Elements of Formal Semantics

An Introduction to the Mathematical Theory of Meaning in Natural Language

Yoad Winter

Open Access Materials: Chapter 3

Elements of Formal Semantics introduces some of the foundational concepts, principles and techniques in formal semantics of natural language. It is intended for mathematically-inclined readers who have some elementary background in set theory and linguistics. However, no expertise in logic, math, or theoretical linguistics is presupposed. By way of analyzing concrete English examples, the book brings central concepts and tools to the forefront, drawing attention to the beauty and value of the mathematical principles underlying linguistic meaning.

See webpage below for further materials and information:

http://www.phil.uu.nl/~yoad/efs/main.html

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 χ_S, is a *function from entities to truth-values*. For each of the two elements in S, entities a and c, the function χ_S returns the truth-value 1. For entity b, which is not in S, we let χ_S return 0. Thus, χ_S is the following function:

(3.2) $\chi_S : \mathsf{a} \mapsto 1 \quad \mathsf{b} \mapsto 0 \quad \mathsf{c} \mapsto 1$

The function χ_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 χ_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, χ_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 *smile* is an arbitrary characteristic function. This corresponds to our assumption that the verb *smile* can be associated with *any* set of entities. Suppose that the denotation of *smile* is the function **smile**. In our analysis of sentence (3.1), this function applies to the entity denotation **tina**. In a formula, sentence (3.1) is analyzed as follows:

(3.3) **smile**(**tina**)

The expression in (3.3) describes the truth-value that the function **smile** assigns to the entity **tina**. For example, in the model we described above, the denotation **smile** is the function χ_S in (3.2). Suppose that in the same model, the denotation **tina** is the entity a. As a result, the denotation (3.3) of sentence (3.1) is $\chi_S(\mathsf{a})$, which equals 1. If **tina** is the entity b, the denotation (3.3) equals 0. This way, by letting the denotation of the verb *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) The domain of characteristic functions in a model M is the set of all the functions from D_e^M 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}.

Subset of D_e	**Characteristic function in D_{et}**
$\emptyset$	f_1 : $\mathsf{a} \mapsto 0$ $\mathsf{b} \mapsto 0$ $\mathsf{c} \mapsto 0$
{a}	f_2 : $\mathsf{a} \mapsto 1$ $\mathsf{b} \mapsto 0$ $\mathsf{c} \mapsto 0$
{b}	f_3 : $\mathsf{a} \mapsto 0$ $\mathsf{b} \mapsto 1$ $\mathsf{c} \mapsto 0$
{c}	f_4 : $\mathsf{a} \mapsto 0$ $\mathsf{b} \mapsto 0$ $\mathsf{c} \mapsto 1$
{a, b}	f_5 : $\mathsf{a} \mapsto 1$ $\mathsf{b} \mapsto 1$ $\mathsf{c} \mapsto 0$
{a, c}	f_6 : $\mathsf{a} \mapsto 1$ $\mathsf{b} \mapsto 0$ $\mathsf{c} \mapsto 1$
{b, c}	f_7 : $\mathsf{a} \mapsto 0$ $\mathsf{b} \mapsto 1$ $\mathsf{c} \mapsto 1$
{a, b, c}	f_8 : $\mathsf{a} \mapsto 1$ $\mathsf{b} \mapsto 1$ $\mathsf{c} \mapsto 1$

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 χ_S, which characterizes the set $S = \{\mathsf{a}, \mathsf{c}\}$. 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 χ_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 τ and σ and domains D_τ and D_σ 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_τ to D_σ. 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 $\mathcal{T}$ that satisfies:*

(i) $\{e, t\} \subseteq \mathcal{T}$

(ii) If τ and σ are types in $\mathcal{T}$ then $(\tau\sigma)$ is also a type in $\mathcal{T}$.

This inductive definition specifies the set of types as an infinite set $\mathcal{T}$, 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 $\mathcal{T}$. Let us summarize how this is done. The domain D_e^M is directly specified by the model. The domain D_t is fixed as $\{0, 1\}$ for

$e, t,$
$ee, tt, et, te,$
$e(ee), e(tt), e(et), e(te), t(ee), t(tt), t(et), t(te),$
$(ee)e, (tt)e, (et)e, (te)e, (ee)t, (tt)t, (et)t, (te)t,$
$(ee)(ee), (ee)(tt), (ee)(et), (ee)(te), (tt)(ee), (tt)(tt), (tt)(et), (tt)(te)$

Figure 3.1 Examples for types.

all models. Domains for other types are inductively defined, as formally summarized in Definition 2 below:

Definition 2. *For all types τ and σ 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_τ to D_σ.*

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) $D_{e(et)}$ is the set of functions from D_e to D_{et}
= the functions from entities to D_{et}
= the functions from entities to the functions from D_e to D_t
= 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:

$$\mathbf{smile}_{et} + \mathbf{tina}_{e} \;=\; \mathbf{smile}(\mathbf{tina}) : t$$

By the notation '$\mathbf{smile}_{et}$' we refer to the denotation of the verb *smile*, and state that it is of type et. Similarly for '$\mathbf{tina}_{t}$'. An alternative notation is '**smile** : et' and '**tina** : e'. This colon becomes more convenient when we wish to state the type t of the result **smile**(**tina**), and write '**smile**(**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:

$$\mathbf{tina}_{e} + \mathbf{smile}_{et} \;=\; \mathbf{smile}(\mathbf{tina}) : t.$$

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 τ gives an object $f(x)$ of type σ. In short:*

Types: $(\tau\sigma) + \tau \;= \tau \quad + (\tau\sigma) = \sigma$

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:

$e + ee = e$	applying a function g_{ee} to an entity x_e gives an entity $g(x)$
$e(et) + e = et$	applying a function $h_{e(et)}$ to an entity x_e gives an et function $h(x)$
$et + (et)(et) = et$	applying a function $F_{(et)(et)}$ to a function k_{et} 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 *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 *transitive sentences*. In (3.6) we standardly assume that a transitive verb (*praise*) forms a constituent with the object noun phrase (*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 *praised Mary*. To do that, we follow our treatment of intransitive verbs. In the same way that the denotation of the verb *smiled* characterizes the set of entities that smiled, we now want the denotation of the verb phrase *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 et function from the denotations of the words *praised*

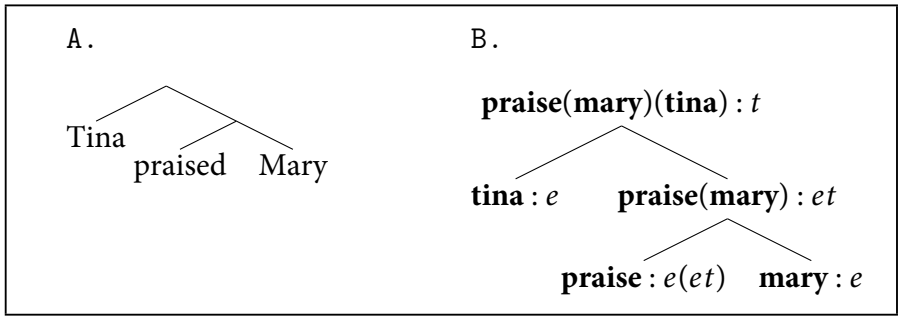

Figure 3.2 Syntactic structure and semantic interpretation for Tina praised Mary.

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 **praise**. Applying this function to the entity **mary**, we get a denotation of type et for the verb phrase *praised Mary*. This is formally written as follows:

(3.7) $\mathbf{praise}_{e(et)} + \mathbf{mary}_{e} \;=\; \mathbf{praise}(\mathbf{mary}) : et$

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

(3.8) $\mathbf{praise}(\mathbf{mary}) + \mathbf{tina}_{e} \;=\; (\mathbf{praise}(\mathbf{mary}))(\mathbf{tina}) : t$

In words: when the et function **praise**(**mary**) applies to the entity **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) **praise**(**mary**)(**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* between entities, 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:

(3.10) $U = \{\langle \mathsf{t}, \mathsf{m}\rangle, \langle \mathsf{m}, \mathsf{t}\rangle, \langle \mathsf{m}, \mathsf{j}\rangle, \langle \mathsf{m}, \mathsf{m}\rangle\}$

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:

(3.11)
a. Who praised Tina? Answer: only Mary.
b. Who praised John? Answer: only Mary.
c. Who praised Mary? Answer: only Tina and Mary herself.

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

$e(et)$ denotation of *praise* also tells us, for each entity, which entities praised that entity. Thus, when our domain of entities D_e is the set {t, j, m}, any $e(et)$ function over this domain answers precisely the same questions as in (3.11). In particular, the same situation that the binary relation U encodes is also described by the following $e(et)$ function in our model, which we call χ_U:

(3.12)
$$\begin{array}{lllll} \chi_U : & \mathsf{t} & \mapsto & [\mathsf{t} \mapsto 0 & \mathsf{j} \mapsto 0 \quad \mathsf{m} \mapsto 1] \\ & \mathsf{j} & \mapsto & [\mathsf{t} \mapsto 0 & \mathsf{j} \mapsto 0 \quad \mathsf{m} \mapsto 1] \\ & \mathsf{m} & \mapsto & [\mathsf{t} \mapsto 1 & \mathsf{j} \mapsto 0 \quad \mathsf{m} \mapsto 1] \end{array}$$

The function χ_U maps each of the three entities in D_e to an et function. More specifically:

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

Note the parallelism between this specification of χ_U and the question–answer pairs in (3.11). When the denotation **praise** is χ_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 sum. Thus, we use the following notation:

$$sum(x, y) = x + y$$

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 $n(nn)$: a function from numbers to functions from numbers to numbers. Let us refer to this function as ADD. To define 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 ADD we will now specify the *nn* function that ADD assigns to any number. For every number y, we define:

(3.13) The *nn* function ADD(y) sends every number x to the number $sum(x, y)$.

As we expect from number addition, the function 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 ADD(y), and this function applies to another number x and returns the sum $sum(x, y)$, or more simply: $x + y$. As a result, for every two numbers x and y, we get:

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

For example, let us consider how we calculate the sum of 1 and 5 using the function ADD. We first give ADD the number 1 as an argument, and get the function 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 ADD(1) the argument 5. Unsurprisingly, the result of calculating (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 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?

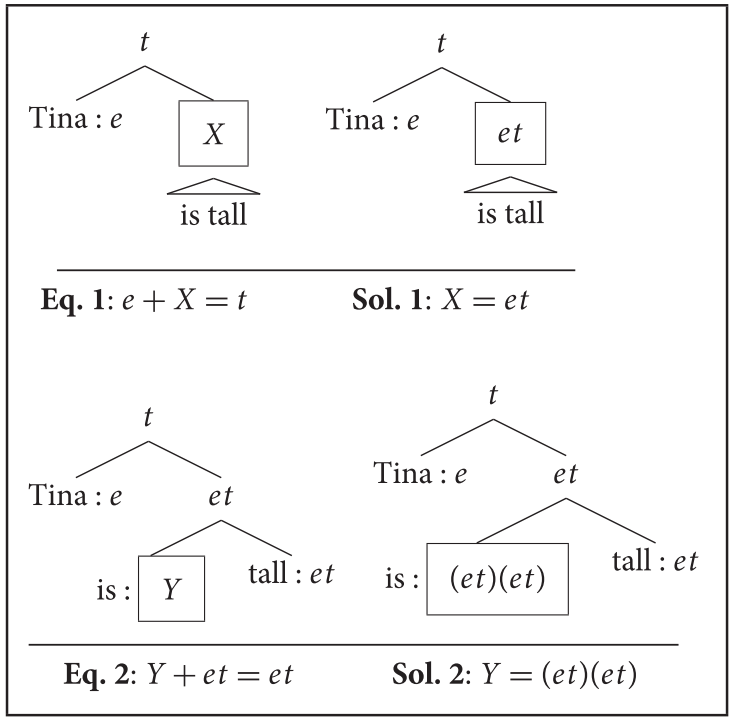

Figure 3.3 Solving type equations for the sentence Tina is tall.

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 $\mathbf{tall}_{et}$ 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) IS is the function sending every function g of type et to g itself.

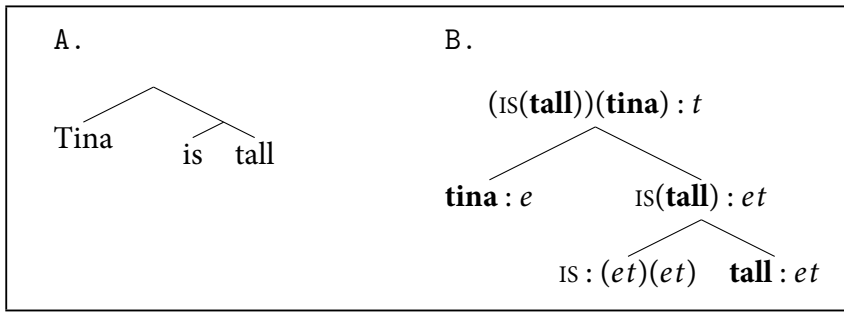

Figure 3.4 Syntactic structure and semantic interpretation for Tina is tall.

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. IS(**tall**) = **tall** (by definition of IS in (3.16))
b. (IS(**tall**))(**tina**) = **tall**(**tina**) (due to the equality in (3.17*a*))

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 $(et)(et)$ 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 $(et)(et)$ function, of the same type as the copula *is*. Our definition of the denotation NOT in (3.19) below respects the status of the word *not* in (3.18) as a predicate negator:

(3.19) NOT is the $(et)(et)$ function sending every et function g to the et function NOT(g) that satisfies the following, for every entity x (see next page):

$$(\text{NOT}(g))(x) = \begin{cases} 1 & \text{if } g(x) = 0 \\ 0 & \text{if } g(x) = 1 \end{cases}$$

In words: we define the function NOT, which takes a function g of type et and returns a function NOT(g) of the same type, separating into the following two cases:

- for any entity x such that $g(x)$ is 0, we define (NOT(g))(x) to be 1;
- for any entity x such that $g(x)$ is 1, we define (NOT(g))(x) to be 0.

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

(3.20) $(\text{IS}_{(et)(et)}(\text{NOT}_{(et)(et)}(\mathbf{tall}_{et})))(\mathbf{tina}_{e})$

Because the function IS is the identity function, the truth-value in (3.20) is the same as:

$(\text{NOT}_{(et)(et)}(\mathbf{tall}_{et}))(\mathbf{tina}_{e})$

By definition of the function NOT, this truth-value is 1 if and only if the entity **tina** is in the complement of the set characterized by the et function **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 'λ-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 "λg_{et}".
- Instead of "to g itself", we write "$.g$".

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

(3.22) $\lambda g_{et}.g$

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 'λ' 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 'g_{et}'

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 λ-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) $\text{IS} = \lambda g_{et}.g$

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 τ is a type, we mean:*
"the function sending every element x of the domain D_τ to φ".

The expression φ 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 φ was simply the argument g itself. However, in general, any mathematical expression φ that describes an object in one of our domains would be appropriate in such a λ-term. The type of the object that φ 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
$\lambda f_{et}.\mathbf{tina}_e$	the $(et)e$ function sending every et function f to the entity **tina**
$\lambda h_{et}.h(\mathbf{tina}_e)$	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 φ describes an object of type σ, the whole λ-term $\lambda x_\tau.\varphi$ describes a function of type $\tau\sigma$: from objects in D_τ to objects in D_σ.

FUNCTION APPLICATION WITH LAMBDA TERMS

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

(3.24) a. IS($\mathbf{tall}_{et}$)
b. $= (\lambda g_{et}.g)$(**tall**)
c. = **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 **tall**" by writing more simply and concretely "**tall**". Function application with λ-terms always involves this sort of concretization. Suppose that we have a function f described by the instruction:

> "for every x of type τ 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 φ in a λ-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 φ is the argument g itself. However, φ 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:

> DOUBLER is the function from numbers to numbers sending every number x to $2 \cdot x$.

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

(3.25) $\text{DOUBLER}_{nn} = \lambda x_n.2 \cdot x$

Having a name like DOUBLER for this function is convenient when we commonly want to double numbers. However, when using λ-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) $(\lambda x_n.2 \cdot x)(17)$

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 λ-term $\lambda x_n.2 \cdot x$. In sum, we get:

(3.27) $(\lambda x_n.2 \cdot x)(17) = 2 \cdot 17$

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_τ *is equal to the value of the expression* φ*, with all occurrences of* x *replaced by* a.

Consider how this convention works in (3.27). The expression 'φ' 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 λ-term for defining

the function ADD that we introduced in (3.13):

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

As in (3.13), this λ-term defines the function 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 ADD to the values 1 and 5, we get the following simplifications:

$$(3.28)\quad ((\lambda y_n.\lambda x_n.y + x)(1))(5) = (\lambda x_n.1 + x)(5) = 1 + 5$$

The simplification in (3.28) involves two steps. First, when the function 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 λ-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 φ. 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 λ-terms for actual calculations, we need a more general rule for simplifying λ-terms under function application. This rule is known as *beta-reduction*, and it constitutes part of the rule system for computing λ-term equivalences, known as the *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 λ-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 *reflexive pronouns* like *herself* or *himself*. Consider the following sentence:

(3.29) Tina [praised herself]

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

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

In words, (3.30) is the truth-value that results from applying the denotation of *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 **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 **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 **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.

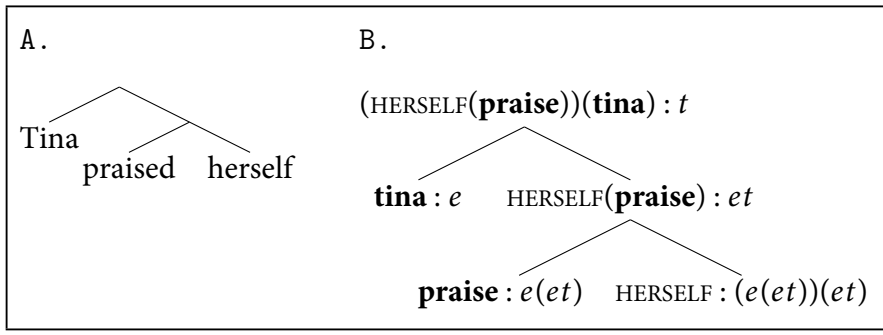

Figure 3.5 Syntactic structure and semantic interpretation for Tina praised herself.

The analysis in Figure 3.5 does not yet define the $(e(et))(et)$ function that the pronoun 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) $[\![\textit{praised herself}]\!](\mathbf{tina}) = \mathbf{praise}(\mathbf{tina})(\mathbf{tina})$

In words: the one-place predicate for *praised herself* holds of the entity **tina** if and only if the two-place predicate **praise** holds of the pair ⟨**tina**, **tina**⟩.

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

(3.32) $(\text{HERSELF}_{(e(et))(et)}(\mathbf{praise}_{e(et)}))(\mathbf{tina})$
$= [\![\textit{praised herself}]\!](\mathbf{tina})$
$= \mathbf{praise}(\mathbf{tina})(\mathbf{tina})$

In words: when the function HERSELF applies to the denotation **praise**, the result is the et function $[\![\textit{praised herself}]\!]$ in (3.31) above. As we saw, this resulting et function sends the entity **tina** to 1 if and only if the two-place predicate **praise** holds of the pair ⟨**tina**, **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:
$(\text{HERSELF}_{(e(et))(et)}(R))(x) = R(x)(x)$

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

(3.34) $\text{HERSELF}_{(e(et))(et)}$ 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 λ-term for rewriting it:

(3.35) $\text{HERSELF}_{(e(et))(et)}$
$= \lambda R_{e(et)}$.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) $\text{HERSELF}_{(e(et))(et)}$
$= \lambda R_{e(et)}.(\lambda x_e . R(x)(x))$

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 λ-terms alone, with some notes for clarification, but without any

description of functions in natural language:

(3.38) a. $(\text{HERSELF}_{(e(et))(et)}(\mathbf{praise}_{e(et)}))(\mathbf{tina}_e)$
▷ compositional analysis (Figure 3.5)

b. $= ((\lambda R_{e(et)}.\lambda x_e.R(x)(x))(\mathbf{praise}))(\mathbf{tina})$
▷ definition (3.37) of HERSELF

c. $= (\lambda x_e.\mathbf{praise}(x)(x))(\mathbf{tina})$
▷ application to **praise**

d. $= \mathbf{praise}(\mathbf{tina})(\mathbf{tina})$ ▷ application to **tina**

Our compositional analysis assigns the verb phrase *praised herself* the denotation HERSELF(**praise**), of type et. This is the same type we assigned to the intransitive verb *smiled* and to the verb phrase *praised Mary*. As we see in (3.38b–c), with our definition of the function HERSELF, the denotation we get for *praised herself* is the et function $\lambda x_e.\mathbf{praise}(x)(x)$, which characterizes the set of self-praisers, as intuitively required by sentence (3.29). By letting all verb phrases denote 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 *smiled and praised herself*, as we will do in (3.53)–(3.54) below, and in Exercise 12c.

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

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 *Tina*, *smile*, *tall* and *praise*, we assume arbitrary denotations of the type we assign. By contrast, when analyzing functional words like *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.

PROPOSITIONAL NEGATION AND PREDICATE NEGATION

Let us now get back to the negation in sentence (3.18) (*Tina is not tall*). We analyzed this sentence using definition (3.19), which describes

the set complement operation using characteristic functions. Using λ-terms we restate this definition as follows:

$$(3.39)\quad \textsc{not} \;=\; \lambda g_{et}.\lambda x_e.\begin{cases} 1 & \text{if } g(x) = 0 \\ 0 & \text{if } g(x) = 1 \end{cases}$$

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 '∼'. For its formal definition we can use subtraction, as stated in (3.40) below:

$$(3.40)\quad \sim \;=\; \lambda x_t.1 - x$$

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

$$(3.41)\quad \textsc{not} \;=\; \lambda g_{et}.\lambda x_e.{\sim}(g(x))$$

The three definitions (3.19), (3.39) and (3.41) are equivalent, and they all boil down to the analysis of the word *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 *Tina is not tall* as in (3.42) below.

(3.42) a. $(\textsc{is}_{(et)(et)}(\textsc{not}_{(et)(et)}(\mathbf{tall}_{et})))(\mathbf{tina}_e)$
▷ compositional analysis of structure (3.18)

b. $= ((\lambda g_{et}.g)(\textsc{not}(\mathbf{tall})))(\mathbf{tina})$
▷ definition of IS as identity function

c. $= (\textsc{not}(\mathbf{tall}))(\mathbf{tina})$
▷ application to NOT(**tall**)

d. $= ((\lambda g_{et}.\lambda x_e.{\sim}(g(x)))(\mathbf{tall}))(\mathbf{tina})$
▷ definition (3.41) of NOT

e. $= ((\lambda x_e.{\sim}(\mathbf{tall}(x))))(\mathbf{tina})$
▷ application to **tall**

f. $= {\sim}(\mathbf{tall}(\mathbf{tina}))$ ▷ application to **tina**

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 *propositional conjunction* operator $\wedge$. Propositional conjunction is a two-place function that maps two truth-values into a single truth-value. In numeric terms, it amounts to *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) For any two truth-values x and y: the truth-value $x \wedge y$ 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 'AND^t':

(3.47) $\text{AND}^t = \lambda x_t.\lambda y_t.y \wedge x$

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

(3.48) $\text{AND}^t([\![\textit{Tina is thin}]\!])([\![\textit{Tina is tall}]\!])$
$= [\![\textit{Tina is tall}]\!] \wedge [\![\textit{Tina is thin}]\!]$

The denotation of the second conjunct in (3.45) is used as the first argument of the function AND^t. When AND^t sends its arguments to the $\wedge$ operator, we let it reverse their order, so that the first argument of AND^t is the second argument of $\wedge$. This is innocuous, since $y \wedge x$ is the same as $x \wedge y$. The reason we reverse the order is merely aesthetic: it is more pleasing to the eye to see a '$y \wedge x$' 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 AND^t for *and*, and the identity function IS for *is*.

In (3.49) below we use our definition of the denotations AND^t and IS to analyze the truth-value derived in Figure 3.6.

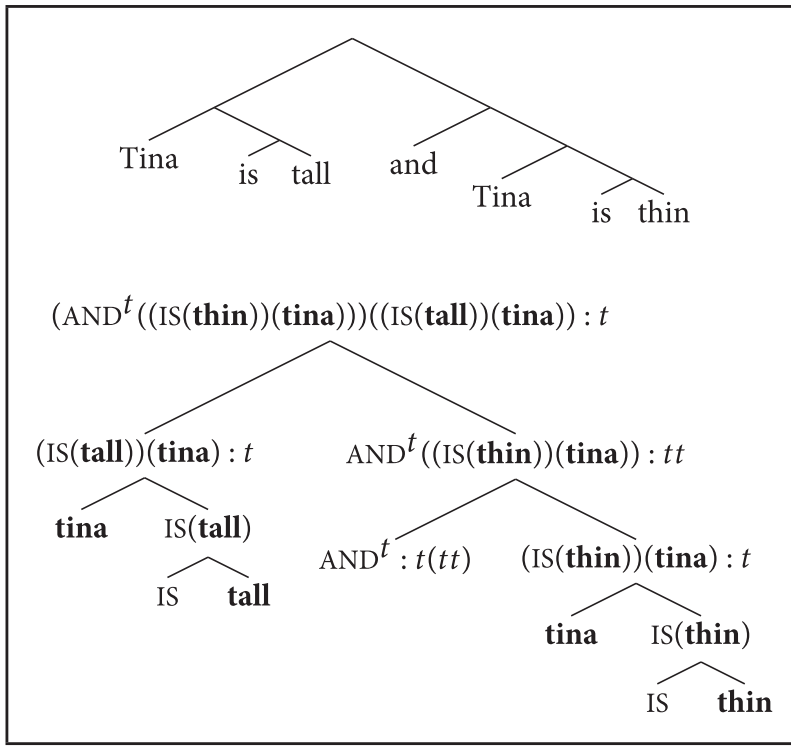

Figure 3.6 Syntactic structure and semantic interpretation for Tina is tall and Tina is thin.

(3.49) a. (AND^t((IS(**thin**))(**tina**)))((IS(**tall**))(**tina**))
▷ compositional analysis in Figure 3.6

b. = (AND^t(**thin**(**tina**)))(**tall**(**tina**))
▷ applying IS (identity function)

c. = $((\lambda x_t.\lambda y_t.y \wedge x)$(**thin**(**tina**)))(**tall**(**tina**))
▷ definition of AND^t

d. = $((\lambda y_t.y \wedge$ **thin**(**tina**)))(**tall**(**tina**))
▷ application to **thin**(**tina**)

e. = **tall**(**tina**) $\wedge$ **thin**(**tina**)
▷ application to **tall**(**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) Tina [is [tall [and thin]]

To mimic our intersective analysis of conjunction, here we let the word *and* denote a function of type $(et)((et)(et))$. When we read it in English, this is the type that describes *functions from characteristic functions to 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 $(et)((et)(et))$ 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 $(et)((et)(et))$, 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^{et}', using the operator $\wedge$ between truth-values:

(3.51) $\text{AND}^{et} = \lambda f_{et}.\lambda g_{et}.\lambda x_{e}.g(x) \wedge f(x)$

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 $g(x) \wedge f(x)$. 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 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 $(et)((et)(et))$ 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:

(3.52) a. $(\text{IS}((\text{AND}^{et}(\textbf{thin}))(\textbf{tall})))(\textbf{tina})$
▷ compositional analysis of (3.50)

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

c. $= (((\lambda f_{et}.\lambda g_{et}.\lambda x_e.g(x) \wedge f(x))(\textbf{thin}))(\textbf{tall}))(\textbf{tina})$
▷ definition of AND^{et}

d. $= (((\lambda g_{et}.\lambda x_e.g(x) \wedge \textbf{thin}(x)))(\textbf{tall}))(\textbf{tina})$
▷ application to **thin**

e. $= (((\lambda x_e.\textbf{tall}(x) \wedge \textbf{thin}(x))))(\textbf{tina})$
▷ application to **tall**

f. $= \textbf{tall}(\textbf{tina}) \wedge \textbf{thin}(\textbf{tina})$ ▷ application to **tina**

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 χ_A and χ_B be *et* functions that characterize the sets of entities A and B, respectively. For every entity x, we have:

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

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 λ-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:

Tina thanked Mary and praised herself ⇔ 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 (∧) 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: AND^{t} for sentential conjunction and AND^{et} for predicate conjunction. Although the two functions are of different types, namely $t(tt)$ and $(et)((et)(et))$, they are logically related. In fact, the parallelism we have observed only reflects a small part of the semantic relations between *and* conjunctions of different categories in natural language. Similar relations exist between *disjunctive* coordinations with *or* in different categories, and, to a lesser extent, between negation in different categories. These relations are often analyzed as revealing *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 *Tina is tall*, where they appear in *predicative* positions, following the copula *is*. However, English also allows adjectives to precede nouns, as in the following sentences:

(3.55) Tina is a *tall* woman; the *tall* engineer visited us; I met five *tall* astronomers.

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

(3.56) a. Tina is a *Chinese* pianist $\Leftrightarrow$ Tina is *Chinese* and Tina is a pianist.

b. My doctor wears no *white* shirts ⇔ No shirts that my doctor wears are *white*.

c. 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) $\text{A}_{(et)(et)} = \text{IS} = \lambda g_{et}.g$

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

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

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 '$\mathbf{chinese}^{\mathsf{mod}}$'. Now, since there is a semantic connection between the two usages of the adjective *Chinese*, we define the $(et)(et)$ denotation $\mathbf{chinese}^{\mathsf{mod}}$ on the basis of the et function **chinese**. Specifically, we associate $\mathbf{chinese}^{\mathsf{mod}}$ 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 $\mathbf{chinese}^{\mathsf{mod}}$ as follows:

(3.61) $\mathbf{chinese}^{\mathsf{mod}}_{(et)(et)} = \lambda f_{et}.\lambda x_e.\mathbf{chinese}(x) \wedge f(x)$

The function $\mathbf{chinese}^{\mathsf{mod}}$ 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):

(3.62) $(\text{IS}(\text{A}(\mathbf{chinese}^{\mathsf{mod}}(\mathbf{pianist}))))(\mathbf{tina})$
▷ compositional analysis of (3.60)
$= (\mathbf{chinese}^{\mathsf{mod}}(\mathbf{pianist}))(\mathbf{tina})$
▷ applying IS and A (identity functions)
$= ((\lambda f_{et}.\lambda x_e.\mathbf{chinese}(x) \wedge f(x))(\mathbf{pianist}))(\mathbf{tina})$
▷ definition (3.61) of $\mathbf{chinese}^{\mathsf{mod}}$
$= (\lambda x_e.\mathbf{chinese}(x) \wedge \mathbf{pianist}(x))(\mathbf{tina})$
▷ application to **pianist**
$= \mathbf{chinese}(\mathbf{tina}) \wedge \mathbf{pianist}(\mathbf{tina})$
▷ 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:

$\textbf{pianist}_{et}$:	characterizes the singleton set {**tina**}
$\textbf{biologist}_{et}$:	characterizes the set {**tina**, **mary**}
$[\![\textit{skillful}]\!]_{(et)(et)}(\textbf{pianist})$:	characterizes the singleton set {**tina**}
$[\![\textit{skillful}]\!]_{(et)(et)}(\textbf{biologist})$:	characterizes the singleton set {**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) For any model M, the denotation of *skillful* in M is an $(et)(et)$ function $\mathbf{skillful}^{\mathsf{mod}}$ that satisfies the following: for every et function f in M, the set characterized by $\mathbf{skillful}^{\mathsf{mod}}(f)$ 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 $\mathbf{skillful}^{\mathsf{mod}}$ so that it satisfies (3.67) by relying on another, arbitrary, function of type $(et)(et)$. When we denote this function $\mathbf{skillful}^{\mathsf{arb}}$, the definition of $\mathbf{skillful}^{\mathsf{mod}}$ reads as follows:

(3.68) $\mathbf{skillful}^{\mathsf{mod}} = \lambda f_{et}.\lambda x_{e}.(\mathbf{skillful}^{\mathsf{arb}}(f))(x) \wedge f(x)$

In words: the function $\mathbf{skillful}^{\mathsf{mod}}$ sends every et function f to the characteristic function of the (arbitrary) set characterized by $\mathbf{skillful}^{\mathsf{arb}}(f)$, 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 $\mathbf{skillful}^{\mathsf{arb}}$ 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 $\mathbf{skillful}^{\mathsf{arb}}$ 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) $(\text{IS}(\text{A}(\textbf{skillful}^{\text{mod}}(\textbf{pianist}))))(\textbf{tina})$

▷ compositional analysis

$= (\textbf{skillful}^{\text{mod}}(\textbf{pianist}))(\textbf{tina})$

▷ applying IS and A

$= (\lambda f_{et}.\lambda x_e.(\textbf{skillful}^{\text{arb}}(f))(x) \wedge f(x))(\textbf{pianist})(\textbf{tina})$

▷ definition (3.68)

$= \textbf{skillful}^{\text{arb}}(\textbf{pianist})(\textbf{tina}) \wedge \textbf{pianist}(\textbf{tina})$

▷ application to **pianist** and **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 $\textbf{skillful}^{\text{arb}}(\textbf{pianist})$.

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.

The following entailments define an adjective **A** *as being intersective/subsective, where* **X** *is a proper name and* **N** *is a common noun:*

A is **intersective** – **X** *is a* **A N** $\Leftrightarrow$ **X** *is* **A** *and* **X** *is a* **N**
e.g. *Dan is a Dutch man*
$\Leftrightarrow$ *Dan is Dutch and Dan is a man*

A is **subsective** – **X** *is a* **A N** $\Rightarrow$ **X** *is* **N**
e.g. *Dan is a skillful pianist* $\Rightarrow$ *Dan is a pianist*

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:* $F(B) = A \cap B$.

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) a. Tina [smiled [charmingly]] $\Rightarrow$ Tina smiled.
b. Tina [ran [with John]] $\Rightarrow$ Tina ran.

(3.71) a. Tina [is [a [pianist [from Rome]]]]
$\Leftrightarrow$ Tina is a pianist and Tina is from Rome.
b. Tina [is [a [pianist [who [praised herself]]]]]
$\Leftrightarrow$ Tina is a pianist and Tina praised herself.

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' λ-terms, and they are also referred to as *combinators*.
2. Denotations like NOT, AND^t 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.

Denotation	Type	Restrictions	Category
tina	e	-	proper name
smile	et	-	intransitive verb
praise	$e(et)$	-	transitive verb
pianist	et	-	common noun
chinese	et	-	predicative adjective
chinese$^{\text{mod}}$	$(et)(et)$	intersective: $\lambda f_{et}.\lambda x_e.$ **chinese**$(x) \wedge f(x)$	modificational adjective
skillful$^{\text{mod}}$	$(et)(et)$	subsective: $\lambda f_{et}.\lambda x_e.$ (**skillful**$^{\text{arb}}(f))(x) \wedge f(x)$	modificational adjective
IS	$(et)(et)$	combinator: $\lambda g_{et}.g$	copula (auxiliary verb)
A	$(et)(et)$	combinator: $\lambda g_{et}.g$	indefinite article
HERSELF	$(e(et))(et)$	combinator: $\lambda R_{e(et)}.\lambda x_e.R(x)(x)$	reflexive pronoun
NOT	$(et)(et)$	logical: $\lambda g_{et}.\lambda x_e.\sim(g(x))$	predicate negation
AND^{t}	$t(tt)$	logical: $\lambda x_t.\lambda y_t.y \wedge x$	sentential conjunction
AND^{et}	$(et)((et)(et))$	logical: $\lambda f_{et}.\lambda g_{et}.\lambda x_e.g(x) \wedge f(x)$	predicate conjunction

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

Introductory: For introductions of the lambda calculus from a linguistic perspective see Dowty et al. (1981); Gamut (1982). The treatment we used for reflexive pronouns is based on the variable-free approach to anaphora, introduced in Jacobson (2014). For an overview of other treatments of reflexives and other anaphors see Büring (2005). On various problems of negation and relevant references see Horn and Kato (2003). On coordination see Haspelmath (2004); Zamparelli (2011). On adjectives see McNally and Kennedy (2008).

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

and Seldin (1986). On the Boolean approach to coordination and negation phenomena see Keenan and Faltz (1985); Winter (2001). For a survey on adjectives and modification see Lassiter (2015).

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. $\overline{\{u, w\}}$, 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 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_{tt}.
   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:

$e + e(et)$,	$(et)t + et$,
$tt + (ee)(tt)$,	$et + e(tt)$,
$(e(et))(et) + ee$,	$e(et) + (e(et))(et)$,
$(e(et))(et) + e$,	$(e((et)t))(tt) + e((et)t)$,
$(e((et)t))(tt) + e$,	$(ee)(et) + e$,
$e((et)(et)) + e(et)$,	$(et)(et) + ((et)(et))(e(e(et)))$.

5. a. In a model with $D_e = \{\mathsf{t}, \mathsf{j}, \mathsf{m}\}$, 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 *praise* be in this model?
   b. Consider sentences of the form *John* [[*read Mary*] *Moby Dick*]. We let the *ditransitive* verb *read* be of type $e(e(et))$. Assume that John read Mary *Moby Dick*, Tina read Mary *Lolita*, and nobody else read anything else to anybody else. In such a model, give the et denotation of the expression *read Mary Moby Dick* and the $e(et)$ denotation of the expression *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 <u>two</u> 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 6*a* and 6*b*:

      (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) [Mary_e [walked_{et} <u>quickly</u>$_X$]$_Y$]$_t$
   (ii) [Mary_e [walked_{et} [<u>in</u>$_X$ Utrecht_e]$_Z$]$_Y$]$_t$
   (iii) [[<u>the</u>$_X$ pianist_{et}]$_e$ [smiled_{et}]$_{et}$]$_t$
   (iv) [[the_X [<u>skillful</u>$_Y$ pianist_{et}]$_{et}$]$_e$ smiled_{et}]$_t$

(v) [[$\text{Walk}_{et}\underline{\text{ing}}_X$]$_e$ [$\text{is}_{(et)(et)}$ fun_{et}]$_{et}$]$_t$
(vi) [[the_X [man_{et} [$\underline{\text{who}}_Y$ walked_{et}]$_Z$]$_M$]$_e$ smiled_{et}]$_t$
(vii) [[$\underline{\text{if}}_X$[you_e smile_{et}]$_t$]$_Y$ [you_e win_{et}]$_t$]$_t$
(viii) [I_e [[$\text{love}_{e(et)}$ it_e]$_Y$ [$\underline{\text{when}}_X$ [you_e smile_{et}]$_t$]$_Z$]$_M$]$_t$

b. Now consider the following puzzle: [[$\underline{\text{no}}_X$ man_{et}]$_Y$ smiled_{et}]$_t$. Give two solutions for Y. For each solution give the corresponding solution for X.

c. Based on your answers to 7*a* and 7*b*, find at least one solution for X, Y and Z in the following puzzle:
[$\underline{\text{There}}_X$ [$\text{is}_{(et)(et)}$ [trouble_{et} [$\underline{\text{in}}_Y\text{Paradise}_e$]$_Z$]]]$_t$.
Can you find any more solutions?

8. a. Give English descriptions (cf. (3.34)) for the following λ-terms:

(i) $\lambda f_{(et)t}.\lambda y_t.f(\lambda z_e.y)$
(ii) $\lambda x_e.\lambda f_{et}.f(x)$
(iii) $\lambda f_{et}.\lambda g_{tt}.\lambda x_e.g(f(x))$
(iv) $\lambda f_{ee}.\lambda g_{(ee)t}.\lambda x_e.g(\lambda y_e.f(x))$

b. Give λ-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 λ-terms as much as possible using function application:

(i) $((\lambda x_e.\lambda f_{et}.f(x))(\mathbf{tina}_e))(\mathbf{smile}_{et})$
(ii) $((\lambda f_{et}.\lambda g_{tt}.\lambda x_e.g(f(x)))(\mathbf{smile}_{et}))(\lambda y_t.y)$
(iii) $(((\lambda g_{e(et)}.\lambda x_e.\lambda y_e.(g(y))(x))(\mathbf{praise}_{e(et)}))(\mathbf{tina}_e))(\mathbf{mary}_e)$

b. We analyze sentences like *Mary is Tina* by letting the word *is* denote an $e(et)$ function F, which is different from the identity function of type $(et)(et)$. Define F as a λ-term, basing your definition on the equality formula $x = y$, which has the truth-value 1 iff x and y are equal. Write the λ-term for the structure *Lewis Carroll* [*is* [*C. L. Dodgson*]], and then simplify it as much as possible.

c. We analyze the sentence [*Tina* [*praised* [*her mother*]]] using the following denotations:
HER$=\lambda f_{ee}.\lambda g_{e(et)}.\lambda x_e.(g(f(x)))(x)$ and **mother**$_{ee}$ = the function sending each entity x to x's mother.
Assuming these denotations, give the λ-term we derive for the sentence. Then simplify the λ-term you got as much as possible using function application.

10. a. Give English descriptions for the λ-terms:

(i) $\lambda y_n.(\text{DOUBLER})(y)$ (see (3.25))
(ii) $\lambda y_e.(\lambda x_e.x)(y)$

b. Write simpler descriptions for the functions you described in 10*a*.
c. Complete the following conclusion:
for any function $f_{\tau\sigma}$, the function _____ equals f.
In the λ-calculus, the simplification rule that this conclusion supports is known as *eta-reduction*.
d. Simplify the following λ-term as much as possible using function application and eta-reductions:
$((\lambda f_{e(et)}.\lambda g_{tt}.\lambda x_e.\lambda y_e.g(f(x)(y)))(\lambda u_e.\lambda z_e.\textbf{praise}(z)(u)))(\lambda w_t.w)$

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

12. a. Simplify the following λ-term as much as possible using function application and the definition of AND^t:
$(\lambda f_{e(et)}.\lambda x_e.\lambda y_e.(\text{AND}^t((f(x))(y)))((f(y))(x)))(\textbf{praise}_{e(et)})$.
Describe in words the $e(et)$ function that you got.
b. (i) Give the two binary structures for the sentence *Tina is not tall and thin*.
(ii) For each of these structures, give the semantic interpretation derived using the lexical denotations NOT$_{(et)(et)}$ and AND^{et}.
(iii) Simplify the resulting λ-terms as much as possible using function application and the definitions of NOT$_{(et)(et)}$ and AND^{et}.
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 AND^{et} and HERSELF.

(iii) Simplify the resulting λ-terms as much as possible using function application and the definitions of AND^{et} and HERSELF.

d. For every function f of type 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 χ_S in (3.2) we denote: ${\chi_S}^* = S = \{\text{a}, \text{c}\}$. Show that for all functions h_1, h_2 of type et, the following holds:
$(\text{AND}^{et}(h_2)(h_1))^* = {h_1}^* \cap {h_2}^*$.

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

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

13. a. Account for the entailment (3.70a) with the adverb *charmingly*: describe the restriction on the adverb's denotation by completing the following sentence:
the function **charmingly** of type ___ maps any set A characterized by ___ to ___.

b. Account for the same entailment by postulating a λ-term for **charmingly** in terms of an arbitrary function $\textbf{charmingly}^{\text{arb}}$. Simplify the λ-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 13*a*, but now for the entailment (3.70b) and the preposition *with*. Complete the following sentence:
the function **with** of type ___ maps any entity (e.g. for *John*), to a function mapping any characteristic function χ_A (e.g. for *ran*) to ___.
(ii) Repeat your analysis in *b*, but now for (3.70b). Postulate a λ-term for **with** in terms of an arbitrary function $\textbf{with}^{\text{arb}}$. Simplify the λ-terms you get for the sentences in (3.70b).

d. Account for the equivalence (3.71a) by defining the denotation **from** (of which type?) on the basis of an arbitrary function $\textbf{from}^{\text{arb}}$ of type $e(et)$. 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 λ-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. $D_{tt}=\{[0\mapsto 0, 1\mapsto 0], [0\mapsto 0, 1\mapsto 1], [0\mapsto 1, 1\mapsto 0], [0\mapsto 1, 1\mapsto 1]\}$.
   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_{te} is along similar lines.
   c. For $D_e=\{l, n\}$, there are four functions in D_{ee}: $[l\mapsto l, n\mapsto l]$, $[l\mapsto l, n\mapsto n]$, $[l\mapsto n, n\mapsto l]$ and $[l\mapsto n, n\mapsto n]$. If we substitute these four functions for u, v, w and m in the answer to *3b*, we get the sixteen functions in $D_{(ee)t}$.
4. et, t, –, –, –, et, –, tt, –, –, –, $e(e(et))$.
5. a. **praise** $= t\mapsto [t\mapsto 0\ \ j\mapsto 0\ \ m\mapsto 0]\quad j\mapsto [t\mapsto 1\ \ j\mapsto 0\ \ m\mapsto 1]$
$m\mapsto [t\mapsto 1\ j\mapsto 1\ m\mapsto 1]$

b. $[\![$*read Mary Moby Dick*$]\!] = [\mathsf{t}\mapsto 1\ \mathsf{j}\mapsto 0\ \mathsf{m}\mapsto 0\ \mathsf{md}\mapsto 0\ \mathsf{lo}\mapsto 0]$
$[\![$*read Mary*$]\!] = \mathsf{t}\mapsto[\mathsf{t}\mapsto 0\ \mathsf{j}\mapsto 0\ \mathsf{m}\mapsto 0\ \mathsf{md}\mapsto 0\ \mathsf{lo}\mapsto 0],$
$\mathsf{j}\mapsto[\mathsf{t}\mapsto 0\ \mathsf{j}\mapsto 0\ \mathsf{m}\mapsto 0\ \mathsf{md}\mapsto 0\ \mathsf{lo}\mapsto 0],$
$\mathsf{m}\mapsto[\mathsf{t}\mapsto 0\ \mathsf{j}\mapsto 0\ \mathsf{m}\mapsto 0\ \mathsf{md}\mapsto 0\ \mathsf{lo}\mapsto 0],$
$\mathsf{md}\mapsto[\mathsf{t}\mapsto 0\ \mathsf{j}\mapsto 1\ \mathsf{m}\mapsto 0\ \mathsf{md}\mapsto 0\ \mathsf{lo}\mapsto 0],$
$\mathsf{lo}\mapsto[\mathsf{t}\mapsto 1\ \mathsf{j}\mapsto 0\ \mathsf{m}\mapsto 0\ \mathsf{md}\mapsto 0\ \mathsf{lo}\mapsto 0]].$

6. a. $(tt)(t(tt))$; $e(ee)$; $t(et)$; $((et)t)(e((et)t))$.
b. t and $(t(tt))(tt)$; e and $(ee)e$; e and $(et)t$; e and $(e((et)t))((et)t)$.
c. the solution $X = yz$; the solutions $X = y$ and $X = (yz)z$.

7. a. (i) $X{=}(et)(et)$ (ii) $X{=}e((et)(et))$,
(iii) $X{=}(et)e$ (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 7*c*:
$X{=}et$, $Y{=}e((et)(((et)(et))e))$, $Z{=}(et)(((et)(et))e)$; $X{=}e$,
$Y{=}e((et)(((et)(et))(et)))$, $Z{=}(et)(((et)(et))(et))$.

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_e 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) $\lambda R_{e(et)}.\lambda x_e.\lambda y_e.(R(y))(x)$.

9. a. (i) **smile**(**tina**); (ii) λx_e.**smile**(x); (iii) (**praise**(**mary**))(**tina**)
b. $F = \lambda y_e.\lambda x_e.x = y$; $(F(\mathbf{cld}))(\mathbf{lc}) = (\mathbf{lc}{=}\mathbf{cld})$

c. $((\text{HER}(\textbf{mother}))(\textbf{praise}_{e(et)}))(\textbf{tina}_e) =$
$(\textbf{praise}(\textbf{mother}(\textbf{tina})))(\textbf{tina})$

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) DOUBLER
(ii) the function sending every entity y to y, a.k.a. the function sending every entity x to x
c. the function $\lambda x_\tau . f(x)$ equals f
d. **praise**.

11. $[\![\textit{was praised by}]\!] = \lambda x_e.\lambda y_e.\textbf{praise}(y)(x)$

12. a. $\lambda x_e.\lambda y_e.(\textbf{praise}(y))(x) \wedge (\textbf{praise}(x))(y)$ – 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) $(\text{IS}(\text{NOT}((\text{AND}^{et}(\textbf{thin}))(\textbf{tall}))))(\textbf{tina})$;
$(\text{IS}((\text{AND}^{et}(\textbf{thin}))(\text{NOT}(\textbf{tall}))))(\textbf{tina})$
(iii) $\sim(\textbf{tall}(\textbf{tina}) \wedge \textbf{thin}(\textbf{tina}))$; $(\sim(\textbf{tall}(\textbf{tina}))) \wedge \textbf{thin}(\textbf{tina})$.
d. Suppose $h_1{}^* = A_1$, $h_2{}^* = A_2$. By def. of AND^{et}: $(\text{AND}^{et}(h_2)(h_1))^* = (\lambda x_e.h_1(x) \wedge h_2(x))^*$, i.e. the set $A = \{x \in E : (h_1(x) \wedge h_2(x)) = 1\}$. By def. of $\wedge$, 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. $y \in A_1 \cap A_2$.

13. c. **with** of type $e((et)(et))$ maps any entity x (e.g. for *John*), to a function mapping any characteristic function χ_A (e.g. for *ran*) to a function characterizing subsets of A;
$\textbf{with} = \lambda x_e.\lambda f_{et}.\lambda y_e.((\textbf{with}^{\text{arb}}_{e((et)(et))}(x))(f))(y) \wedge f(y)$.
d. $\textbf{from}_{e((et)(et))} = \lambda x_e.\lambda f_{et}.\lambda y_e.(\textbf{from}^{\text{arb}}_{e(et)}(x))(y) \wedge f(y)$.
f. WHO is assigned type $(et)(et)$, which leads to the term: $(\text{IS}(\text{A}(\text{WHO}(\text{HERSELF}(\textbf{praise}))(\textbf{pianist}))))(\textbf{tina})$. With the assumption $\text{WHO} = \text{AND}^{et}$, this term is simplified to: $\textbf{pianist}(\textbf{tina}) \wedge \textbf{praise}(\textbf{tina})(\textbf{tina})$. We get the same term when simplifying the term for the sentence *Tina is a pianist and Tina praised herself.*

BIBLIOGRAPHY

Abbott, B. (1999), 'The formal approach to meaning: Formal semantics and its recent developments', *Journal of Foreign Languages* **119**, 2–20. `https://www.msu.edu/~abbottb/formal.htm`. Accessed: 2015-3-24.

Abbott, B. (2011), 'Support for individual concepts', *Linguistic and Philosophical Investigations* **10**, 23–44.

ACG (2015), 'The Abstract Categorial Grammar Homepage', `http://www.loria.fr/equipes/calligramme/acg/`. Accessed: 2015-3-4.

Adler, J. E. and Rips, L. J., eds (2008), *Reasoning: studies of human inference and its foundation*, Cambridge University Press, New York.

Ajdukiewicz, K. (1935), 'Die syntaktische konnexität', *Studia Philosophia* **1**, 1–27.

Austin, J. L. (1962), *How to do Things with Words: The William James Lectures delivered at Harvard University in 1955*, Clarendon, Oxford. Edited by J. O. Urmson.

Bach, E., Jelinek, E., Kratzer, A. and Partee, B. B. H., eds (1995), *Quantification in Natural Languages*, Kluwer Academic Publishers, Dordrecht.

Barendregt, H., Dekkers, W. and Statman, R. (2013), *Lambda Calculus with Types*, Cambridge University Press, Cambridge.

Barker, C. and Jacobson, P. (2007), Introduction: Direct compositionality, *in* C. Barker and P. Jacobson, eds, 'Direct Compositionality', Oxford University Press, Oxford.

Barker-Plummer, D., Barwise, J. and Etchemendy, J. (2011), *Language, Proof, and Logic*, 2 edn, CSLI Publications, Stanford.

Barwise, J. and Cooper, R. (1981), 'Generalized quantifiers and natural language', *Linguistics and Philosophy* **4**, 159–219.

Beaver, D. I. and Geurts, B. (2014), Presupposition, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', winter 2014 edn.

Bos, J. (2011), 'A survey of computational semantics: Representation, inference and knowledge in wide-coverage text understanding', *Language and Linguistics Compass* **5**(6), 336–366.

Boye, K. (2012), *Epistemic meaning: a crosslinguistic and functional-cognitive study*, De Gruyter, Berlin.

Brewka, G., J. D. and Konolige, K. (1997), *Nonmonotonic Reasoning: An Overview*, CSLI Publications, Stanford.

Büring, D. (2005), *Binding Theory*, Cambridge University Press, Cambridge.

Carlson, G. (2011), Genericity, *in* Von Heusinger et al. (2011), pp. 1153–1185.

Carnie, A. (2013), *Syntax: A generative introduction*, 3 edn, Wiley-Blackwell.

Carpenter, B. (1997), *Type-Logical Semantics*, MIT Press, Cambridge, Massachusetts.

Chemla, E. and Singh, R. (2014), 'Remarks on the experimental turn in the study of scalar implicature, part I', *Language and Linguistics Compass* **8**(9), 373–386.

Chierchia, G., Fox, D. and Spector, B. (2012), Scalar implicature as a grammatical phenomenon, *in* Maienborn et al. (2012), pp. 2297–2332.

Chierchia, G. and McConnel-Ginet, S. (1990), *Meaning and Grammar: an introduction to semantics*, MIT Press, Cambridge, Massachusetts.

Crain, S. (2012), *The emergence of meaning*, Cambridge University Press, Cambridge.

Cruse, D. A. (1986), *Lexical Semantics*, Cambridge University Press, Cambridge.

Curry, H. B. (1961), Some logical aspects of grammatical structure, *in* R. O. Jakobson, ed., 'Structure of Language and its Mathematical Aspects', Vol. 12 of *Symposia on Applied Mathematics*, American Mathematical Society, Providence.

Dagan, I., Roth, D., Sammons, M. and Zanzotto, F. M. (2013), *Recognizing textual entailment: Models and applications*, Morgan & Claypool.

De Groote, P. (2001), Towards abstract categorial grammars, *in* 'Proceedings of the 39th annual meeting of the Association for Computational Linguistics (ACL)'.

De Groote, P. and Kanazawa, M. (2013), 'A note on intensionalization', *Journal of Logic, Language and Information* **22**, 173–194.

De Saussure, F. (1959), *Course in General Linguistics*, Philosophical Library, New York. Translation of *Cours de Linguistique Générale*, Payot & Cie, Paris, 1916.

Dehaene, S. (2014), *Consciousness and the brain: Deciphering how the brain codes our thoughts*, Viking Penguin, New York.

Dowty, D., Wall, R. and Peters, S. (1981), *Introduction to Montague Semantics*, D. Reidel, Dordrecht.

Elbourne, P. (2011), *Meaning: a slim guide to semantics*, Oxford University Press, Oxford.
Fitting, M. and Mendelsohn, R. L. (1998), *First-Order Modal Logic*, Kluwer Academic Publishers, Dordrecht.
Fodor, J. D. (1970), The linguistic description of opaque contexts, PhD thesis, Massachusetts Institute of Technology.
Frege, G. (1892), 'Über sinn und bedeutung', *Zeitschrift für Philosophie und philosophische Kritic* **100**, 25–50. Translated in as 'On sense and reference' in Geach and Black (1960, pp.56-78).
Fromkin, V., Rodman, R. and Hyams, N. (2014), *An introduction to language*, 10 edn, Wadsworth, Cengage Learning.
Gamut, L. T. F. (1982), *Logica, Taal en Betekenis*, Het Spectrum, De Meern. In two volumes. Appeared in English as *Logic, Language and Meaning*, The University of Chicago Press, 1991.
Geach, P. and Black, M., eds (1960), *Translations from the Philosophical Writings of Gottlob Frege*, Basil Blackwell, Oxford. Second edition.
Geurts, B. (2010), *Quantity implicatures*, Cambridge University Press, Cambridge.
Goranko, V. and Galton, A. (2015), Temporal logic, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', summer 2015 edn.
Grice, H. P. (1975), Logic and conversation, *in* P. Cole and J. L. Morgan, eds, 'Syntax and Semantics, Vol. 3, Speech Acts', Academic Press, New York, pp. 41–58.
Groenendijk, J. and Stokhof, M. (1984), Studies on the Semantics of Questions and the Pragmatics of Answers, PhD thesis, University of Amsterdam.
Groenendijk, J. and Stokhof, M. (2011), Questions, *in* Van Benthem and ter Meulen (2011), pp. 1059–1132.
Gunter, C. (1992), *Semantics of programming languages: structures and techniques*, MIT Press, Cambridge, Massachusetts.
Halmos, P. R. (1960), *Naive set theory*, Springer Science & Business Media.
Hamm, F. and Bott, O. (2014), Tense and aspect, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', spring 2014 edn.
Haspelmath, M. (2004), Coordinating constructions: an overview, *in* M. Haspelmath, ed., 'Coordinating Constructions', John Benjamins Publishing Company, Amsterdam/Philadelphia.
Heim, I. (2011), Definiteness and indefiniteness, *in* Von Heusinger et al. (2011), pp. 996–1024.
Heim, I. and Kratzer, A. (1997), *Semantics in Generative Grammar*, Blackwell.
Hendricks, V. and Symons, J. (2014), Epistemic logic, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', spring 2014 edn.
Hendriks, H. (1993), Studied Flexibility: categories and types in syntax and semantics, PhD thesis, University of Amsterdam.
Herskovits, A. (1986), *Language and Spatial Cognition: an interdisciplinary study of the prepositions in English*, Cambridge University Press, Cambridge.
Hindley, J. R. and Seldin, J. P. (1986), *Introduction to Combinators and the Lambda-Calculus*, Cambridge University Press, Cambridge.
Horn, L. R. and Kato, Y. (2003), Introduction: Negation and polarity at the millenium, *in* L. R. Horn and Y. Kato, eds, 'Negation and polarity : syntactic and semantic perspectives', Oxford University Press, Oxford, pp. 1–20.
Hutton, G. (2007), *Programming in Haskell*, Cambridge University Press, Cambridge.
Jacobson, P. (1999), 'Towards a variable-free semantics', *Linguistics and Philosophy* **22**, 117–185.
Jacobson, P. (2014), *Compositional Semantics: An Introduction to the Syntax/Semantics Interface*, Oxford University Press, Oxford.
Janssen, T. M. V. (1983), Foundations and Applications of Montague Grammar, PhD thesis, Mathematisch Centrum, Amsterdam.
Janssen, T. M. V. (2011), Compositionality, *in* Van Benthem and ter Meulen (2011), pp. 495–554. with B. H. Partee.
Joseph, J. E. (2012), *Saussure*, Oxford University Press, Oxford.
Kamareddine, F., Laan, T. and Nederpelt, R. (2004), *A Modern Perspective on Type Theory*, Kluwer Academic Publishers, Dordrecht.
Kamp, H. and Reyle, U. (1993), *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*, Kluwer, Dordrecht.
Keenan, E. L. (1989), Semantic case theory, *in* R. Bartsch, J. van Benthem and P. van Emde Boas, eds, 'Semantics and Contextual Expression', Foris, Dordrecht.
Keenan, E. L. (1996), The semantics of determiners, *in* Lappin (1996), pp. 41–64.
Keenan, E. L. (2003), 'The definiteness effect: semantics or pragmatics?', *Natural Language Semantics* **11**(2), 187–216.
Keenan, E. L. (2006), Quantifiers: Semantics, *in* E. K. Brown, ed., 'Encyclopedia of language & linguistics', 2

edn, Vol. 10, Elsevier, Amsterdam, pp. 302–308.
Keenan, E. L. (2007), 'On the denotations of anaphors', *Research on Language and Computation* **5**, 5–17.
Keenan, E. L. (2011), Quantifiers, *in* Von Heusinger et al. (2011), pp. 1058–1087.
Keenan, E. L. and Faltz, L. (1978), *Logical Types for Natural Language*, UCLA Occasional Papers in Linguistics 3, Department of Linguistics UCLA.
Keenan, E. L. and Faltz, L. (1985), *Boolean Semantics for Natural Language*, D. Reidel, Dordrecht.
Keenan, E. L. and Paperno, D., eds (2012), *Handbook of Quantifiers in Natural Language*, Springer, Dordrecht.
Keenan, E. L. and Westerståhl, D. (2011), Generalized quantifiers in linguistics and logic, *in* Van Benthem and ter Meulen (2011), pp. 859–910.
Kennedy, C. (2007), 'Vagueness and grammar: the semantics of relative and absolute gradable adjectives', *Linguistics and Philosophy* **30**, 1–45.
Kennedy, C. (2011), Ambiguity and vagueness: An overview, *in* Maienborn et al. (2011), pp. 507–535.
Klein, E. (1980), 'A semantics for positive and comparative adjectives', *Linguistics and Philosophy* **4**, 1–45.
Koons, R. (2014), Defeasible reasoning, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', spring 2014 edn.
Kracht, M. (2003), *The Mathematics of Language*, De Gruyter, Berlin.
Krifka, M., Pelletier, F. J., Carlson, G. N., ter Meulen, A., Chierchia, G. and Link, G. (1995), Genericity: an introduction, *in* G. N. Carlson and F. J. Pelletier, eds, 'The Generic Book', University of Chicago Press, Chicago.
Lambek, J. (1958), 'The mathematics of sentence structure', *American Mathematical Monthly* **65**, 154–169.
Lappin, S., ed. (1996), *The Handbook of Contemporary Semantic Theory*, Blackwell.
Lappin, S. and Fox, C., eds (2015), *Handbook of Contemporary Semantic Theory*, 2 edn, Wiley-Blackwell. In print.
Lasersohn, P. (1995), *Plurality, Conjunction and Events*, Kluwer, Dordrecht.
Lasersohn, P. (2011), Mass nouns and plurals, *in* Von Heusinger et al. (2011), pp. 1131–1153.
Lassiter, D. (2015), Adjectival modification and gradation, *in* Lappin and Fox (2015). In print.
Laurence, S. and Margolis, E. (1999), Introduction, *in* E. Margolis and S. Laurence, eds, 'Concepts: Core Readings', MIT Press, Cambridge, Massachusetts.
Levin, B. (1993), *English verb classes and alternations: A preliminary investigation*, University of Chicago Press, Chicago.
Levinson, S. C. (1983), *Pragmatics*, Cambridge University Press, Cambridge.
Liang, P. and Potts, C. (2015), 'Bringing machine learning and compositional semantics together', *Annual Review of Linguistics* **1**(1), 355–376.
Maienborn, C. (2011), Event semantics, *in* Maienborn et al. (2011), pp. 802–829.
Maienborn, C., Heusinger, K. V. and Portner, P., eds (2011), *Semantics: An International Handbook of Natural Language Meaning*, Vol. 1, De Gruyter, Berlin.
Maienborn, C., Heusinger, K. V. and Portner, P., eds (2012), *Semantics: An International Handbook of Natural Language Meaning*, Vol. 3, De Gruyter, Berlin.
Matthewson, L., ed. (2008), *Quantification: a cross-linguistic perspective*, Emerald Group Publishing Limited, Bingley, UK.
McAllester, D. A. and Givan, R. (1992), 'Natural language syntax and first-order inference', *Artificial Intelligence* **56**, 1–20.
McGinn, C. (2015), *Philosophy of Language: The Classics Explained*, MIT Press, Cambridge, Massachusetts.
McNally, L. (2011), Existential sentences, *in* Von Heusinger et al. (2011), pp. 1829–1848.
McNally, L. and Kennedy, C. (2008), Introduction, *in* L. McNally and C. Kennedy, eds, 'Adjectives and Adverbs: Syntax, semantics, and discourse', Oxford University Press, Oxford, pp. 1–15.
Menzel, C. (2014), Possible worlds, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', fall 2014 edn.
Mikkelsen, L. (2011), Copular clauses, *in* Von Heusinger et al. (2011), pp. 1805–1829.
Montague, R. (1970*a*), English as a formal language, *in* B. Visentini, ed., 'Linguaggi nella Società e nella Technica', Edizioni di Communità, Milan. Reprinted in Thomason (1974).
Montague, R. (1970*b*), 'Universal grammar', *Theoria* **36**, 373–398. Reprinted in Thomason (1974).
Montague, R. (1973), The proper treatment of quantification in ordinary English, *in* J. Hintikka, J. Moravcsik and P. Suppes, eds, 'Approaches to Natural Languages: proceedings of the 1970 Stanford workshop on grammar and semantics', D. Reidel, Dordrecht. Reprinted in Thomason (1974).
Moortgat, M. (2011), Categorial type logics, *in* Van Benthem and ter Meulen (2011), pp. 95–190.
Moot, R. and Retoré, C. (2012), *The Logic of Categorial Grammars: a Deductive Account of Natural Language Syntax and Semantics*, Springer, Berlin.
Morrill, G. (1994), *Type Logical Grammar: Categorial Logic of Signs*, Dordrecht, Kluwer.

Moss, L. S. (2010), Natural logic and semantics, *in* M. Aloni, H. Bastiaanse, T. de Jager and K. Schulz, eds, 'Proceedings of the Seventeenth Amsterdam Colloquium', Vol. 6042 of *Lecture Notes in Computer Science*, Springer, pp. 84–93.

Murphy, M. L. (2010), *Lexical meaning*, Cambridge University Press, Cambridge.

Muskens, R. (2003), Language, Lambdas, and Logic, *in* G.-J. Kruijff and R. Oehrle, eds, 'Resource Sensitivity in Binding and Anaphora', Studies in Linguistics and Philosophy, Kluwer, Dordrecht, pp. 23–54.

Oehrle, R. (1994), 'Term-labeled categorial type systems', *Linguistics and Philosophy* **17**, 633–678.

Pagin, P. and Westerståhl, D. (2010), 'Compositionality I: Definitions and variants', *Philosophy Compass* **5**, 265–82.

Partee, B. H. (1984), Compositionality, *in* F. Landman and F. Veltman, eds, 'Varaities of Formal Semantics', Foris, Dordrecht. reprinted in Partee (2004).

Partee, B. H. (1996), The development of formal semantics in linguistic theory, *in* Lappin (1996), pp. 11–38.

Partee, B. H. (2015), *History of Formal Semantics*. Materials for a book in preparation, to appear in Oxford University Press. `http://people.umass.edu/partee/Research.htm`. Accessed: 2015-3-24.

Partee, B. H., ed. (2004), *Compositionality in formal semantics: Selected papers by Barbara H. Partee*, Blackwell, Malden.

Partee, B. H., ter Meulen, A. and Wall, R. (1990), *Mathematical Methods in Linguistics*, Kluwer, Dordrecht.

Pelletier, F. J. (2000), A history of natural deduction and elementary logic textbooks, *in* J. Woods and B. Brown, eds, 'Logical Consequence: Rival approaches', Vol. 1, Hermes Science Pubs, pp. 105–138.

Penka, D. and Zeijlstra, H. (2010), 'Negation and polarity: an introduction', *Natural Language and Linguistic Theory* **28**, 771–786.

Peters, S. and Westerståhl, D. (2006), *Quantifiers in Language and Logic*, Oxford University Press, Oxford.

Portner, P. (2009), *Modality*, Oxford University Press, Oxford.

Potts, C. (2015), Presupposition and implicature, *in* Lappin and Fox (2015). In print.

Prawitz, D. (1965), *Natural Deduction: A Proof-Theoretical Study*, Almqvist & Wiksell, Stockholm.

Pylkkänen, L. (2008), 'Mismatching meanings in brain and behavior', *Language and Linguistics Compass* **2**, 712–738.

Quine, W. V. O. (1956), 'Quantifiers and propositional attitudes', *The Journal of Philosophy* **53**, 177–187.

Saeed, J. I. (1997), *Semantics*, Blackwell, Oxford.

Sánchez, V. (1991), Studies on Natural Logic and Categorial Grammar, PhD thesis, University of Amsterdam.

Schönfinkel, M. (1924), 'Über die bausteine der mathematischen logik', *Mathematische Annalen* **92**(3), 305–316.

Schroeder-Heister, P. (2014), Proof-theoretic semantics, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', summer 2014 edn.

Searle, J. R. (1969), *Speech Acts*, Cambridge University Press, Cambridge.

SIGSEM (2015), 'Web site of SIGSEM, the Special Interest Group on Computational Semantics, Association for Computational Linguistics (ACL)', `http://www.sigsem.org`. Accessed: 2015-6-26.

Smith, E. E. (1988), Concepts and thought, *in* R. J. Sternberg and E. E. Smith, eds, 'The Psychology of Human Thought', Cambridge University Press, Cambridge.

ST (2015), 'Set Theory – Wikibooks', `http://en.wikibooks.org/wiki/Set_Theory`. Accessed: 2015-3-25.

Steedman, M. (1997), *Surface Structure and Interpretation*, MIT Press, Cambridge, Massachusetts.

Stenning, K. and van Lambalgen, M. (2007), *Human Reasoning and Cognitive Science*, MIT Press, Cambridge, Massachusetts.

Suppes, P. (1957), *Introduction to Logic*, Van Nostrand Reinhold Company, New York.

Szabolcsi, A. (1987), Bound variables in syntax: are there any?, *in* 'Proceedings of the 6th Amsterdam Colloquium'.

Szabolcsi, A. (2010), *Quantification*, Cambridge University Press, Cambridge.

Taylor, J. R. (1989), *Linguistic Categorization: prototypes in linguistic theory*, Oxford University Press, Oxford.

Thomason, R., ed. (1974), *Formal Philosophy: selected papers of Richard Montague*, Yale, New Haven.

Thompson, S. (1991), *Type Theory and Functional Programming*, Addison-Wesley, Wokingham.

Van Benthem, J. (1986), *Essays in Logical Semantics*, D. Reidel, Dordrecht.

Van Benthem, J. (1991), *Language in Action: Categories, Lambdas and Dynamic Logic*, North-Holland, Amsterdam.

Van Benthem, J. and ter Meulen, A., eds (2011), *Handbook of Logic and Language*, 2 edn, Elsevier, Amsterdam.

Van Eijck, J. and Unger, C. (2010), *Computational Semantics with Functional Programming*, Cambridge University Press, Cambridge.

Von Fintel, K. (2004), Would you believe it? The king of France is back! (presuppositions and truth-value intuitions), *in* M. Reimer and A. Bezuidenhout, eds, 'Descriptions and Beyond', Oxford University Press,

Oxford, pp. 315–341.

Von Fintel, K. (2006), Modality and language, *in* D. M. Borchert, ed., 'Encyclopedia of Philosophy – Second Edition', MacMillan Reference USA, Detroit, pp. 315–341.

Von Fintel, K. and Heim, I. (2011), Intensional semantics. Unpublished lecture notes, Massachusetts Institute of Technology, `http://web.mit.edu/fintel/fintel-heim-intensional.pdf`, Accessed: 2015-3-10.

Von Heusinger, K., Maienborn, C. and Portner, P., eds (2011), *Semantics: An International Handbook of Natural Language Meaning*, Vol. 2, De Gruyter, Berlin.

Werning, M., Hinzen, W. and Machery, E. (2012), *The Oxford handbook of compositionality*, Oxford University Press, Oxford.

Westerståhl, D. (2015), Generalized quantifiers in natural language, *in* Lappin and Fox (2015). In print.

Winter, Y. (2001), *Flexibility Principles in Boolean Semantics: coordination, plurality and scope in natural language*, MIT Press, Cambridge, Massachusetts.

Winter, Y. and Scha, R. (2015), Plurals, *in* Lappin and Fox (2015). In print.

XPRAG (2015), 'Homepage of the Experimental Pragmatics conference', `https://lucian.uchicago.edu/blogs/xprag2015/`. Accessed: 2015-6-26.

Zamparelli, R. (2011), Coordination, *in* Von Heusinger et al. (2011), pp. 1713–1741.

Zimmermann, T. E. and Sternefeld, W. (2013), *Introduction to semantics: an essential guide to the composition of meaning*, De Gruyter, Berlin.