \text{tall}(\text{tina}) \) (due to the equality in (3.17a))

Due to this simple derivation, the truth-value that we derive in (3.17b) for sentence (3.14) is 1 if and only if the entity *tina* is in the set characterized by the function *tall*.

Functions of type \((\text{et})(\text{et})\) are also useful for adjusting our account of predicate negation from Chapter 2. Let us reconsider, for example, the following negative sentence:

(3.18) *Tina [ is [ not tall ]]*

In Chapter 2 we let the negation word *not* denote the complement function, which sends every set of entities to its complement set. With our characteristic functions substituting sets, we now treat the word *not* as an \((\text{et})(\text{et})\) function, of the same type as the copula *is*. Our definition of the denotation \(\text{NOT}\) in (3.19) below respects the status of the word *not* in (3.18) as a predicate negator:

(3.19) \(\text{NOT}\) is the \((\text{et})(\text{et})\) function sending every \(\text{et}\) function \(g\) to the \(\text{et}\) function \(\text{NOT}(g)\) that satisfies the following, for every entity \(x\) (see next page):
In words: we define the function \textsc{not}, which takes a function \textit{g} of type \textit{et} and returns a function \textsc{not}(g) of the same type, separating into the following two cases:

- for any entity \textit{x} such that \textit{g}(\textit{x}) is 0, we define \textsc{not}(\textit{g})(\textit{x}) to be 1;
- for any entity \textit{x} such that \textit{g}(\textit{x}) is 1, we define \textsc{not}(\textit{g})(\textit{x}) to be 0.

In this way, the function \textsc{not}(\textit{g}) reverses the value that \textit{g} assigns to any entity. Because of that, \textsc{not}(\textit{g}) is the \textit{et} function characterizing the complement set of the set characterized by \textit{g}. Thus, by applying the function \textsc{not} to an \textit{et} denotation \texttt{tall}, we achieve the same analysis that we got in Chapter 2 by complementing the set characterized by \texttt{tall}. More formally, we analyze the structure (3.18) as denoting the following truth-value:

\begin{equation}
(\textsc{is}(\textit{et})(\textit{et})(\textsc{not}(\textit{et})(\textit{et})(\texttt{tall}_{\textit{et}})))(\texttt{tina}_{\textit{e}})
\end{equation}

Because the function \textsc{is} is the identity function, the truth-value in (3.20) is the same as:

\begin{equation}
(\textsc{not}(\textit{et})(\texttt{tall}_{\textit{et}})))(\texttt{tina}_{\textit{e}})
\end{equation}

By definition of the function \textsc{not}, this truth-value is 1 if and only if the entity \texttt{tina} is in the complement of the set characterized by the \textit{et} function \texttt{tall}. Thus, the structure (3.18) is analyzed on a par with our analysis of the sentence in Chapter 2.

Let us now take stock of what we have done in Part 2. In this part we have aimed to maintain the informal analyses of Chapter 2 within a more structured type system. This system was fully defined by the two simple Definitions 1 and 2 in Part 1. As we have seen, these two definitions are highly expressive: they allowed models to mimic sets by using characteristic functions, and mimic two-place functions by using one-place Curried functions. Despite this expressiveness, so far we have done little to extend the empirical coverage of Chapter 2 besides adding a treatment of transitive verbs. However, by employing types as part of our theory we now have a rather powerful system that elucidates our notions of denotations and compositionality. This unified
framework will be employed throughout this book for dealing with new phenomena while relying on the same foundational principles.

You are now advised to solve Exercises 4, 5, 6 and 7 at the end of this chapter.

PART 3: USING LAMBDA NOTATION

Having functions of different types in our semantic framework gives us an extremely powerful tool. In order to use this tool efficiently, it is convenient to have a standard notation for the functions we use, and the way they apply to their arguments. In this part we study the common notation of lambda terms, and see how it is used within our semantic system.

DEFINING FUNCTIONS USING LAMBDA TERMS

Below we restate definition (3.16) of the denotation for the copula is:

(3.21) the function sending every function $g$ of type $et$ to $g$ itself.

We may feel that (3.21) is an unnecessarily long and cumbersome way of defining the identity function. Indeed, in formal semantics we often use a more convenient notation, by employing lambda terms, or ‘$\lambda$-terms’. Let us illustrate it by rewriting definition (3.21) in our new notation:

- Instead of writing “the function sending every function $g$ of type $et$,” we write “$\lambda get\cdot g$.”
- Instead of “to $g$ itself”, we write “$g\cdot g$.”

After rewriting (3.21) in this way, we get the following formula:

(3.22) $\lambda get\cdot g$

Since (3.22) is nothing but an alternative way of defining the function in (3.21), it gives us exactly the same information about it:

(i) The letter ‘$\lambda$’ tells us that it is a function.

(ii) The dot separates the specification of the function’s argument and the definition of the function’s result. Before the dot, writing ‘$get\cdot g$’
introduces ‘g’ as an ad hoc name for the argument of the function. The type et in the subscript of g tells us that this argument can be any object in the domain $D_{et}$.

(iii) The re-occurrence of ‘g’ after the dot tells us that the function we define in (3.22) simply returns the value of its argument.

From (ii) and (iii) we immediately conclude that the function $\lambda g_{et}.g$ in (3.22) returns an object in the domain $D_{et}$. Hence this function is of type $(et)(et)$, as we wanted. Now, with our $\lambda$-term conventions, we are fully justified in saving space and writing our definition of the denotation for the copula is concisely, as in (3.23) below:

(3.23) $IS = \lambda g_{et}.g$

It is important to note that the letter ‘g’ has no special significance in definition (3.23). If we prefer, we may define the function IS equivalently, as $\lambda h_{et}.h$: “the function sending every function $h$ of type et to $h$ itself”. This would not change anything about our definition of the identity function, since it would still do the same thing: return the et function that it got as argument. When defining a function, it hardly matters if we decide to call the argument ‘g’, ‘h’, ‘x’, ‘y’ or any other name. Any name will do, as long as we use it consistently within the function definition.

With our new lambda notation, we adopt the following convention for writing function definitions:

**Lambda notation:** When writing “$\lambda x_{\tau}.\varphi$”, where $\tau$ is a type, we mean:

“the function sending every element $x$ of the domain $D_{\tau}$ to $\varphi$”.

The expression $\varphi$ within the lambda term $\lambda x_{\tau}.\varphi$ specifies the object that we want the function to return. In our definition of the identity function in (3.23), the expression $\varphi$ was simply the argument $g$ itself. However, in general, any mathematical expression $\varphi$ that describes an object in one of our domains would be appropriate in such a $\lambda$-term. The type of the object that $\varphi$ describes is the type of the value returned by the function. In (3.23), the type of the value $g$ returned by the function is et, and hence the function is of type $(et)(et)$. Similarly,
we have:

\[ \lambda x_e. x \]  
the \( ee \) function sending every entity \( x \) to \( x \) itself

\[ \lambda f_{et}. tina_e \]  
the \( et \)\( e \) function sending every \( et \) function \( f \) to the entity \( tina \)

\[ \lambda h_{et}. h(tina_e) \]  
the \( et \)\( t \) function sending every \( et \) function \( h \) to the truth-value \( h(tina) \) that \( h \) assigns to the entity \( tina \)

More generally, when \( \varphi \) describes an object of type \( \sigma \), the whole \( \lambda \)-term \( \lambda x_\tau. \varphi \) describes a function of type \( \tau \sigma \): from objects in \( D_\tau \) to objects in \( D_\sigma \).

**FUNCTION APPLICATION WITH LAMBDA TERMS**

Now we can see how \( \lambda \)-terms are used in the semantic analysis. Based on definition (3.23), we can rewrite the equation \( IS(\text{tall}_{et}) = \text{tall} \) in (3.17a) as follows:

\[
(3.24) \quad \text{a. } IS(\text{tall}_{et}) = \text{tall} \\
\text{b. } = (\lambda g_{et}. g)(\text{tall}) \\
\text{c. } = \text{tall}
\]

The move from (3.24a) to (3.24b) is according to the definition of the function \( IS \). The move from (3.24b) to (3.24c) involves applying a function to its argument. As a result, we replace the abstract description in (3.24b) “what the identity function returns when it gets the argument \( \text{tall} \)” by writing more simply and concretely “\( \text{tall} \)”. Function application with \( \lambda \)-terms always involves this sort of concretization. Suppose that we have a function \( f \) described by the instruction:

“for every \( x \) of type \( \tau \) do such and such to \( x \) and return the result”.

Suppose further that we apply \( f \) to an argument \( a \) of the right type. What we get as the value \( f(a) \) is whatever “doing such and such” does to \( a \).

In our lambda notation, the expression \( \varphi \) in a \( \lambda \)-term \( \lambda x. \varphi \) describes what the function does with its argument. Within the identity function \( \lambda g_{et}. g \) in (3.24), the expression \( \varphi \) is the argument \( g \) itself. However, \( \varphi \) may encode a more complex operation on the function argument. For instance, let us again consider operations on numbers. Consider a
function DOUBLER that maps any number to its multiplication by 2:

\[
\text{DOUBLER is the function from numbers to numbers sending}
\text{every number } x \text{ to } 2 \cdot x.
\]

In lambda format, this definition is written as follows, where \( n \) is again the type of numbers:

\[
(3.25) \quad \text{DOUBLER}_n = \lambda x_n . 2 \cdot x
\]

Having a name like DOUBLER for this function is convenient when we commonly want to double numbers. However, when using \( \lambda \)-terms we can also avoid giving this function a name, and apply the term that corresponds to the function definition directly to the function’s argument, as we did in (3.24b) above. Suppose that we apply the function DOUBLER to the number 17. We get the following term:

\[
(3.26) \quad (\lambda x_n . 2 \cdot x)(17)
\]

This corresponds to the following verbal description:

“the result of applying the function sending every number \( x \) to \( 2 \cdot x \), to the number 17”.

Of course, the result is the value of the arithmetic expression \( 2 \cdot 17 \). We get this expression by substituting ‘17’ for ‘\( x \)’ in the result definition \( 2 \cdot x \), as it is defined in the \( \lambda \)-term \( \lambda x_n . 2 \cdot x \). In sum, we get:

\[
(3.27) \quad (\lambda x_n . 2 \cdot x)(17) = 2 \cdot 17
\]

In general, we describe this kind of simplification as follows:

**Function application with lambda terms**: The result \( (\lambda x_\tau . \varphi)(a_\tau) \) of applying a function described by a lambda term \( \lambda x_\tau . \varphi \) to an argument \( a_\tau \) is equal to the value of the expression \( \varphi \), with all occurrences of \( x \) replaced by \( a \).

Consider how this convention works in (3.27). The expression ‘\( \varphi \)’ within the lambda term \( \lambda x_n . 2 \cdot x \) is the expression \( 2 \cdot x \). The argument ‘\( a \)’ is the number 17. Substituting 17 for \( x \) in \( 2 \cdot x \), we get the result \( 2 \cdot 17 \). Something similar happens when we use a \( \lambda \)-term for defining
the function \( \text{ADD} \) that we introduced in (3.13):

\[
\text{ADD} = \lambda y_n.\lambda x_n. y + x
\]

As in (3.13), this \( \lambda \)-term defines the function \( \text{ADD} \) by specifying that it sends every number \( y \) to the function \( \lambda x_n. y + x \): the function sending every number \( x \) to \( y + x \). When describing the result of applying the function \( \text{ADD} \) to the values 1 and 5, we get the following simplifications:

\[
(\lambda y_n.\lambda x_n. y + x)(1)(5) = (\lambda x_n. 1 + x)(5) = 1 + 5
\]

The simplification in (3.28) involves two steps. First, when the function \( \text{ADD} \) takes the argument 1, this value is substituted for ‘\( y \)’. The result is the function \( \lambda x_n. 1 + x \). This function applies to 5. When 5 is substituted for ‘\( x \)’, we get the result \( 1 + 5 \).

In (3.24), (3.27) and (3.28) above we use \( \lambda \)-terms for function application. In all of these cases, when a function \( \lambda x. \varphi \) applies to an argument \( a \), the result \( (\lambda x. \varphi)(a) \) was displayed in a simplified notation, by substituting \( a \) for \( x \)’s occurrences in \( \varphi \). This substitution technique is based on common mathematical intuitions about the workings of functions. However, as usual, we should be careful when formalizing intuitions. There are some cases where naively using substitution as described above fails to derive the correct results. When using \( \lambda \)-terms for actual calculations, we need a more general rule for simplifying \( \lambda \)-terms under function application. This rule is known as \textit{beta-reduction}, and it constitutes part of the rule system for computing \( \lambda \)-term equivalences, known as the \textit{lambda calculus}. For our purposes in this book we will not need the full lambda calculus. Rather, the informal notational conventions above for writing and simplifying \( \lambda \)-terms will be sufficient. The reader is assured that none of our examples involve the formal complications that motivated the more intricate rule of beta-reduction in the lambda calculus. For more on this point, see the further reading at the end of this chapter.

**REFLEXIVE PRONOUNS**

In order to get a better feel for the value of lambda notation, let us now see how it works in a more complicated example: the case of \textit{reflexive pronouns} like \textit{herself} or \textit{himself}. Consider the following sentence:
(3.29) Tina [praised herself]

Intuitively, this sentence should be treated as having the following truth-value:

(3.30) $\text{praise}_{e(et)}(tina)(tina)$

In words, (3.30) is the truth-value that results from applying the denotation of $\text{praise}$ twice to the entity denotation of Tina. What denotation of the pronoun herself can guarantee that this truth-value is derived for sentence (3.29)? One obvious way to derive (3.30) from (3.29) is to let the pronoun herself denote the entity tina. This would be a correct treatment of sentence (3.29), but it definitely could not work as a general account of reflexive pronouns. To see why, let us consider the sentence Mary praised herself. Here, we must guarantee that the entity for Mary is given twice to the function $\text{praise}$. More generally, for any entity $x$ denoted by the subject, we must guarantee that that same entity is given twice as an argument to the function $\text{praise}$. This “sameness” cannot be described by treating the pronoun herself as denoting an entity of type $e$. For instance, if we analyzed herself as denoting the entity tina, the unwelcome result would be that the sentence Mary praised herself would be analyzed as having the same truth-value as Mary praised Tina. Similar problems would appear for any analysis of the pronoun herself as an entity-denoting noun phrase.

Let us see how we solve the problem. First, following our previous analyses, we let the verb phrase praised herself in (3.29) denote a function of type $et$. In this way we can treat it on a par with verb phrases like praised Mary. Thus, we want to solve the following type equation for the verb phrase praised herself, where $X$ is the type for herself:

$$e(et) + X = et$$

Letting $X$ be $e$ was fine type-theoretically, but it did not allow us to respect the semantic properties of reflexive pronouns. Fortunately, there is another solution: $X = (e(et))(et)$. Thus, we will let the pronoun herself denote a function that takes the $e(et)$ denotation $\text{praise}$ as an argument, and returns a function of type $et$: the denotation of the phrase praised herself. These type-theoretical ideas about the analysis of sentence (3.29) are summarized in Figure 3.5.
The analysis in Figure 3.5 does not yet define the \((e(et))(et)\) function that the pronoun \(\textit{herself}\) should denote. To define it correctly, we rely on our intuition that sentence (3.29) has to have the same truth-value as (3.30). Thus, what we want to achieve is the following identity:

\[(3.31) \left[ \left[ \textit{praised herself} \right] \right](\textit{tina}) = \text{praise}(\textit{tina})(\textit{tina})\]

In words: the one-place predicate for \textit{praised herself} holds of the entity \textit{tina} if and only if the two-place predicate \textit{praise} holds of the pair \(⟨\textit{tina}, \textit{tina}⟩\).

The denotation \(\left[ \left[ \textit{praised herself} \right] \right]\) in (3.31) is obtained by applying the denotation of \textit{herself} to the function \textit{praise}. Thus, by spelling out the denotation \(\left[ \left[ \textit{praised herself} \right] \right]\), we restate (3.31) as the following identity:

\[(3.32) \left( \text{HERSELF}(e(et))(et) \left( \text{praise}_{e(et)} \right) \right)(\textit{tina})
= \left[ \left[ \textit{praised herself} \right] \right](\textit{tina})
= \text{praise}(\textit{tina})(\textit{tina})\]

In words: when the function \textit{herself} applies to the denotation \textit{praise}, the result is the \textit{et} function \(\left[ \left[ \textit{praised herself} \right] \right]\) in (3.31) above. As we saw, this resulting \textit{et} function sends the entity \textit{tina} to 1 if and only if the two-place predicate \textit{praise} holds of the pair \(⟨\textit{tina}, \textit{tina}⟩\).
Since the denotations of the verb *praise* and the name *Tina* are arbitrary, we want the function *HERSELF* to satisfy the identity in (3.32) for all possible \( e(et) \) and \( e \) denotations. We therefore conclude:

(3.33) For all functions \( R \) of type \( e(et) \), for all entities \( x \):

\[ (HERSELF(e(et))(et))(R))(x) = R(x)(x) \]

This generalization defines the result of the function \( HERSELF \) for every argument of type \( e(et) \). Thus, we get:

(3.34) \( HERSELF(e(et))(et) \) is

the function sending every function \( R \) of type \( e(et) \) to the \( et \) function sending every entity \( x \) to \( R(x)(x) \).

This definition is rather long. Let us save space and use a \( \lambda \)-term for rewriting it:

(3.35) \( HERSELF(e(et))(et) \)

\[ = \lambda R_{e(et)}.the\;et\;function\;sending\;every\;entity\;x\;to\;R(x)(x) \]

But behold: we can save more space! The \( et \) function that \( HERSELF \) returns can also be written as a lambda term: \( \lambda x.e.R(x)(x) \). Here is what we get when we use it in (3.35):

(3.36) \( HERSELF(e(et))(et) \)

\[ = \lambda R_{e(et)}.(\lambda x.e.R(x)(x)) \]

Since we don’t like unnecessary parentheses, we will write instead:

(3.37) \( \lambda R_{e(et)}.\lambda x.e.R(x)(x) \)

And this is as much as we get by abbreviating our definition for the function \( HERSELF \).

We can now write our analysis of the truth-value of sentence (3.29) using \( \lambda \)-terms alone, with some notes for clarification, but without any
description of functions in natural language:

\[(3.38)\]

\[a. \quad (\text{HERSELF}_{e(\text{et})})(\text{praise}_{e(\text{et})}))(\text{tina}) \quad \blacktriangleright \quad \text{compositional analysis (Figure 3.5)}
\]

\[b. \quad = ((\lambda R_{e(\text{et})}. \lambda x_{e}. R(x)(x)))(\text{praise}))(\text{tina}) \quad \blacktriangleright \quad \text{definition (3.37) of HERSELF}
\]

\[c. \quad = (\lambda x_{e}. \text{praise}(x)(x))(\text{tina}) \quad \blacktriangleright \quad \text{application to praise}
\]

\[d. \quad = \text{praise}(\text{tina})(\text{tina}) \quad \blacktriangleright \quad \text{application to tina}
\]

Our compositional analysis assigns the verb phrase \textit{praised herself} the denotation \text{HERSELF}_{\text{praise}}, of type \text{et}. This is the same type we assigned to the intransitive verb \textit{smiled} and to the verb phrase \textit{praised Mary}. As we see in (3.38b–c), with our definition of the function \text{HERSELF}, the denotation we get for \textit{praised herself} is the \text{et} function \(\lambda x_{e}. \text{praise}(x)(x)\), which characterizes the set of self-praisers, as intuitively required by sentence (3.29). By letting all verb phrases denote \text{et} functions we obtain pleasing uniformity in our system. This uniformity will prove useful later on in this chapter, when we analyze verb phrase conjunctions like \textit{smiled and praised herself}, as we will do in (3.53)–(3.54) below, and in Exercise 12c.

\textit{You are now advised to solve Exercises 8, 9, 10 and 11 at the end of this chapter.}

\textbf{PART 4: RESTRICTING DENOTATIONS}

In Chapter 2 we introduced the distinction between constant and arbitrary denotations. This distinction is also instrumental with our new type system. For words like \textit{Tina, smile, tall} and \textit{praise}, we assume arbitrary denotations of the type we assign. By contrast, when analyzing functional words like \textit{is, not and herself}, we focus on one denotation of the relevant type. In this part we study more examples where denotations are restricted in this way. This will lead us to some general conclusions about the relations between formal semantics and the specification of lexical meanings.

\textbf{PROPOSITIONAL NEGATION AND PREDICATE NEGATION}

Let us now get back to the negation in sentence (3.18) (\textit{Tina is not tall}). We analyzed this sentence using definition (3.19), which describes
the set complement operation using characteristic functions. Using \( \lambda \)-terms we restate this definition as follows:

\[
(3.39) \text{NOT} = \lambda g_{et}. \lambda x_e. \begin{cases} 
1 & \text{if } g(x) = 0 \\
0 & \text{if } g(x) = 1
\end{cases}
\]

In words: the function NOT sends every \( et \) function \( g \) to the function that sends every entity \( x \) to 1 in case \( g(x) = 0 \), and to 0 in case \( g(x) = 1 \).

Driven by our wish to save space, we can shorten up the “piece-wise” definition in (3.39) by using propositional negation: the function from truth-values to truth-values that sends 0 to 1, and 1 to 0. We denote this \( tt \) function ‘\( \sim \)’. For its formal definition we can use subtraction, as stated in (3.40) below:

\[
(3.40) \sim = \lambda x_{e}. 1 - x
\]

Using this definition of propositional negation, we can rewrite definition (3.39) above more concisely as:

\[
(3.41) \text{NOT} = \lambda g_{et}. \lambda x_e. \sim(g(x))
\]

The three definitions (3.19), (3.39) and (3.41) are equivalent, and they all boil down to the analysis of the word \textit{not} as set complementation. However, since we now work with characteristic functions, our use of propositional negation in (3.41) has some presentational advantages. Given the structure in (3.18) (page 62), we analyze the sentence \textit{Tina is not tall} as in (3.42) below.

\[
(3.42) \begin{align*}
&\text{a. } (\text{IS}(et)(et)(\text{NOT}(et)(et)(\text{tall}_{et}))(\text{tina}_{e})) (tina) \\
&\quad \triangleright \text{compositional analysis of structure (3.18)} \\
&\text{b. } = ((\lambda g_{et}. g)(\text{NOT(tall)}))(\text{tina}) \\
&\quad \triangleright \text{definition of IS as identity function} \\
&\text{c. } = (\text{NOT(tall)})(\text{tina}) \\
&\quad \triangleright \text{application to NOT(tall)} \\
&\text{d. } = ((\lambda g_{et}. \lambda x_e. \sim(g(x)))(\text{tall}))(\text{tina}) \\
&\quad \triangleright \text{definition (3.41) of NOT} \\
&\text{e. } = ((\lambda x_e. \sim(\text{tall}(x)))(\text{tina})) \\
&\quad \triangleright \text{application to tall} \\
&\text{f. } = \sim(\text{tall(tina)}) \\
&\quad \triangleright \text{application to tina}
\end{align*}
\]
In (3.42) we see two different representations of the same truth-value that our model assigns to sentence (3.18). In (3.42a–c) it is easy to see that the truth-value assigned to the sentence is 1 if the denotation *tina* is in the complement of the set characterized by the *et* denotation *tall*. This was the view that we adopted in Chapter 2. It is also very much in line with our compositional analysis of the sentence. After simplifying the representation by employing propositional negation, (3.42d–f) makes it easier to see that the truth-value that we assign to (3.18) is 1 if the function *tall* sends *tina* to 0. In simple sentences such as (3.18) these two perspectives are equivalent. However, the general question of how we should use propositional negation when analyzing the semantics of negative sentences is more complex. Later on in this chapter, and in Exercise 12e below, we briefly touch upon this problem. See also the further reading at the end of this chapter.

PROPOSITIONAL CONJUNCTION AND PREDICATE CONJUNCTION

Another useful propositional operator is *propositional conjunction*. In Chapter 2 we focused on predicate conjunction of adjectives and simple entailments as in (3.43) below:

(3.43) Tina is tall and thin ⇒ Tina is thin.

We also briefly noted the use of the conjunction *and* between sentences, and the very similar entailments that it often leads to. This is illustrated again in (3.44) below.

(3.44) Tina is tall and Tina is thin ⇒ Tina is thin.

For the sentential conjunction in (3.44) we assume the following binary structure:

(3.45) [ [ Tina [ is tall ]] and [ Tina [ is thin ]] ]

In (3.45) the conjunction word *and* combines with the right-hand sentence and forms the constituent *and Tina is thin*. This expression combines with the left-hand sentential conjunct, of type *t*, and together they form a sentence of the same type. Thus, the type we assign to the constituent *and Tina is thin* in (3.45) is a function of type
tt, from truth-values to truth-values. This function is derived by combining the conjunction word and with a sentence of type \( t \), and hence we conclude that the word and in (3.45) is of type \( t(tt) \): a function from truth-values to functions from truth-values to truth-values. We define this denotation as the Curried version of the classical \textit{propositional conjunction} operator \( \land \). Propositional conjunction is a two-place function that maps two truth-values into a single truth-value. In numeric terms, it amounts to \textit{multiplication} between truth-values: the binary function that maps a pair of truth-values to 1 if both of them are 1, and to 0 otherwise. Formally:

\[ (3.46) \text{ For any two truth-values } x \text{ and } y: \text{ the truth-value } x \land y \text{ is } x \cdot y, \]

the multiplication of \( x \) by \( y \).

In (3.47) below, we define our Curried \( t(tt) \) version of propositional conjunction using lambda notation. In order to distinguish this \( t(tt) \) denotation of and from other denotations of this word, we refer to this \( t(tt) \) function as ‘\( \text{AND}^{t} \)’:

\[ (3.47) \text{ AND}^{t} = \lambda x_{t} . \lambda y_{t}. y \land x \]

In words: the denotation of sentential and maps any truth-value \( x \) to the function mapping any truth-value \( y \) to the multiplication \( y \land x \) of \( x \) and \( y \). For example, for structure (3.45), we get:

\[ (3.48) \text{ AND}^{t}([\text{[Tina is thin]]})([\text{[Tina is tall]]}) \]

\[ = [\text{[Tina is tall]]} \land [\text{[Tina is thin]]} \]

The denotation of the second conjunct in (3.45) is used as the first argument of the function \( \text{AND}^{t} \). When \( \text{AND}^{t} \) sends its arguments to the \( \land \) operator, we let it reverse their order, so that the first argument of \( \text{AND}^{t} \) is the second argument of \( \land \). This is innocuous, since \( y \land x \) is the same as \( x \land y \). The reason we reverse the order is merely aesthetic: it is more pleasing to the eye to see a ‘\( y \land x \)’ notation when \( x \) is the denotation of the right-hand conjunct in the sentence.

For a fuller analysis of structure (3.45), see Figure 3.6. The semantic interpretation uses the functions \( \text{AND}^{t} \) for and, and the identity function \( \text{IS} \) for is.

In (3.49) below we use our definition of the denotations \( \text{AND}^{t} \) and \( \text{IS} \) to analyze the truth-value derived in Figure 3.6.
Figure 3.6  Syntactic structure and semantic interpretation for Tina is tall and Tina is thin.

(3.49) a. \( \text{AND}^f((\text{IS}(\text{thin}))(\text{tina}))((\text{IS}(\text{tall}))(\text{tina})) : t \)
\( \text{▷ compositional analysis in Figure 3.6} \)

b. \( = (\text{AND}^f((\text{thin}(\text{tina})))((\text{tall}(\text{tina}))) \)
\( \text{▷ applying IS (identity function)} \)

c. \( = ((\lambda x.(\lambda y. y \land x))(\text{thin}(\text{tina})))((\text{tall}(\text{tina}))) \)
\( \text{▷ definition of AND}^f \)

d. \( = ((\lambda y. y \land \text{thin}(\text{tina})))((\text{tall}(\text{tina}))) \)
\( \text{▷ application to thin(\text{tina})} \)

e. \( = \text{tall}(\text{tina}) \land \text{thin}(\text{tina}) \)
\( \text{▷ application to tall(\text{tina})} \)
As in our example of number addition, we see that lambda notation allows us to use Curried functions while reverting to traditional notation when convenient. The Curried notation in (3.49a–b) is convenient when considering the truth-value derivation from syntactic forms as in Figure 3.6. The standard notation in (3.49e) is convenient if we are interested in the relations between our results and classical logical analyses. However, the two notations are equivalent in terms of the truth-values they represent.

As noted in Chapter 2, the connections between our compositional treatment of conjunction and classical analyses become less straightforward when we consider predicate conjunction. In Chapter 2 we analyzed the conjunction between adjectives in the sentence Tina is tall and thin using set intersection. Given our current emphasis on the use of Curried functions, it is convenient to analyze the same sentence analogously to (3.45), using only binary structures. Thus, we now assume the structure in (3.50) below:

\[(3.50) \quad \text{Tina [ is [ tall [ and thin ]] ]}\]

To mimic our intersective analysis of conjunction, here we let the word and denote a function of type \((\text{et})(\text{et})(\text{et}))\). When we read it in English, this is the type that describes functions from characteristic functions to characteristic functions. You may say gee whiz, but when replacing “characteristic functions” by “sets”, we see that it is just our usual Currying practice. The functions we have just mentioned correspond to functions that map every pair of sets to a set. The intersection operator is such a function: it sends every pair of sets to their intersection. Thus, type \((\text{et})(\text{et})(\text{et}))\) is the proper type for defining an operator on et functions that mimics set intersection.

When we define the denotation of predicate conjunction in (3.50) as a function of type \((\text{et})(\text{et})(\text{et}))\), it is convenient, as we did in the case of predicate negation, to use the corresponding propositional operator. Below we give our denotation of predicate conjunction, denoted ‘AND\text{et}’, using the operator \(\land\) between truth-values:

\[(3.51) \quad \text{AND}^{\text{et}} = \lambda f_{\text{et}}.\lambda g_{\text{et}}.\lambda x_{\text{et}}. g(x) \land f(x)\]
In words: the denotation of the predicate conjunction and maps any et function \( f \) to the function mapping any et function \( g \) to the function mapping any entity \( x \) to \( g(x) \land f(x) \). This definition has the following simple property: when two et functions \( h_1 \) and \( h_2 \) characterize sets of entities \( A_1 \) and \( A_2 \), respectively, the result of applying the function \( \text{AND}^{et} \) to \( h_1 \) and \( h_2 \) is the function characterizing the intersection of \( A_1 \) and \( A_2 \) (see Exercise 12). Thus, treating predicate conjunction using \((\text{et})(\text{et})(\text{et}))\) functions encodes the same analysis of (3.50) as in Chapter 2, where we used set intersection.

To see in more detail how definition (3.51) works, consider the analysis of structure (3.50) in (3.52) below:

\[
\text{(3.52)} \quad \begin{align*}
a. & \quad \text{(IS}((\text{AND}^{et}(\text{thin}))(\text{tall})))\text{(tina)} \quad \triangleright \text{compositional analysis of (3.50)} \\
b. & \quad = ((\text{AND}^{et}(\text{thin}))(\text{tall}))(\text{tina}) \quad \triangleright \text{applying IS (identity function)} \\
c. & \quad = (((f.e.t.\lambda x.e.g(x) \land f(x))(\text{thin}))(\text{tall}))(\text{tina}) \quad \triangleright \text{definition of \text{AND}^{et}} \\
d. & \quad = (((\lambda g.e.\lambda x.e.g(x) \land \text{thin}(x)))(\text{tall}))(\text{tina}) \quad \triangleright \text{application to \text{thin}} \\
e. & \quad = (((\lambda x.e.\text{tall}(x) \land \text{thin}(x)))(\text{tina}) \quad \triangleright \text{application to \text{tall}} \\
f. & \quad = \text{tall}(\text{tina}) \land \text{thin}(\text{tina}) \quad \triangleright \text{application to \text{tina}}
\end{align*}
\]

The truth-value that we end up deriving for sentence (3.50) (Tina is tall and thin) is the same as the one we got for (3.45) (Tina is tall and Tina is thin). Thus, we have captured the intuitive semantic equivalence between these sentences. However, note that simplifications as in (3.49) and (3.52) are only one way of explaining this equivalence, and sometimes they can obscure insights about denotations of constituents. Specifically, in (3.52f) we no longer see that our compositional analysis of predicate conjunction is equivalent to the set intersection of Chapter 2. The direct compositional analysis in (3.49a) highlights this fact more clearly. More generally, we analyze the equivalence between (3.50) and (3.45) as following from elementary considerations about the relationships between set intersection
(in (3.50)) and multiplication of truth-values (in (3.45)). In formula, let $\lambda_A$ and $\lambda_B$ be et functions that characterize the sets of entities $A$ and $B$, respectively. For every entity $x$, we have:

$$x \in A \cap B \text{ if and only if } \lambda_A(x) \cdot \lambda_B(x) = 1$$

For instance, the entity tina is in the intersection of the sets characterized by the et functions tall and thin if and only if both functions send tina to 1. Thus, we can see that we account for the equivalence already in the interpreted structures (3.52a) and (3.49a), without any simplification of $\lambda$-terms. This point is of some historical interest: unlike early analyses of conjunction in the 1960s, our compositional semantics directly interprets the structures of both sentences (3.50) and (3.45). Neither of these structures is assumed to be “deeper” or “more basic” than the other.

Our treatment of adjective conjunction can now be directly used for other predicates, especially verb phrases as in the examples below:

(3.53) Tina smiled and danced. Tina smiled and praised Mary. Tina praised Mary and smiled. Tina praised Mary and thanked John.

(3.54) Tina smiled and praised herself. Tina thanked Mary and praised herself.

These sentences with verb phrase conjunctions also demonstrate an equivalence with sentential conjunction. Consider for instance the following equivalence:

$$\text{Tina thanked Mary and praised herself } \leftrightarrow \text{ Tina thanked Mary and Tina praised herself}$$

Such equivalences are immediately explained by our analysis of propositional conjunction and predicate conjunction. However, it should be noted that the equivalence scheme is not valid in general: it does not necessarily hold with more complex subjects. For instance, the sentence someone is smiling and someone is dancing does not entail someone is smiling and dancing: the existence of people doing two different things does not mean that someone is doing both. We will return to this point in Chapter 4.

As with our treatment of negation, the lambda notation exposes the semantic relation between a propositional operator ($\wedge$) and a
set-theoretical operator (intersection), where the latter is presented by means of characteristic functions. However, in contrast to our treatment of negation, where propositional negation was not directly used in our compositional analysis, now it becomes clear that both propositional conjunction and set intersection are useful as denotations of conjunction: the first has a straightforward use with sentential conjunctions, the latter with predicate conjunction. To conclude, our type system requires the use of two denotations for conjunction: \texttt{AND} for sentential conjunction and \texttt{AND}^* for predicate conjunction. Although the two functions are of different types, namely \((\texttt{t}(\texttt{t}))\) and \((\texttt{e}(\texttt{t})(\texttt{e})(\texttt{t}))\), they are logically related. In fact, the parallelism we have observed only reflects a small part of the semantic relations between \textit{and} conjunctions of different categories in natural language. Similar relations exist between \textit{disjunctive} coordinations with \textit{or} in different categories, and, to a lesser extent, between negation in different categories. These relations are often analyzed as revealing \textit{Boolean structures} in natural language semantics. Relevant details can be found in the further reading at the end of this chapter.

INTERSECTIVE ADJECTIVES AND SUBSECTIVE ADJECTIVES

Let us move on to another example where set intersection and propositional conjunction play a major role: the different usages of adjectives. So far we have only considered adjectives in sentences like \textit{Tina is tall}, where they appear in \textit{predicative} positions, following the copula \textit{is}. However, English also allows adjectives to precede nouns, as in the following sentences:

\begin{enumerate}
  \item [(3.55)] Tina is a \textit{tall} woman; the \textit{tall} engineer visited us; I met five \textit{tall} astronomers.
\end{enumerate}

Occurrences of adjectives before the noun as in (3.55) are often referred to as ‘attributive’, or \textit{modification}. In many cases, we find strong semantic relations between these modificational occurrences and the predicative use. Consider for instance the following equivalences:

\begin{enumerate}
  \item [(3.56)] a. Tina is a \textit{Chinese} pianist ⇔ Tina is \textit{Chinese} and Tina is a pianist.
\end{enumerate}
b. My doctor wears no white shirts ⇔ No shirts that my doctor wears are white.

b. Dan saw six carnivorous animals ⇔ Six animals that Dan saw are carnivorous.

In each of these examples, we see an equivalence between a sentence with a modificational adjective, and another sentence with the same adjective in a predicative position. When analyzing such equivalences, we will concentrate on (3.56a) as a representative example. Before further analyzing the sentence Tina is a Chinese pianist in (3.56a), let us first consider the following simpler sentence:

(3.57) Tina [ is [ a pianist ]]

Given the constituency that we assume in (3.57), we analyze the noun pianist as denoting a function pianist of type et. This is similar to our treatment of intransitive verbs and predicative adjectives. The indefinite article a is analyzed, similarly to the copula is, as denoting the identity function of type (et)(et). Formally:

(3.58) \( A(\text{et})(\text{et}) = \text{IS} = \lambda \text{get}.\text{g} \)

With these assumptions, structure (3.57) is analyzed as denoting the following truth-value.

(3.59) \( \text{IS}(A(\text{pianist}))(\text{tina}) = \text{pianist}(\text{tina}) \)

This truth-value is 1 when the entity tina is in the set characterized by the function pianist, and 0 otherwise. As for the modificational construction in (3.56a), we now assume the following structure:

(3.60) Tina [ is [ a [ Chinese pianist ]]]

We already know how to analyze sentences with predicative adjectives like Tina is Chinese, where we let the adjective denote an et function. It might be tempting to let the modificational occurrence of the adjective Chinese in (3.60) denote the same et function. However, with such an analysis, we would have a problem when treating the constituent Chinese pianist in (3.60). If both the adjective and the noun are assigned the type et, function application would not be
able to combine their denotations. Our solution to this problem is to assume that there are two different denotations of the adjective *Chinese*. One denotation is the arbitrary *et* function *chinese* that we use for the predicative use, e.g. in *Tina is Chinese*. The other denotation is used for modificational occurrences as in (3.60). The modificational denotation is a function of type (*et*)(*et*), from characteristic functions to characteristic functions. This will allow the adjective to combine with nouns like *pianist*, as in (3.60). To highlight its use, we refer to this function as ′*chinese*′. Now, since there is a semantic connection between the two usages of the adjective *Chinese*, we define the (*et*)(*et*) denotation *chinese* on the basis of the *et* function *chinese*. Specifically, we associate *chinese* with the function mapping any set *A* to the intersection of *A* with the *Chinese* entities. In this way, the expression *Chinese pianist* is associated with the set of pianists who are also Chinese. In lambda notation we define the function *chinese* as follows:

\[
\text{chinese}_{(et)(et)} = \lambda f_{et}.\lambda x_{et}.\text{chinese}(x) \land f(x)
\]

The function *chinese* maps any *et* function *f* to the function that maps an entity *x* to 1 if and only if *x* is in both sets that are characterized by the *et* functions *f* and *chinese*. Treating the constituent *Chinese pianist* using this denotation, we get the following analysis of sentence (3.60):

\[
(\text{IS(A(chinese_{mod}(pianist)))(tina)})
\]

▷ compositional analysis of (3.60)

\[
= \text{(chinese}_{mod}(\text{pianist}))(\text{tina})
\]

▷ applying IS and ∨ (identity functions)

\[
= ((\lambda f_{et}.\lambda x_{et}.\text{chinese}(x) \land f(x))(\text{pianist}))(\text{tina})
\]

▷ definition (3.61) of *chinese* 

\[
= (\lambda x_{et}.\text{chinese}(x) \land \text{pianist(x)})(\text{tina})
\]

▷ application to *pianist*

\[
= \text{chinese}(\text{tina}) \land \text{pianist(tina)}
\]

▷ application to *tina*

The analysis in (3.62) immediately accounts for the equivalence we observed in (3.56a). Similar analyses of the modificational usage of
adjectives also account for the equivalences in (3.56b) and (3.56c) with the adjectives white and carnivorous. Because their analysis as modifiers involves intersecting the noun denotation with the predicative adjective’s denotation, adjectives like Chinese, white and carnivorous are referred to as intersective.

Intersective adjectives also support equivalences like the one between (3.63a) and (3.63b) below:

(3.63)  

a. Tina is a Chinese pianist and a biologist.  

b. Tina is a Chinese biologist and a pianist.

We can now easily account for such equivalences as well. Our analysis of the pre-nominal usage of the adjective Chinese is based on set intersection. The occurrences of Chinese in (3.63a) and (3.63b) are interpreted by intersecting the set C of Chinese entities with another set: the set P of pianists, and the set B of biologists, respectively. With these notations, let us consider the following set-theoretical equality:

(3.64)  

\[(C \cap P) \cap B = (C \cap B) \cap P\]

In words: the intersection of the set B with the intersection of C and P is the same as the intersection of P with the intersection of C and B. Having observed this equality, we see that the equivalence in (3.63) follows directly from our analysis of intersective adjectives.

It is important to note that, although intersective interpretations are pretty common, adjectives may also show non-intersective behavior in their modificational use. Unlike what we saw with the adjective Chinese, there are other adjectives that do not support the equivalence pattern in (3.63). Consider for instance the adjective skillful in (3.65) below:

(3.65)  

a. Tina is a skillful pianist and a biologist.  

b. Tina is a skillful biologist and a pianist.

Sentence (3.65a) can be true if Tina is competent as a pianist but amateurish as a biologist. Thus, (3.65a) does not entail (3.65b). For a similar reason, (3.65b) does not entail (3.65a). This lack of equivalence shows that we cannot analyze constructions like skillful pianist or skillful biologist by intersecting the set of pianists/biologists with
some set of “skillful entities”. Such an analysis, together with the set-theoretical equality we saw in (3.64), would lead us to incorrectly expect equivalence in (3.65). Intuitively, we say that the adjective *skillful* shows a *non-intersective* behavior in (3.65). How can we describe this usage?

To solve this problem, we assume that the basic denotation of the adjective *skillful* is of type \((et)(et)\). This immediately allows us to construct models where one of the sentences in (3.65a–b) denotes 1 and the other denotes 0. The reason is that, when the \((et)(et)\) function associated with *skillful* is arbitrary, we no longer assume that the denotations of *skillful pianist* and *skillful biologist* are formed by intersection. Specifically, let us look at a model with the following denotations:

\[
\text{pianist}_{et}: \text{characterizes the singleton set } \{\text{tina}\} \\
\text{biologist}_{et}: \text{characterizes the set } \{\text{tina, mary}\} \\
[[\text{skillful}]_{(et)(et)}(\text{pianist})]: \text{characterizes the singleton set } \{\text{tina}\} \\
[[\text{skillful}]_{(et)(et)}(\text{biologist})]: \text{characterizes the singleton set } \{\text{mary}\}
\]

In words: the only pianist is Tina, the only biologists are Tina and Mary, the only skillful pianist is Tina, and the only skillful biologist is Mary.

In this model, we use the \((et)(et)\) denotation for *skillful*, and not an \(et\) function denotation. Thus, we consider Tina skillful relative to the denotation of *pianist*, but not relative to the denotation of *biologist*. Because our analysis does not use any set of “skillful entities”, the fact that Tina is a biologist in the model does not entail that she is considered a skillful biologist. Thus, sentence (3.65a) denotes 1 whereas (3.65b) denotes 0. This agrees with our intuitions about the lack of entailment between these sentences.

But now we have to treat another property of the adjective *skillful*. Let us consider the following entailment:

(3.66) Tina is a skillful pianist \(\Rightarrow\) Tina is a pianist.

This entailment reflects the obvious intuition that every skillful pianist is a pianist. Adjectives like *skillful* that show this behavior are often
called *subsective adjectives*. An alternative name for these adjectives is ‘restrictive’ or ‘restricting’. The judgment about the validity of (3.66) must lead us to think a little more about the (et)(et) denotation of the adjective *skillful*. This denotation cannot be allowed to be any (et)(et) function. If such arbitrariness were allowed, models might let the denotation of *skillful pianist* be any set of entities they contain, not necessarily a subset of the set of pianists. For instance, a function from sets to set can send the set \{a, b\} to the set \{c, d\}. To avoid such situations, we require that the denotation of *skillful* sends any set \(P\) to a *subset* of \(P\). More formally, we define this general restriction as follows:

\[
(3.67) \text{For any model } M, \text{ the denotation of } \text{skillful} \text{ in } M \text{ is an (et)(et) function } \text{skillful}^{\text{mod}} \text{ that satisfies the following: for every et function } f \text{ in } M, \text{ the set characterized by } \text{skillful}^{\text{mod}}(f) \text{ is a subset of the set characterized by } f. \]

Another possible way to state the restriction, which is elegant though harder to grasp, is to define \(\text{skillful}^{\text{mod}}\) so that it satisfies (3.67) by relying on another, arbitrary, function of type (et)(et). When we denote this function \(\text{skillful}^{\text{arb}}\), the definition of \(\text{skillful}^{\text{mod}}\) reads as follows:

\[
(3.68) \text{skillful}^{\text{mod}} = \lambda f_{et}.\lambda x_e.(\text{skillful}^{\text{arb}}(f))(x) \land f(x)
\]

In words: the function \(\text{skillful}^{\text{mod}}\) sends every et function \(f\) to the characteristic function of the (arbitrary) set characterized by \(\text{skillful}^{\text{arb}}(f)\), intersected with the set characterized by \(f\). If the restriction (3.67) is all that we want our models to specify about the denotation of the adjective *skillful*, then it is equivalent to the definition in (3.68). The (et)(et) function \(\text{skillful}^{\text{arb}}\) is free to vary from one model to another like the other arbitrary denotations we have assumed. The superscript *arb* is a reminder that we assume that the function \(\text{skillful}^{\text{arb}}\) is arbitrary, although it is not the denotation of the word *skillful*.

With the denotation in (3.68), the sentence *Tina is a skillful pianist* is analyzed as follows:
(3.69) \((\text{IS}(\text{skillful}^{\text{mod}}(\text{pianist}))))(\text{tina})\) 

\(\triangleright\) compositional analysis

\[= (\text{skillful}^{\text{mod}}(\text{pianist}))(\text{tina})\]  
\(\triangleright\) applying \text{IS} and \text{A}

\[= (\lambda f. \lambda x. (\text{skillful}^{\text{arb}}(f))(x) \land f(x))(\text{pianist})(\text{tina})\]  
\(\triangleright\) definition (3.68)

\[= \text{skillful}^{\text{arb}}(\text{pianist})(\text{tina}) \land \text{pianist}(\text{tina})\]  
\(\triangleright\) application to \text{pianist} and \text{tina}

The analysis in (3.69) immediately accounts for the entailment in (3.66). At the same time, it also accounts for the lack of entailment in the opposite direction: when Tina is a pianist, it does not follow that she is a skillful pianist. This is easily explained, since when Tina is a pianist the truth-value derived in (3.69) may still be 0. This happens in models where the entity Tina is not in the set characterized by \text{skillful}^{\text{arb}}(\text{pianist}).

We should note that intersective adjectives also show entailments as in (3.66). This is directly accounted for in our analysis. In other words, our treatment of intersective adjectives correctly expects them to be a sub-class of the adjectives we classified as subsective. Below we summarize the concepts of intersective and subsective adjective, and intersective and subsective adjective functions. For convenience, we look at functions from sets of entities to sets of entities.

<table>
<thead>
<tr>
<th>The following entailments define an adjective (A) as being intersective/subsective, where (X) is a proper name and (N) is a common noun:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A) is intersective (-) (X) is a (A) (N) (\iff) (X) is (A) and (X) is a (N)</td>
</tr>
<tr>
<td>e.g. Dan is a Dutch man</td>
</tr>
<tr>
<td>(\iff) Dan is Dutch and Dan is a man</td>
</tr>
<tr>
<td>(A) is subsective (-) (X) is a (A) (N) (\implies) (X) is (N)</td>
</tr>
<tr>
<td>e.g. Dan is a skillful pianist (\implies) Dan is a pianist</td>
</tr>
</tbody>
</table>

For a function \(F\) from \(\wp(E)\) to \(\wp(E)\) we define:

| \(F\) is intersective \(-\) There is a set \(A\), s.t. for every set \(B\): |
| \(F(B) = A \land B.\) |
| \(F\) is subsective \(-\) For every set \(B\): \(F(B) \subseteq B.\) |

By treating the denotations of adjectives as intersective and subsective functions of type \((et)(et)\), we have been able to analyze the behavior of intersective and subsective adjectives.
The examples above show some cases of intersective adjectives like *Chinese*, as well as one example of a non-intersective, subsective adjective, *skillful*. Classifying adjectives into intersective and non-intersective is a highly complex problem, both empirically and theoretically. To see one example of the difficulty, let us reconsider our favorite adjectives *tall* and *thin*. So far, we have assumed that these adjectives denote arbitrary *et* functions. Therefore, in their modificational usages, e.g. in *tall woman* and *thin woman*, we may like to use the intersective analysis. However, this would be questionable: tall children are not necessarily tall; thin hippos are not necessarily thin. On the other hand, treating *tall* and *thin* as subsective adjectives may also lead to complications. For instance, when saying that *Tina is a child and she is tall*, our assertion that Tina is a child affects our understanding of the adjective *tall* in much the same way as it does in the sentence *Tina is a tall child*. Following this kind of observation, many researchers propose that adjectives like *tall* and *thin* should be treated as intersective, while paying more attention to the way they are affected by the context of the sentence. This and other questions about the semantics of adjectives constitute a large body of current research. For some of these problems, see the further reading at the end of this chapter.

The semantic concepts of subsective and intersective functions are useful for other categories besides adjectives. Consider for instance the following entailments:

\[(3.70)\]

\begin{align*}
a. \text{ Tina [smiled [charmingly]]} & \Rightarrow \text{ Tina smiled.} \\
b. \text{ Tina [ran [with John]]} & \Rightarrow \text{ Tina ran.}
\end{align*}

\[(3.71)\]

\begin{align*}
a. \text{ Tina [is [a [pianist [from Rome]]]]} & \Leftrightarrow \text{ Tina is a pianist and Tina is from Rome.} \\
b. \text{ Tina [is [a [pianist [who [praised herself]]]]]} & \Leftrightarrow \text{ Tina is a pianist and Tina praised herself.}
\end{align*}

In sentences (3.70a–b), we see that the adverbial modifiers *charmingly* and *with John* give rise to ‘subsective’ entailments. Furthermore, the adnominal prepositional modifier *from Rome* in (3.71a) and the relative clause *who praised herself* in (3.71b) show ‘intersective’ equivalences. A simple analysis of the adverb *charmingly* may assign it a subsective denotation of type *(et)(et)*. Similarly, the prepositional phrases *with John* and *from Rome* can be analyzed as subsective, or
even intersective, \((et)(et)\) functions. The relative clause *who praised herself* can also be analyzed as an intersective \((et)(et)\) function. By solving Exercise 13, you may analyze these constructions in more detail.

**SUMMARY: LEXICAL DENOTATIONS**

By way of recapitulation, let us take stock of the variety of lexical denotations that we have so far assumed. First, for some words like *is* and *not* we assigned denotations on the basis of a *definition*. The denotations of these words are fully specified by our analysis, with little freedom left for models to change them. These denotations are subdivided into two classes:

1. Denotations like *is* and *herself*: functions that are exclusively defined by means of their workings on other functions, without further definitions or assumptions. Such denotations are functions that can be expressed as ‘pure’ \(\lambda\)-terms, and they are also referred to as **combinators**.

2. Denotations like *not*, *and* and *and\((et)\)*: functions that are defined by means of some additional concepts, e.g. the functions of propositional negation and propositional conjunction. Because these constant denotations rely on truth-values, they are often referred to as **logical**.

Most words whose denotations are combinatorially or logically defined belong in the class that linguists call **function words** or **functional words**. These words are contrasted with **content words**, which are the bulk of the lexicon in all natural languages. We have seen two kinds of denotations for content words:

3. *Arbitrary* denotations, for which no restrictions hold in our models besides those following from their types. We gave such denotations to proper names (*Tina*), common nouns (*pianist*), verbs (*smile*, *praise*) and predicative usages of adjectives.

4. Denotations that are logically or combinatorially defined on the basis of other, arbitrary denotations. This is how we accounted for modificational adjectives, when deriving their denotations from arbitrary denotations.
Table 3.2: Lexical denotations and their restrictions.

<table>
<thead>
<tr>
<th>Denotation</th>
<th>Type</th>
<th>Restrictions</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>tina</td>
<td>et</td>
<td>-</td>
<td>proper name</td>
</tr>
<tr>
<td>smile</td>
<td>et</td>
<td>-</td>
<td>intransitive verb</td>
</tr>
<tr>
<td>praise</td>
<td>e(et)</td>
<td>-</td>
<td>transitive verb</td>
</tr>
<tr>
<td>pianist</td>
<td>et</td>
<td>-</td>
<td>common noun</td>
</tr>
<tr>
<td>chinese</td>
<td>et</td>
<td>-</td>
<td>predicative adjective</td>
</tr>
<tr>
<td>chinese\textsuperscript{mod}</td>
<td>(et)(et)</td>
<td>intersective: $\lambda f_{et}, \lambda x_{e} \cdot$ chinese$(x) \land f(x)$</td>
<td>modificational adjective</td>
</tr>
<tr>
<td>skillful\textsuperscript{mod}</td>
<td>(et)(et)</td>
<td>subsective: $\lambda f_{et}, \lambda x_{e} \cdot$ chinese$(x) \land f(x)$</td>
<td>modificational adjective</td>
</tr>
<tr>
<td>IS</td>
<td>(et)(et)</td>
<td>combinator: $\lambda g_{et}, g$</td>
<td>copula (auxiliary verb)</td>
</tr>
<tr>
<td>A</td>
<td>(et)(et)</td>
<td>combinator: $\lambda g_{et}, g$</td>
<td>indefinite article</td>
</tr>
<tr>
<td>HERSELF</td>
<td>(et)(et)</td>
<td>combinator: $\lambda R_{et}, \lambda x_{e}. R(x)(x)$</td>
<td>reflexive pronoun</td>
</tr>
<tr>
<td>NOT</td>
<td>(et)(et)</td>
<td>logical: $\lambda g_{et}, \lambda x_{e}. \sim(g(x))$</td>
<td>predicate negation</td>
</tr>
<tr>
<td>AND\textsuperscript{t}</td>
<td>(et)(et)</td>
<td>logical: $\lambda x_{t}, \lambda y_{t}. y \land x$</td>
<td>sentential conjunction</td>
</tr>
<tr>
<td>AND\textsuperscript{et}</td>
<td>(et)(et)</td>
<td>logical: $\lambda f_{et}, \lambda g_{et}, \lambda x_{e}. g(x) \land f(x)$</td>
<td>predicate conjunction</td>
</tr>
</tbody>
</table>

The lexical denotations that we assumed, together with their restrictions, are summarized in Table 3.2.

This summary of our restrictions on lexical denotations only scratches the surface of a vast topic: the organization of lexical meanings. Let us briefly mention some of the questions that we have left untreated. Restrictions on lexical meanings that affect entailments do not only involve restricting the possible denotations of single entries. There are also many strong semantic relations between different lexical entries. Consider for instance the following examples:

(3.72) a. Tina danced ⇒ Tina moved.
   b. John is a bachelor ⇒ John is a man and John is not married.

The entailments in (3.72) illustrate that theories of entailment should also constrain the relations between lexical denotations. Entailment (3.72a) can be explained if the set associated with the verb dance is contained in the set for move; (3.72b) is explained when the set for bachelor is contained in the intersection between the set of men and the complement set of the married entities. Our theory should include an architecture that allows encoding such lexical restrictions on denotations more systematically than we have attempted to do here.
The role of syntactic theory in analyzing lexical meanings is another issue that should be further emphasized. For instance, consider the constituent structure Tina [is [not thin]] that we assumed for sentences with predicate negation. This structure made us adopt the (et)(et) type for the word not. Now, suppose that for syntactic reasons we used the structure not [Tina [is thin]]. A treatment of the word not as propositional negation of type tt would then be forthcoming. We conclude that the choice between the types (et)(et) and tt hinges heavily on theoretical syntactic questions about the structure of negation.

Such puzzles are highly challenging for theories about the relations between formal semantics and other parts of grammar, especially lexical semantics and syntactic theory. They constitute a fascinating and very active area of research in linguistics. Some references for works in this area can be found in the further reading below.

You are now advised to solve Exercises 12 and 13 at the end of this chapter.

FURTHER READING


Advanced: For more on type theory and higher-order logics see Thompson (1991); Kamareddine et al. (2004). The Ajdukiewicz Calculus is from Ajdukiewicz (1935). The original idea of Currying appeared in Schönfinkel (1924). Solving type equations is part of the more general problem of type inference in programming languages (Gunter 1992), especially in relation to functional programming (Hutton 2007; Van Eijck and Unger 2010). For an early semantic treatment of reflexive pronouns see Keenan (1989). For more on variable-free semantics, see Jacobson (1999); Keenan (2007); Hendriks (1993); Steedman (1997); Szabolcsi (1987). On the λ-calculus see Barendregt et al. (2013), and, in relation to combinators, Hindley

EXERCISES (ADVANCED: 7, 9, 10, 11, 12, 13)

1. For $D_e = \{u, v, w, m\}$:
   a. Give the functions in $D_{et}$ that characterize: (i) the set $\{u, w\}$; (ii) the empty set; (iii) the complement set of $\{u, w\}$, i.e. $\{u, w\}$, or $D_e - \{u, w\}$.
   b. Give the set that is characterized by the function that sends every element of $D_e$ to 1.

2. a. Give the types for the following English descriptions (cf. (3.5)):
   (i) functions from functions from entities to entities to functions from entities to truth-values;
   (ii) functions from functions from entities to truth-values to entities;
   (iii) functions from functions from entities to truth-values to functions from truth-values to entities;
   (iv) functions from entities to functions from truth-values to functions from entities to entities;
   (v) functions from functions from entities to truth-values to functions from entities to functions from entities to entities;
   (vi) functions from entities to functions from functions from entities to truth-values to functions from truth-values to entities;
   (vii) functions from functions from functions from truth-values to truth-values to functions from truth-values to entities to functions from entities to entities.
   b. Give English descriptions (cf. (3.5)) for the following types: $(et)t, t(te), (tt)e, (e(et))t, e((et)t), (e(et))(e(tt))$.

3. a. Give the functions in $D_{et}$.
   b. For $D_e = \{u, v, w, m\}$, give the functions in $D_{te}$ and $D_{et}$.
   c. For $D_e = \{l, n\}$, give the functions in $D_{(ee)t}$.
4. For each of the following pairs of types say if our function application rule holds. If that’s the case, write the resulting type:

- \( e + e(et) \), \( (et)t + et \),
- \( tt + (ee)(tt) \), \( et + e(tt) \),
- \( (e(et))(et) + ee \), \( e(et) + (e(et))(et) \),
- \( (e(et))(et) + e \), \( e((et)t)(tt) + e((et)t) \),
- \( (e((et)t))(tt) + e \), \( (ee)(et) + e \),
- \( e(e((et)t)) + e(et) + e((et)(et))(e(e(et))) \).

5. a. In a model with \( D_e = \{ t, j, m \} \), suppose that Tina, John and Mary praised Mary, that Tina and Mary praised John, and that nobody else praised anybody else. What should the denotation of the verb \textit{praise} be in this model?

b. Consider sentences of the form \textit{John \[read Mary Moby Dick\]}. We let the \textit{ditransitive verb read} be of type \( e(e(et)) \). Assume that John read Mary \textit{Moby Dick}, Tina read Mary \textit{Lolita}, and nobody else read anything else to anybody else. In such a model, give the \( et \) denotation of the expression \textit{read Mary Moby Dick} and the \( e(et) \) denotation of the expression \textit{read Mary}.

6. a. Solve the following type equations:

- \( tt + X = t(tt) \), \( Y + e = ee \), \( Z + t = et \), \( (et)t + M = e((et)t) \).

b. Each of the following type equations has two solutions. Find them:

- \( t(tt) + X = tt \), \( Y + ee = e \), \( Z + et = t \), \( e((et)t) + M = (et)t \).

c. Complete the general conclusions from your answers to 6a and 6b:

- (i) Equations of the form \( X + y = z \) always have the solution \( X = \_ \).
- (ii) Equations of the form \( X + yz = z \) always have the solutions \( X = \_ \) and \( X = \_ \).

7. a. The sentence structures below introduce “typing puzzles” similar to (3.15). Solve these puzzles and find appropriate types for the underlined words.

- (i) [\textit{Mary}_e [\textit{walked}_{et} \textit{quickly}_{X}]_Y ];
- (ii) [\textit{Mary}_e [\textit{walked}_{et} \textit{in}\_\textit{X}\_\textit{Utrecht}_{e} \textit{Z}]_Y ];
- (iii) [[\textit{the}_{X}\textit{pianist}_{Y}]_e \textit{smiled}_{et}]_t;
- (iv) [[\textit{the}_{X} \textit{skillful}_{Y} \textit{pianist}_{et}]_e \textit{smiled}_{et}]_t
b. Now consider the following puzzle: \[[ no_X man_{et} ]_Y smiled_{et} \].
   Give two solutions for \( Y \). For each solution give the corresponding solution for \( X \).

8. a. Give English descriptions (cf. (3.34)) for the following \( \lambda \)-terms:
   (i) \( \lambda f_{et} . \lambda y_{et} . f(\lambda z_{et}. y) \)
   (ii) \( \lambda x_{et} . \lambda f_{et} . f(x) \)
   (iii) \( \lambda f_{et} . \lambda g_{et} . \lambda x_{et} . g(f(x)) \)
   (iv) \( \lambda e_{et} . \lambda g(e_{et}) . \lambda x_{et} . g(\lambda y_{et} . f(x)) \)

b. Give \( \lambda \)-terms for the following English descriptions:
   (i) the function sending every function \( f_{ee} \) to the function sending every entity \( x \) to the result of applying \( f \) to \( f(x) \);
   (ii) the function sending every function of type \( (ee) e \) to its value on the identity function of type \( ee \);
   (iii) the function sending every function \( R \) of type \( e(et) \) to its inverse, i.e. the function \( R^{-1} \) of type \( e(et) \) that satisfies for every two entities \( x \) and \( y \): \( (R^{-1}(x))(y) = (R(y))(x) \).

9. a. Simplify the following \( \lambda \)-terms as much as possible using function application:
   (i) \( ((\lambda x_{et} . \lambda f_{et} . f(x))(\text{tina}_{et})))(\text{smile}_{et}) \)
   (ii) \( ((\lambda f_{et} . \lambda g_{et} . \lambda x_{et} . g(f(x)))(\text{smile}_{et}))(\text{mary}_{et}) \)
   (iii) \( (((\lambda g(e_{et}) . \lambda x_{et} . \lambda y_{et} . (g(y))(x))(\text{praise}_{et}))(\text{tina}_{et}))(\text{mary}_{et}) \)

b. We analyze sentences like \( \text{Mary is Tina} \) by letting the word \( \text{is} \) denote an \( e(et) \) function \( F \), which is different from the identity function of type \( e(et) \). Define \( F \) as a \( \lambda \)-term, basing your definition on the equality formula \( x = y \), which has the truth-value 1 iff \( x \) and \( y \) are equal. Write the \( \lambda \)-term for the structure \( \text{Lewis Carroll is C. L. Dodgson} \), and then simplify it as much as possible.
c. We analyze the sentence \([\text{Tina [praised [her mother]]}]\) using the following denotations:
\[
\text{HER} = \lambda f_{ee}.\lambda g_{e(e)}(\lambda x_e.(g(f(x)))(x)) \quad \text{and} \quad \text{mother}_{ee} = \text{the function sending each entity } x \text{ to } x' \text{'s mother.}
\]
Assuming these denotations, give the \(\lambda\)-term we derive for the sentence. Then simplify the \(\lambda\)-term you got as much as possible using function application.

10. a. Give English descriptions for the \(\lambda\)-terms:
   (i) \(\lambda y_e.\text{(DOUBLER)}(y)\) (see (3.25))
   (ii) \(\lambda y_e.((\lambda x_e.x)(y))\)

b. Write simpler descriptions for the functions you described in 10a.

c. Complete the following conclusion:
   for any function \(f_{\tau\sigma}\), the function _____ equals \(f\).
   In the \(\lambda\)-calculus, the simplification rule that this conclusion supports is known as eta-reduction.

d. Simplify the following \(\lambda\)-term as much as possible using function application and eta-reductions:
\[
(\lambda f_{e(e)}\lambda g_{e(e)}\lambda x_e\lambda y_e.(g(f(x))(y))((\lambda u_e\lambda z_e.\text{praise}(z))(u)))(\lambda w_t.w)
\]

11. Consider the equivalence between the active sentence \([\text{Mary [praised Tina]}]\) and its passive form \([\text{Tina [[was praised by] Mary]}]\).
   In this structure, we unrealistically assume that the string \(\text{was praised by}\) is a constituent. Express the denotation of this constituent in terms of the denotation \text{praise}, in a way that captures the equivalence between the active and passive sentences.

12. a. Simplify the following \(\lambda\)-term as much as possible using function application and the definition of \(\text{AND}^t\):
\[
(\lambda f_{e(e)}\lambda x_e\lambda y_e.((\lambda f(x)(y))((f(y))(x)))(\lambda u_e\lambda z_e.\text{praise}(z))(u))
\]
   Describe in words the \(e(e)\) function that you got.

b. (i) Give the two binary structures for the sentence \([\text{Tina is not tall and thin}]\).
   (ii) For each of these structures, give the semantic interpretation derived using the lexical denotations \(\text{NOT}_{(e(e))}\) and \(\text{AND}^t\).
   (iii) Simplify the resulting \(\lambda\)-terms as much as possible using function application and the definitions of \(\text{NOT}_{(e(e))}\) and \(\text{AND}^t\).

c. (i) Give binary structures for the sentences in (3.53) and (3.54).
   (ii) For each of these structures, give the semantic interpretation derived using the lexical denotations \(\text{AND}^t\) and \(\text{HERSELF}\).
(iii) Simplify the resulting \( \lambda \)-terms as much as possible using function application and the definitions of \( \text{AND}^\text{et} \) and \( \text{HERSELF} \).

d. For every function \( f \) of type \( \text{et} \), we use the notation \( f^* \) to denote the set of entities \( \{ x \in D_e : f(x) = 1 \} \) that \( f \) characterizes. For instance, for the function \( \chi_S \) in (3.2) we denote: \( \chi_S^* = S = \{ a, c \} \). Show that for all functions \( h_1, h_2 \) of type \( \text{et} \), the following holds:
\[
(\text{AND}^\text{et}(h_2)(h_1))^* = h_1^* \cap h_2^*.
\]

In words: the function \( \text{AND}^\text{et}(h_2)(h_1) \) characterizes the intersection of the sets that are characterized by \( h_1 \) and \( h_2 \).

e. Assume that the sentence \( \text{Tina is not tall} \) has the structure \( \text{not} [\text{Tina is tall}] \). What should the type and denotation of \( \text{not} \) be under this analysis? How would you account for the ambiguity of \( \text{Tina is not tall and thin} \)? Explain all your structural assumptions.

13. a. Account for the entailment (3.70a) with the adverb \( \text{charmingly} \):

describe the restriction on the adverb’s denotation by completing the following sentence:

the function \( \text{charmingly} \) of type ____ maps any set \( A \) characterized by ____ to ____.

b. Account for the same entailment by postulating a \( \lambda \)-term for \( \text{charmingly} \) in terms of an arbitrary function \( \text{charmingly}^{\text{arb}} \).

Simplify the \( \lambda \)-terms for the two sentences in (3.70a), and explain why the \( \leq \) relation must hold between them in every model.

c. (i) Repeat your analysis in 13a, but now for the entailment (3.70b) and the preposition \( \text{with} \). Complete the following sentence:

the function \( \text{with} \) of type ____ maps any entity (e.g. for \( \text{John} \)), to a function mapping any characteristic function \( \chi_A \) (e.g. for \( \text{ran} \)) to ____.

(ii) Repeat your analysis in b, but now for (3.70b). Postulate a \( \lambda \)-term for \( \text{with} \) in terms of an arbitrary function \( \text{with}^{\text{arb}} \). Simplify the \( \lambda \)-terms you get for the sentences in (3.70b).

d. Account for the equivalence (3.71a) by defining the denotation \( \text{from} \) (of which type?) on the basis of an arbitrary function \( \text{from}^{\text{arb}} \) of type \( e(\text{et}) \). Show that after simplifications, the truth-values you get for the two sentences in (3.71a) are the same.
e. Account for the entailments (i) *Tina is very tall* ⇒ *Tina is tall*, and (ii) *Tina is a [very tall student]* ⇒ *Tina is a tall student*: postulate proper restrictions on the denotation of the word *very* in (i) and (ii), of types (et)(et) and ((et)(et))(et)(et)) respectively.

f. Account for the equivalence in (3.71b) by postulating a proper type and a proper restriction on the denotation of the word *who*. Give it a \( \lambda \)-term. Show that after simplifications, the truth-values you get for the two sentences in (3.71b) are the same.

**SOLUTIONS TO SELECTED EXERCISES**

1. a. \([u \mapsto 1, v \mapsto 0, w \mapsto 1, m \mapsto 0]; [u \mapsto 0, v \mapsto 0, w \mapsto 0, m \mapsto 0]; [u \mapsto 0, v \mapsto 1, w \mapsto 0, m \mapsto 1].\)
   
   b. \(\{u, v, w, m\}\), i.e. the whole domain \(D_e\).

2. a. \((i) (ee)(et), (ii) (et)e, (iii) (et)(te), (iv) e(t(ee)), (v) (et)(e(ee)), (vi) (e(et))(te), (vii) ((tt)(te))(e(ee)).\)

3. a. \(D_{el} = [[[0 \mapsto 0, 1 \mapsto 0], [0 \mapsto 0, 1 \mapsto 1], [0 \mapsto 1, 1 \mapsto 0], [0 \mapsto 1, 1 \mapsto 1]].\)
   
   b. \(D_{et} = \)

   \([u \mapsto 0, v \mapsto 0, w \mapsto 0, m \mapsto 0], [u \mapsto 0, v \mapsto 0, w \mapsto 0, m \mapsto 1], [u \mapsto 0, v \mapsto 0, w \mapsto 1, m \mapsto 0], [u \mapsto 0, v \mapsto 0, w \mapsto 1, m \mapsto 1], [u \mapsto 0, v \mapsto 1, w \mapsto 0, m \mapsto 0], [u \mapsto 0, v \mapsto 1, w \mapsto 0, m \mapsto 1], [u \mapsto 0, v \mapsto 1, w \mapsto 1, m \mapsto 0], [u \mapsto 0, v \mapsto 1, w \mapsto 1, m \mapsto 1], [u \mapsto 1, v \mapsto 0, w \mapsto 0, m \mapsto 0], [u \mapsto 1, v \mapsto 0, w \mapsto 0, m \mapsto 1], [u \mapsto 1, v \mapsto 0, w \mapsto 1, m \mapsto 0], [u \mapsto 1, v \mapsto 0, w \mapsto 1, m \mapsto 1], [u \mapsto 1, v \mapsto 1, w \mapsto 0, m \mapsto 0], [u \mapsto 1, v \mapsto 1, w \mapsto 0, m \mapsto 1], [u \mapsto 1, v \mapsto 1, w \mapsto 1, m \mapsto 0], [u \mapsto 1, v \mapsto 1, w \mapsto 1, m \mapsto 1].\)

   The solution for \(D_{ie}\) is along similar lines.

4. \(et, t, \neg, \neg, et, \neg, tt, \neg, \neg, e(et).\)

5. a. \(\text{praise} = t \mapsto [t \mapsto 0 \; j \mapsto 0 \; m \mapsto 0] \; j \mapsto [t \mapsto 1 \; j \mapsto 0 \; m \mapsto 1] \; m \mapsto [t \mapsto 1 \; j \mapsto 1 \; m \mapsto 1].\)
6. a. \([\text{[read Mary Moby Dick]}] = [t\mapsto 1 \ j\mapsto 0 \ m\mapsto 0 \ lo\mapsto 0]\)
   \([\text{[read Mary]}] = t\mapsto [t\mapsto 0 \ j\mapsto 0 \ m\mapsto 0 \ lo\mapsto 0].\)
   \(j\mapsto [t\mapsto 0 \ j\mapsto 0 \ m\mapsto 0 \ lo\mapsto 0].\)
   \(m\mapsto [t\mapsto 0 \ j\mapsto 0 \ m\mapsto 0 \ lo\mapsto 0].\)
   \(lo\mapsto [t\mapsto 1 \ j\mapsto 0 \ m\mapsto 0 \ lo\mapsto 0]].\)

b. \(\lambda\) \(X\) for \(X\) of type \((et)t\) sends every \(et\) function to \(X\).

7. a. (i) \(X=(et)(et)\) \(X=e((et)(et))\),
   (ii) \(X=(et)e\)
   (iv) \(Y=(et)(et),\)
   (v) \(X=(et)e\)
   (vi) \(Y=(et)((et)(et)),\)
   (vii) \(X=t(tt)\)
   (viii) \(X=t((et)(et)).\)

b. \(Y=e\) and \(X=(et)e; Y=(et)t\)
   and \(X=(et)((et)t).\)

c. \(X=e\) or \((et)t; Y=e((et)(et)); Z=(et)(et).\)

Additional solutions for 7c:
\(X=et, Y=e((et)(((et)(et)e)), Z=(et)(((et)(et))e); X=e,\)
\(Y=e((et)(((et)(et))(et)), Z=(et)(((et)(et))(et)).\)

8. a. (i) the function sending every function \(f\) of type \((et)t\) to the function sending every truth-value \(y\) to the result of applying \(f\) to the constant \(et\) function sending every entity to \(y\).
(ii) the function \(I\) sending every entity \(x\) to the function sending every \(et\) function \(f\) to the truth-value result of applying \(f\) to \(x\) \((I\) sends every \(x\) to the characteristic function of the set of functions characterizing subsets of \(D_x\) containing \(x\).
(iii) the function \(C\) sending every \(et\) function \(f\) to the function sending every \(tt\) function \(g\) to the function from entities \(x\) to the result of applying \(g\) to \(f(x)\) \((C\) returns the function composition of \(g_{tt}\) on \(f_{et}\)).
(iv) the function sending every \(ee\) function \(f\) to the function sending every \((ee)t\) function \(g\) to the function from entities \(x\) to the result of applying \(g\) to the constant function sending every entity to \(f(x)\).

b. (i) \(\lambda f_{ee}.\lambda x.e.f(f(x));\)
   (ii) \(\lambda f_{ee}.e.f(\lambda x.e.x);\)
   (iii) \(\lambda R_{ee}.\lambda x.e.\lambda y.e.(R(y))(x).\)

9. a. (i) \(\text{smile(tina)}\);
   (ii) \(\lambda x.e.\text{smile}(x);\)
   (iii) \(\text{praise(mary)}(\text{tina})\)

b. \(F = \lambda y.e.\lambda x.e.x = y; (F (\text{cld}))(\text{lc}) = (\text{lc} = \text{cld})\)
c. \(((\text{HER}(\text{mother}))(\text{praise}_{\text{et}}))(\text{tina}_e) = \text{praise}(\text{mother}(\text{tina}))(\text{tina})

10. a. (i) the function sending every number \(y\) to \(\text{DOUBLER}(y)\)
(ii) the function sending every entity \(y\) to the result that the function sending every entity \(x\) to \(x\) returns for \(y\)
b. (i) \(\text{DOUBLER}\)
(ii) the function sending every entity \(y\) to \(y\), a.k.a. the function sending every entity \(x\) to \(x\)
c. the function \(\lambda x . f (x)\) equals \(f\)
d. \(\text{praise}\).

11. \([\text{was praised by}] = \lambda x_e . \lambda y_e . \text{praise}(y)(x)\)

12. a. \(\lambda x_e . \lambda y_e . (\text{praise}(y))(x) \land (\text{praise}(x))(y)\) – the function sending \(x\) to the function sending \(y\) to 1 iff \(x\) and \(y\) praised each other.
b. (i) Tina [is [not [tall [and thin]]]]; Tina [is [[not tall] [and thin]]]
(ii) \((\text{IS}(\text{NOT}((\text{AND}\text{et} (\text{tall}))(\text{tall})))))(\text{tina})\);
(iii) \((\text{IS}((\text{AND}\text{et} (\text{tall}))(\text{NOT}(\text{tall})))))(\text{tina})\)
d. Suppose \(h_1^* = A_1, h_2^* = A_2\). By def. of \(\text{AND}^e\):
\((\text{AND}^e(h_2)(h_1))^* = (\lambda x_e . h_1(x) \land h_2(x))^*,\) i.e. the set \(A = \{x \in E : (h_1(x) \land h_2(x)) = 1\}\). By def. of \(\land\), for every \(y \in E: y \in A\) iff \(h_1(y) = 1\) and \(h_2(y) = 1\), i.e. \(y \in A_1\) and \(y \in A_2\), i.e. \(y \in A_1 \cap A_2\).

d. \(\lambda x_e . \lambda y_e . (\text{praise}_e)(x)(y) \land f(y)\).

d. \(\lambda x_e . \lambda y_e . (\text{from}_e)(x)(y) \land f(y)\).

f. \(\text{WHO}\) is assigned type \(e(\text{et})(\text{et})\), which leads to the term:
\((\text{IS}(\text{A}((\text{WHO}(\text{HERSELF}(\text{praise})(\text{pianist})))(\text{tina}))))(\text{tina})\). With the assumption \(\text{WHO} = \text{AND}^e\), this term is simplified to:
\(\text{pianist}(\text{tina}) \land \text{praise}(\text{tina})(\text{tina})\). We get the same term when simplifying the term for the sentence \(\text{Tina is a pianist and Tina praised herself}\).