Formal Semantics

Derek Bridge Steve Harlow

September 10, 2003

Contents

Figures

Tables

Grammars

Chapter 1

An Introduction to Semantics

We will begin by stating some basic assumptions we make about the treatment of semantics; this is followed by two sections that illustrate the so-called 'model-theoretic' approach that we are taking; then, as a precursor to a section that uses logic to mediate in the assignment of semantic values to English expressions, we briefly review the syntax and semantics of a fragment of logic and extend the fragment with so-called 'lambda-expressions'. The chapter ends with a discussion of semantic translation.

1.1 Model-theoretic semantics

Semantic knowledge accounts for an aspect of the meaning of utterances. Specifically, it focuses on their 'core', 'literal', context-independent linguistic meaning (and, hence, on the meaning of sentences, rather than that of utterances). The satisfactory treatment of the meaning of natural language utterances is a highly complex issue with many unresolved problems. In this section we explain the framework that forms the basis of most contemporary linguistic theories of meaning. These theories agree that the basis of any approach to the meanings of natural language sentences should be by means of an analysis of their **truth conditions**. Under this approach, the minimal 'meaning' of a declarative sentence is a truth value. That is, in order to know the meaning of a (declarative) sentence (i.e. a sentence that is typically used to make a statement, as opposed to interrogative sentences, that are typically used to pose questions, or imperative sentences, that are typically used to issue commands or requests), you need to know (at least) how the world would have to be for the sentence to be true. (Note the word "minimal" in the above; it is also recognised that a full treatment of meaning will cover much more than truth conditions, but it seems clear that it could hardly cover less.)

Additionally, these theories, and all truth conditional approaches to natural language semantics, assume that meaning is assigned **compositionally**, that is to say, that the meaning of a linguistic expression is a function of the meanings of its parts. Specifically, what this means is that expressions of, say, English are assigned set-theoretic semantic values (or 'denotations') of various sorts, which are combined constituent by constituent to give eventually the truth value associated with the sentence composed out of them. (This is why we said in previous chapters that semantic processing can be 'driven' by the phrase-structures assigned to strings by syntactic

processing.)[1]

In order to be able to evaluate the truth or falsity of a sentence, we need to construct a **model** of the world, or a state of affairs, with respect to which the truth of the sentence is to be evaluated. (Hence the name **model-theoretic semantics** for this approach.) A basic model consists simply of a set containing the entities in the world (this set is called the **universe of discourse** of the model and we will use the label $\mathcal{U}$ for it), and a function (the **interpretation function**,[2] for which we will use the symbol $\mathcal{I}$) that assigns to expressions of the language under investigation objects constructed out of $\mathcal{U}$.

Perhaps the best way to see how these ideas work is by way of an example. What follows is a considerable simplification over what is required for a full treatment of natural language semantics, but it gives the flavour. We start by introducing some of the basic concepts and terminology that are required.

1.2 Denotations

'Denotation' is the name given to the kind of set-theoretic object taken to be the interpretation of a linguistic expression. We first present the denotations of simple noun phrases, verb phrases that comprise just an intransitive verb, and sentences made up of these; then we present the denotations of verb phrases that contain transitive verbs and their direct object noun phrases.

1.2.1 Simple English expressions

Intuition suggests the following concerning the denotations of some simple English expressions:[3]

- The denotations of NPs such as *"Duncan"* and *"Macbeth"* are taken to be *individuals* in the universe of discourse of the model of the world in which the sentences are being interpreted.[4]
- Intransitive verbs, Vi, such as *"died"* or *"slept"* denote *sets of individuals* in $\mathcal{U}$. That is, *"died"* has as its denotation the set of individuals in the model that

[1] Compositionality is controversial. Definitions vary; some linguists believe that (at least some of) the definitions place constraints that are too restrictive to allow treatment of certain natural language semantic phenomena; others believe that (at least some of) the definition are nearly vacuous, this perhaps being supported by the number of times that non-compositional theories that have been proposed to account for certain phenomena have then been re-formulated in compositional terms.

[2] This use of the word 'interpretation' has no significant relationship with our use of the word 'interpreter' in the previous chapter.

[3] Note that we are adopting the practice here, as elsewhere in this book, of enclosing in double quotes and displaying in italics words of the language that we are describing; the expression *"died"*, for example, should be read as 'the English word *"died"*'. These conventions become very important in this chapter, where expressions from other languages, notably logic, and representations of non-linguistic objects all appear in the text.

[4] This is an over-simplification. In fact, we need to assign more complex denotations to NPs in general to handle NPs such as *"every witch"*, *"few soldiers"*, etc., each of whose denotations could hardly simply be an individual like the denotation of *"Duncan"* and *"Macbeth"*. We do this in chapter 2.

died.[5]

- Sentences, S, denote *truth values*, the conventional notation for which is {**0**, **1**} (**0** being 'false' and **1** being 'true').

One problem of presentation that we face immediately is how we distinguish between linguistic items, such as the word *"Duncan"*, and the denotation of this expression with respect to a model. Such denotations are real-world entities, but it is clearly not possible to display these real-world entities in a book. One option might be to use some non-linguistic representation (such as pictures), but this too presents considerable problems. The strategy that we will adopt here is to use the symbols $\mathbf{i_1}$, $\mathbf{i_2}$, $\mathbf{i_3}$,..., $\mathbf{i_n}$ to denote entities in the model that are intended to correspond to real-world individuals. Since expressions like these are not a part of normal English, there will be little chance of confusing them with words of English, but it is important to try and bear in mind that these symbols are intended to stand for real-world individuals.

We will construct a very simple model of a world which only contains two individuals who we will identify as $\mathbf{i_1}$ and $\mathbf{i_2}$. The model then consists of the set

$$\mathcal{U} = \{\mathbf{i_1}, \mathbf{i_2}\}$$

as its universe of discourse.

We define the relationship between the words *"Duncan"*, *"Macbeth"* and *"died"* through the interpretation function, $\mathcal{I}$, as follows:

$$\begin{array}{lcl} \mathcal{I}(\text{``Duncan''}) & = & \mathbf{i_1} \\ \mathcal{I}(\text{``Macbeth''}) & = & \mathbf{i_2} \\ \mathcal{I}(\text{``died''}) & = & \{\mathbf{i_1}\} \end{array}$$

This states that, in this model, the semantic value of the word *"Duncan"* is the individual $\mathbf{i_1}$, the semantic value of the word *"Macbeth"* is the individual $\mathbf{i_2}$, and the semantic value of the word *"died"* is the set consisting of the single individual $\mathbf{i_1}$. (In other words, Duncan is the only dead individual.)[6]

We use the notation $[\![\alpha]\!]^M$ to represent the **semantic value** or **denotation** of some syntactic expression α with respect to some model M. So far, we have specified the semantic values of three English words: *"Duncan"*, *"Macbeth"* and *"died"*. Using this notation we can state the following:

$$\begin{array}{lcl} [\![\text{``Duncan''}]\!]^M & = & \mathbf{i_1} \\ [\![\text{``Macbeth''}]\!]^M & = & \mathbf{i_2} \\ [\![\text{``died''}]\!]^M & = & \{\mathbf{i_1}\} \end{array}$$

Having given a semantic value to items from the lexicon, we now need to state how the semantic values of more complex expressions are constructed. The essence of this process is to pair with each syntactic rule of the grammar, a semantic rule stating how the construction it describes is assigned a semantic value in the model. We will use the trivial (syntactic) grammar and lexicon given as grammar 1.1. (This

[5] We will be ignoring tense and aspect of verb phrases in our simplified treatment.

[6] As ever, we do not claim to have attained Shakespearean fidelity.

$\mathsf{S} \rightarrow \mathsf{NP\ VP}$	*Duncan*: NP
$\mathsf{VP} \rightarrow \mathsf{Vi}$	*Macbeth*: NP
	died: Vi
$\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$	*killed*: Vt

Grammar 1.1: The grammar whose semantics will be developed

grammar is different from the ones we have used so far in this book. We have had to simplify the syntactic constructions to give an accessible treatment of semantics in this chapter. We have also split verbs into intransitive verbs, Vi, and transitive verbs, Vt, as their denotations are different. We have shown the lexical entry of the transitive verb and the transitive VP rule slightly apart from the rest of the grammar to reflect the fact that their denotations are explained in the next subsection; the denotations of the rest of the grammar are explained in this subsection.)

We know, by virtue of the specification given by $\mathcal{I}$ above, what the semantic values of the lexical items in the grammar are. We now need to specify how to arrive at the semantic values of the other categories in the grammar. We will deal first of all with the lexicon:

- If W is a lexical item, then $[\![W]\!]^M$ is the value assigned to W by $\mathcal{I}$
- For any lexical entry
$$W : C,$$
where W is a lexical item, then $[\![C]\!]^M = [\![W]\!]^M$

So, if we have a sentence which contains the NP *"Duncan"*, that NP will have the same semantic value as the word *"Duncan"*.

For each of the other rules of the grammar, we specify its semantic value by pairing with the syntactic rule the instructions for computing its semantic value. The term generally used for this strategy is the **Rule to Rule Hypothesis**, a term attributed to the linguist Emmon Bach (Bach 1976). This is an obvious way of giving a compositional account, an account in which a finite set of semantic rules can assign meanings to an infinite set of well-formed expressions (the sentences and phrases of the language).

In the case of VPs containing only intransitive verbs, we specify that the VP and the verb it introduces have the same semantic value:

English syntax	Semantics
$\mathsf{VP} \rightarrow \mathsf{Vi}$;	$[\![\mathsf{VP}]\!]^M = [\![\mathsf{Vi}]\!]^M$

For S we specify that the sentence is true ($= \mathbf{1}$) if and only if the semantic value of the NP is a member ($\in$) of the set which is the semantic value of the VP, otherwise it is false ($= \mathbf{0}$):

English syntax	Semantics
$\mathsf{S} \rightarrow \mathsf{NP\ VP}$;	$[\![\mathsf{S}]\!]^M = \mathbf{1}$ if and only if $[\![\mathsf{NP}]\!]^M \in [\![\mathsf{VP}]\!]^M$

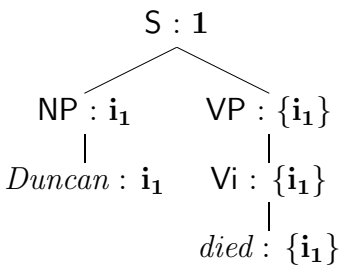

Figure 1.1: Interpreted tree for *"Duncan died"*

We can illustrate the constituent-by-constituent construction of the semantic value of the sentence *"Duncan died"* in accordance with these rules by means of an **interpreted tree**, in which we embellish each node with its semantic value. The interpreted tree is shown as figure 1.1. In the figure, as specified by the semantic rules, the semantic values of the nodes labelled *"Duncan"* and *"died"* are the values assigned to these words by the interpretation function (the individual $\mathbf{i_1}$ and the set $\{\mathbf{i_1}\}$, respectively). The semantic value of the subject NP and the intransitive verb, Vi, being lexical categories, are the same as those of the words they dominate. The semantic value of the VP, as specified in the appropriate semantic rule, is simply the same as that of the intransitive verb. Finally, the semantic value of the S is $\mathbf{1}$, i.e. the sentence is true, because the semantic value of the subject NP ($\mathbf{i_1}$) is a member of the set that is the semantic value of the verb phrase ($\{\mathbf{i_1}\}$).

1.2.2 Transitive verb phrases

Having illustrated some basic concepts *via* intransitive verbs, we now extend the treatment to cover the semantics of transitive verbs such as *"killed"*. The standard treatment of the semantics of transitive verbs is to consider them as denoting **relations** between two sets of individuals: one set containing those individuals corresponding to the denotation of the subject NP and the other set containing those corresponding to the denotation of the object NP. Each pair of individuals of whom the relation holds constitutes an **ordered pair**, notated $\langle x, y\rangle$, where x is a member of the first set (called the **domain** of the relation) and y a member of the second (called the **range**). The relation can then be characterised as the set of such ordered pairs. In the case of the model being used here, the relation is between $\mathcal{U}$ and itself. Staying with the same extremely simple model, if we want to represent a state of affairs in which Macbeth killed Duncan, and there are no other killings, we would specify the value assigned to the verb *"killed"* by the interpretation function as follows:

$$\mathcal{I}(\text{"killed"}) \quad = \quad \{\langle \mathbf{i_2}, \mathbf{i_1}\rangle\}$$

a set whose members are one ordered pair. This is to be interpreted as representing that the individual $\mathbf{i_2}$ stands in the 'killed' relation to $\mathbf{i_1}$ (and that nobody else is

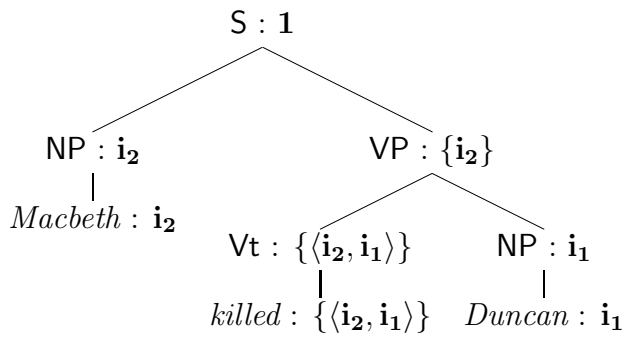

Figure 1.2: Interpreted tree for *"Macbeth killed Duncan"*

involved in killing in the model).[7] (Obviously, if Macbeth had also killed himself, the relation would contain another ordered pair, giving $\{\langle \mathbf{i_2}, \mathbf{i_1}\rangle, \langle \mathbf{i_2}, \mathbf{i_2}\rangle\}$: more killings means more pairs in the relation.)

We have said that the semantic value of any transitive verb denotes a set of ordered pairs. We also know that the object NP contributes an individual. To determine whether a sentence containing a transitive verb phrase is true, we need to have access to the set of individuals who form the domain of the relation and we can do this with the following piece of notation (where Vt abbreviates 'transitive verb'):

$$\{x \mid \langle x, y\rangle \in [\![\mathsf{Vt}]\!]^M\}$$

As readers will know, the notation $\{x \mid \ldots\}$ is read 'the set of xs such that…'. The condition in the above is that x is the first coördinate of an ordered pair which is a member of the semantic value of the transitive verb. In our example $[\![$ *"killed"* $]\!]^M = \{\langle \mathbf{i_2}, \mathbf{i_1}\rangle\}$, so the set picked out by the expression above is $\{\mathbf{i_2}\}$. The semantic rule for transitive VPs needs to ensure that the semantic value of the VP consists of this set of individuals, where the object NP contributes the value of y:

Syntax	Semantics
$\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$;	$[\![\mathsf{VP}]\!]^M = \{x \mid \langle x, [\![\mathsf{NP}]\!]^M\rangle \in [\![\mathsf{Vt}]\!]^M\}$

This rule, in conjunction with the others provided earlier, gives the semantic values for the constituents of the sentence *"Macbeth killed Duncan"* illustrated in the tree in figure 1.2.

In the figure, the semantic values of the words are again given by the interpretation function, and those of the lexical categories are the same as those of the words they dominate. The semantic value of the VP, given that $[\![\mathsf{NP}]\!]^M$ for the object NP is $\mathbf{i_1}$ and that $[\![\mathsf{Vt}]\!]^M$ is $\{\langle \mathbf{i_2}, \mathbf{i_1}\rangle\}$, is the set of x such that $\langle x, \mathbf{i_1}\rangle$ is a member of $\{\langle \mathbf{i_2}, \mathbf{i_1}\rangle\}$, which is the set $\{\mathbf{i_2}\}$. Finally, the semantic value of the S is **1** (true) because the denotation of the subject NP ($\mathbf{i_2}$) is a member of the denotation of the VP ($\{\mathbf{i_2}\}$).

[7] The set of *all* ordered pairs drawn from two sets A and B is called the **cross product** of A and B, written $A \times B$. For example, for the set $\mathcal{U}$ in our model,

$$\mathcal{U} \times \mathcal{U} = \{\langle \mathbf{i_1}, \mathbf{i_1}\rangle, \langle \mathbf{i_1}, \mathbf{i_2}\rangle, \langle \mathbf{i_2}, \mathbf{i_1}\rangle, \langle \mathbf{i_2}, \mathbf{i_2}\rangle\}$$

Any particular relation in our model, such as that denoted by *"killed"*, will be a *subset* of this cross product $\mathcal{U} \times \mathcal{U}$.

1.3 Functions

We have used simple set-theoretic concepts in introducing the basic ideas needed to provide an explicit compositional account of the truth conditions of declarative sentences in the belief that such an account lends itself to intelligibility. It would be perfectly feasible to continue to develop this discussion of semantics using only sets and set membership. However, the literature on truth conditional semantics typically provides an equivalent account which uses **functions** rather than sets. Because one of our aims in this book is to make the linguistics literature accessible, we will now recast the account given above in terms of functions.[8]

A function is, like a relation, a set of ordered pairs. Functions are, however, more restricted than relations in that no individual in the domain of the function may be paired with more than one individual in the range. This means that whereas {⟨**demetrius**, **hermia**⟩, ⟨**demetrius**, **helena**⟩} is a relation, it is not a function (because **demetrius** is paired with more than one individual in the range), but {⟨**lysander**, **hermia**⟩, ⟨**demetrius**, **helena**⟩} *is* a function (because no object in the domain is paired with more than one individual in the range).

In addition to the ordered pair notation of functions, there is a variant known as **functional notation**. Using this notation, the function

$$\{\langle \textbf{lysander}, \textbf{hermia}\rangle, \langle \textbf{demetrius}, \textbf{helena}\rangle\}$$

looks like this:

$$\begin{array}{rcl} f(\textbf{lysander}) & = & \textbf{hermia} \\ f(\textbf{demetrius}) & = & \textbf{helena} \end{array}$$

assuming that we call the function f. Using this notation we talk of **lysander** being an **argument** of the function and **hermia** being the **value** of the function at the argument **lysander**. We also talk of **applying** the function to its argument.

One particular kind of function allows us to express the notion of set membership in functional terms. This is called the **characteristic function** of a set (because it 'characterises' the membership of the set). The characteristic function f_A of a set A is defined as follows: for each object x in the universe of discourse $\mathcal{U}$,

$$f_A(x) = \begin{cases} \textbf{1} & \text{if } x \in A \\ \textbf{0} & \text{otherwise} \end{cases}$$

With this definition, the set denoted by *"died"* in the model, i.e. $\{\mathbf{i_1}\}$, can be represented instead by the following characteristic function:

$$\{\langle \mathbf{i_1}, \mathbf{1}\rangle, \langle \mathbf{i_2}, \mathbf{0}\rangle\}^9$$

The only argument to this function which receives the value **1** is $\mathbf{i_1}$, who is the sole member of the original set.

This equivalence allows us to convert freely between sets and functions.

[8] We will see below that this reformulation also has beneficial consequences when we come to consider computational aspects.

[9] Or, in functional notation, calling the function f_{died}:

$$\begin{array}{rcl} f_{died}(\mathbf{i_1}) & = & \mathbf{1} \\ f_{died}(\mathbf{i_2}) & = & \mathbf{0} \end{array}$$

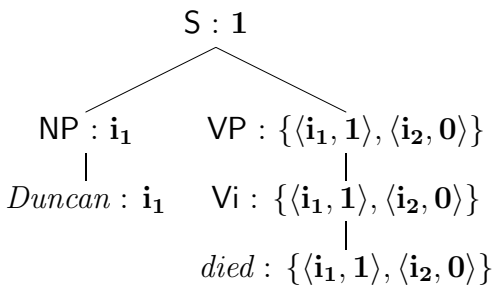

Figure 1.3: Revised interpreted tree for *"Duncan died"*

1.3.1 Simple English expressions

We now have enough apparatus to rework our initial example in functional rather than set-theoretic terms. For convenience, we recapitulate the model (with the interpretation of *"died"* revised):

$$\mathcal{U} = \{\mathbf{i_1}, \mathbf{i_2}\}$$

$$\begin{array}{lcl} \mathcal{I}(\textit{"Duncan"}) & = & \mathbf{i_1} \\ \mathcal{I}(\textit{"Macbeth"}) & = & \mathbf{i_2} \\ \mathcal{I}(\textit{"died"}) & = & \{\langle \mathbf{i_1}, \mathbf{1}\rangle, \langle \mathbf{i_2}, \mathbf{0}\rangle\} \end{array}$$

Under the revision, the value that $\mathcal{I}$ assigns to the intransitive verb *"died"* is a characteristic function instead of a set.

The grammar and corresponding revised semantic rules look like this:

- If W is a lexical item, then $[\![W]\!]^M$ is the value assigned to W by $\mathcal{I}$
- For any lexical entry
$$W : C,$$
where W is a lexical item, then $[\![C]\!]^M = [\![W]\!]^M$
- English syntax: $\mathsf{VP} \rightarrow \mathsf{Vi}$; Semantics: $[\![\mathsf{VP}]\!]^M = [\![\mathsf{Vi}]\!]^M$
- English syntax: $\mathsf{S} \rightarrow \mathsf{NP\ VP}$; Semantics: $[\![\mathsf{S}]\!]^M = [\![\mathsf{VP}]\!]^M([\![\mathsf{NP}]\!]^M)$

The only significant difference here is the rule for obtaining the semantic value of S. This is now obtained by applying the semantic value of VP (which is a characteristic function) to the semantic value of NP. For example, if we work out the semantic value of *"Duncan died"* in the revised framework, we get the interpreted tree shown in figure 1.3. In this figure, denotations of words, lexical categories and the VP are straightforward (although, in the case of the Vi and the VP, the denotations are now functions). Only the denotation of the S is computed any differently: the denotation of the VP, which is the function $\{\langle \mathbf{i_1}, \mathbf{1}\rangle, \langle \mathbf{i_2}, \mathbf{0}\rangle\}$, is applied to the denotation of the subject NP, $\mathbf{i_1}$. The value of the function at the argument $\mathbf{i_1}$ is $\mathbf{1}$, hence the semantic value of the sentence is $\mathbf{1}$ (true).

1.3.2 Transitive verb phrases

Transitive verbs can be assigned a similar, if more complex, functional interpretation. The important central point to hold onto here is the idea that VPs will denote characteristic functions of sets of individuals, no matter what their internal structure may be. In other words, they always denote sets of individuals.

In the example above involving the intransitive verb *"died"*, the denotation of the VP *"died"* is the (characteristic function of) the set of individuals who died (in the model). A VP such as *"killed Duncan"*, which contains the transitive verb *"killed"*, also denotes (the characteristic function of) a set: in this case the set of individuals that killed Duncan. We continue to assume that the semantic value of *"Duncan"* in this VP is an individual, since we have no reason to assume anything to the contrary, and it obviously makes our treatment of the semantics of English more straightforward if NPs like *"Duncan"* always denote the same thing, irrespective of their grammatical function (subject, direct object or indirect object). Therefore, in a framework in which the semantic value of a complex constituent (such as *"killed Duncan"*) is to be computed *via* the application of a function to an argument, we are led to the conclusion that the semantic value of the transitive verb *"killed"* must be a function. Furthermore, it must be a function from individuals (e.g. $[\![\text{ "Duncan"}]\!]^M$) to the semantic value of VPs (i.e. characteristic functions of sets of individuals). *"Killed"* therefore denotes a function from individuals, such as $\mathbf{i_1}$, to the (characteristic function of) the set of individuals that killed $\mathbf{i_1}$.

This leads to the conclusion that the semantic value for transitive VPs should be specified as follows:

- English syntax: VP → Vt NP; Semantics: $[\![\mathsf{VP}]\!]^M = [\![\mathsf{Vt}]\!]^M([\![\mathsf{NP}]\!]^M)$

The semantic value of a transitive VP is arrived at by applying the semantic value of the transitive verb to the semantic value of the object NP.

It remains to specify what the semantic value of a given transitive verb is. We can formalise the relational interpretation ($\mathcal{I}($ *"killed"*$) = \{\langle \mathbf{i_2}, \mathbf{i_1}\rangle\}$) assigned to *"killed"* in subsection 1.2.2 above in functional terms as follows:

$$\begin{aligned}\mathcal{I}(\text{ "killed"}) \quad = \quad & \{\langle \mathbf{i_1}, \langle \mathbf{i_2}, \mathbf{1}\rangle\rangle,\\ & \langle \mathbf{i_1}, \langle \mathbf{i_1}, \mathbf{0}\rangle\rangle,\\ & \langle \mathbf{i_2}, \langle \mathbf{i_1}, \mathbf{0}\rangle\rangle,\\ & \langle \mathbf{i_2}, \langle \mathbf{i_2}, \mathbf{0}\rangle\rangle \ \}^{10}\end{aligned}$$

The relational version contains only one pair, $\langle \mathbf{i_2}, \mathbf{i_1}\rangle$; only $\mathbf{i_2}$ (Macbeth) killed $\mathbf{i_1}$ (Duncan). The functional version contains only *one* entry which is assigned the ultimate value **1** (that is, 'true'), namely $\langle \mathbf{i_1}, \langle \mathbf{i_2}, \mathbf{1}\rangle\rangle$. Rather surprisingly, perhaps, it appears as if the two individuals are now written in the reverse order, and hence that

[10] An alternative representation of the same information, which may be easier to follow is:

$$\begin{aligned}\mathcal{I}(\text{ "killed"}) \quad = \quad & \{\langle \mathbf{i_1}, \{\langle \mathbf{i_2}, \mathbf{1}\rangle, \langle \mathbf{i_1}, \mathbf{0}\rangle\}\rangle,\\ & \langle \mathbf{i_2}, \{\langle \mathbf{i_1}, \mathbf{0}\rangle, \langle \mathbf{i_2}, \mathbf{0}\rangle\}\rangle \ \}\end{aligned}$$

This shows more graphically the set of values that the function associates with the individuals that form its first arguments.

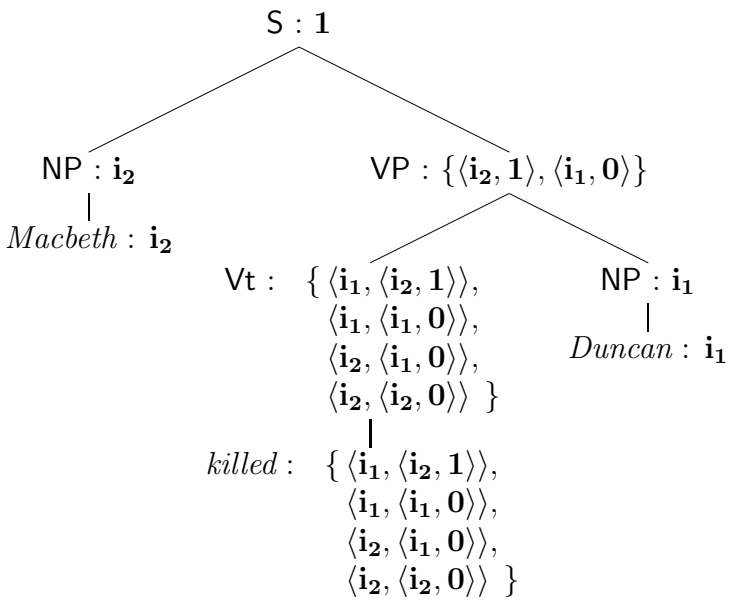

Figure 1.4: Revised interpreted tree for *"Macbeth killed Duncan"*

we have somehow got the participants in the killing relation reversed. This is not the case, as we can demonstrate by showing how the semantic value of the sentence *"Macbeth killed Duncan"* is computed on the basis of the information provided above. This is shown in the interpreted tree in figure 1.4. In the figure, the application of the function denoted by the transitive verb to the denotation of the NP *"Duncan"* involves identifying those ordered pairs in the function whose first coördinate is $\mathbf{i_1}$ (i.e. $[\![\textit{"Duncan"}]\!]^M$). These are $\langle \mathbf{i_1}, \langle \mathbf{i_2}, \mathbf{1} \rangle \rangle$ and $\langle \mathbf{i_1}, \langle \mathbf{i_1}, \mathbf{0} \rangle \rangle$. The value of the function at this argument is given by the second coördinates of these pairs: the set $\{\langle \mathbf{i_2}, \mathbf{1} \rangle, \langle \mathbf{i_1}, \mathbf{0} \rangle\}$. This is the semantic value of the VP according to the rule above.

Note that the revised semantic rule for the S → NP VP rule does not require any further modification, since we have ensured that both intransitive and transitive VPs have the same kind of denotation, namely characteristic functions.

We conclude by pointing out that our treatment of the semantics of these sentences has been compositional: we have specified the semantic values of each of the lexical items in the sentence, and have then specified for each constituent how its semantic value is constructed from the semantic values of its parts. One of the advantages of moving to a functional (as opposed to a purely set-based) treatment of semantics is that we are able to define a uniform mode for the construction of the meanings of complex constituents out of their parts: functional application. In our examples, one of the children in each construction denotes a function, while the other denotes that function's argument.

1.4 Logic

So far we have been interpreting the syntactic structures of English directly and it would be perfectly feasible to continue to develop our analysis in these terms.[11] However, it is more common in work on natural language semantics and in computational semantics to see some form of logical notation being used to mediate between natural language and semantic value. The technique is to translate each natural language expression into an expression of a suitable logic for which there exist standard techniques for assigning model theoretic interpretations. We can then 'trade off' the latter to provide an interpretation indirectly for the natural language expression. Using logic to mediate between natural language and semantic value has a number of practical benefits: as you will appreciate already, directly interpreting natural language expressions can be rather difficult to follow.

We assume some familiarity with First-Order Logic (FOL), so the following account of its syntax and semantics is primarily to provide a summary of the notation and terminology we will be using.

Expressions in FOL are constructed out of a set of **basic expressions**. At this stage we will limit ourselves to constants, of which there are two kinds: (i) individual constants, and (ii) predicate constants. The predicate constants are subdivided according to the number of individual constants they may combine with; so we have 1-place predicate constants, 2-place predicate constants, and so on. All these constants are represented by arbitrary symbols, but we will follow common practice in using single lower case letters (e.g. m, d, etc.) for individual constants. For predicate constants we will use English words, with an integer added to them (e.g. $died1$, $killed1$, etc.). This allows us, firstly, to draw a clear distinction between expressions of English and expressions of logic, and, secondly, to provide logically unambiguous translations for lexically ambiguous English words. For example, we can represent one meaning of the noun *"bank"* as $bank1$ and another as $bank2$, and we can represent the intransitive instance of *"drinks"* (as in *"Toby drinks"*) as $drink1$, and the transitive one (as in *"Toby drinks scotch"*) as $drink2$.

Predicate constants and individual constants can be combined, in accordance with a **formation rule**, to form formulae. The rule will allow us to construct formulae such as $died1(d)$ or $killed1(m, d)$. Although the intended interpretation of these expressions may be fairly clear, it is necessary to specify explicitly how to assign semantic values to logical expressions, just as we assigned semantic values to expressions of English above. Table 1.1 summarises the syntax of FOL, in the left-hand column, and, in the right-hand column, provides an explicit statement prescribing how a semantic value is assigned to the expressions in the left-hand column.[12] (In the table, 'iff' abbreviates 'if and only if'.)

To illustrate these definitions, we need to define a set of basic expressions and also a model with reference to which the logical expressions are to be interpreted. Here is a simple example. The constants we will use are:

[11] See, for example, Cooper (1983), for an extended treatment along these lines.

[12] We are deliberately over-simplifying in this presentation, and, to keep matters as uncomplicated as possible at this stage, are omitting other logical expressions such as quantifiers and connectives, which will not play any part in the discussion of this chapter.

Syntax	**Semantics**
Basic expressions	
1.1.1. A set of **individual constants**, here represented by expressions such as d, m.	2.1.1. If α is an individual constant, then $[\![\alpha]\!]^M = \mathcal{I}(\alpha)$. (That is, the semantic value of an individual constant is simply that which is assigned to it by the interpretation function.)
1.1.2. A set of n-place **predicate constants** (where $1 \leq n$), here represented by expressions such as *died*1 (1-place), and *killed*1 (2-place).	2.1.2. If α is a predicate constant, then $[\![\alpha]\!]^M = \mathcal{I}(\alpha)$. (That is, the semantic value of a predicate constant is simply that which is assigned to it by the interpretation function.)
Formation rules	
1.2.1. If P is an n-place predicate and $a_1, \ldots, a_n$ are individual constants (where $1 \leq n$), then $P(a_1, \ldots, a_n)$ is a **formula**.	2.2.1. If P is an n-place predicate and $a_1, \ldots, a_n$ are individual constants, then $[\![P(a_1, \ldots, a_n)]\!]^M = \mathbf{1}$ iff $\langle [\![a_1]\!]^M, \ldots, [\![a_n]\!]^M \rangle \in [\![P]\!]^M$
1.2.2. Nothing else is a formula.	

Table 1.1: The syntax and semantics of a fragment of FOL

Individual constants:	d, m
1-place predicate constants:	$died1$
2-place predicate constants:	$killed1$

and the universe and interpretation function are defined as follows:

$$\mathcal{U} = \{\mathbf{i_1}, \mathbf{i_2}\}$$

$$\begin{array}{lcl} \mathcal{I}(d) & = & \mathbf{i_1} \\ \mathcal{I}(m) & = & \mathbf{i_2} \end{array}$$

$$\begin{array}{lcl} \mathcal{I}(died1) & = & \{\mathbf{i_1}\} \\ \mathcal{I}(killed1) & = & \{\langle \mathbf{i_2}, \mathbf{i_1} \rangle\} \end{array}$$

The definitions of basic expressions and the formation rule 1.2.1 license expression of the following kind as formulae:

$$died1(d)$$
$$died1(m)$$
$$killed1(d, m)$$
$$killed1(m, m)$$

As we see, $died1(d)$ is a well-formed formula of FOL according to the rules given in the table. The corresponding semantic rules state that $[\![died1(d)]\!]^M = \mathbf{1}$ if and only if $\langle [\![d]\!]^M \rangle \in [\![died1]\!]^M$.[13] Consulting the model reveals that $[\![d]\!]^M = \mathcal{I}(d) = \mathbf{i_1}$, and $[\![died1]\!]^M = \mathcal{I}(died1) = \{\mathbf{i_1}\}$. Since $\mathbf{i_1} \in \{\mathbf{i_1}\}$, the formula $died1(d)$ is true ($= \mathbf{1}$) with respect to this model.

Similarly, $killed1(m, d)$ is a well-formed formula. Its semantic value, $[\![killed1(m, d)]\!]^M$ is calculated in an analogous fashion: $[\![killed1(m, d)]\!]^M = \mathbf{1}$ if and only if $\langle [\![m]\!]^M, [\![d]\!]^M \rangle \in [\![killed1]\!]^M$. $[\![m]\!]^M = \mathcal{I}(m) = \mathbf{i_2}$ and $[\![d]\!]^M = \mathcal{I}(d) = \mathbf{i_1}$. Since $[\![killed1]\!]^M = \mathcal{I}(killed1) = \{\langle \mathbf{i_2}, \mathbf{i_1} \rangle\}$, we now need to check whether $\langle \mathbf{i_2}, \mathbf{i_1} \rangle \in \{\langle \mathbf{i_2}, \mathbf{i_1} \rangle\}$. Since this is true, it follows that $[\![killed1(m, d)]\!]^M = \mathbf{1}$.

Comparison of the truth values for $died1(d)$ and $killed1(m, d)$ with those that we arrived at in our discussion of the direct interpretation of English expressions will reveal that the following pairs of English sentences and logical formulae have the same truth conditions with respect to the model we have set up:

English	First Order Logic
"Duncan died"	$died1(d)$
"Macbeth killed Duncan"	$killed1(m, d)$

Although it is indeed the intention that they should have the same truth conditions, we have not yet established any systematic correspondence between them. Our task in what follows is to explain how to get from the English expressions in the left-hand column to their FOL translation in the right-hand column and how these representations relate to what we have covered so far.

[13] An ordered monad, i.e. a tuple with just a single element in it, such as $\langle [\![d]\!]^M \rangle$, is simply equivalent to $[\![d]\!]^M$, so $\langle [\![d]\!]^M \rangle = [\![d]\!]^M$.

To take up the latter point first, we have been developing a treatment of semantics which relies heavily on the use of functions, but functions have not figured in the account of FOL that we have presented so far. We therefore need to add function-denoting expressions to FOL if we are to be able to carry out the programme of mediating the interpretation of English through FOL. We will do this by first introducing the notion of 'type'.[14]

1.5 Type theory

It is common in work on natural language semantics to use **type theory** to 'glue' the different aspects of the analysis together. The **semantic types** used in type theory can be thought of as forming classes out of the kinds of objects that can be constructed out of the universe of discourse. For example, we have seen that different expressions of English denote individual entities, truth-values, characteristic functions of sets of entities, and characteristic functions of characteristic functions of sets of entities. Clearly, many other kinds of objects can be constructed out of the objects in the universe of discourse. These classes are the semantic types, and we need a way of naming each of them.

The definition of the types (and their names) is as follows. There are two basic types, named $\mathbf{e}$ and $\mathbf{t}$. In addition, there is a recursive definition of a type:

> If a is a type and b is a type, then $\langle a, b\rangle$ is a type.

These definitions license an infinite set of types such as the following:

$$\begin{array}{lll} \langle \mathbf{e}, \mathbf{t}\rangle & \langle \mathbf{e}, \mathbf{e}\rangle & \langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle \\ \langle \mathbf{t}, \mathbf{e}\rangle & \langle \mathbf{t}, \mathbf{t}\rangle & \langle \mathbf{t}, \langle \mathbf{e}, \mathbf{t}\rangle\rangle \end{array}$$

Semantically, expressions of type $\mathbf{e}$ denote individuals (entities), and expressions of type $\mathbf{t}$ denote truth values. Expressions of type $\langle a, b\rangle$ denote functions from denotations of type a to denotations of type b. Consequently, an expression of type $\langle \mathbf{e}, \mathbf{t}\rangle$ is a function from individuals (type $\mathbf{e}$) to truth values (type $\mathbf{t}$); in other words, it is a characteristic function of a set of individuals.

In the fragment of English that we covered earlier, the semantic type of English names and NPs is $\mathbf{e}$, the type of sentences is $\mathbf{t}$, the type of intransitive verbs and of verb phrases is $\langle \mathbf{e}, \mathbf{t}\rangle$, and that of transitive verbs is $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$.

In the fragment of logic that we presented in the previous section, the type of individual constants is $\mathbf{e}$, and the type of formulae is $\mathbf{t}$. The fragment does not, as yet, contain any expressions that denote functions (in particular, it has no expressions whose semantic types are $\langle \mathbf{e}, \mathbf{t}\rangle$ and $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$).

There are two main senses in which type theory will now 'glue' together different aspects of the analysis. Firstly, when pairing an English expression with a logic expression, we will ensure that paired expressions have the same semantic type. (This already suggests that English names should be translated by individual constants of

[14] The extension of FOL which follows is brutally informal. We should in fact provide a completely revised definition of the formal language. The presentation adopted here is chosen for brevity and for perspicuity. More extensive and detailed treatments of this topic can be found in Dowty et al. (1981, chapter 4) and Partee et al. (1990, pp.338-51).

the logic, since both are of type **e**, and that English sentences should be translated by formulae of the logic, since both are of type **t**. It also highlights the point that English verbs, which denote functions of various kinds, have no correlate in our logic fragment so far, as nothing in that fragment is function-denoting.) Secondly, we will use type theory (in the next section) as a constraint on the syntactic expressions that we add to our fragment of logic.

1.6 The lambda operator

Having introduced functional types, we now use them to give an extended logic by defining a new class of expression: **lambda expressions**. In order to do this, we need to extend the basic expressions by including, in addition to the set of constants, a set of **variables**.

The set of variables is divided into variables of different types. Just as there are individual constants, there are also individual variables, which we will represent by lower case letters chosen from the end of the alphabet, augmented by subscript numerals if necessary: $x, y, z, x_1, y_2, \ldots$ Individual variables have the same syntactic distribution as individual constants: they occur as arguments to predicates. Hence, $died1(x)$ and $killed1(m, y)$, in which individual variables (x and y) appear where individual constants would normally appear, are well-formed formulae. Individual variables have the same type as the corresponding constants, namely type **e**.

But, there are also variables of types $\langle \mathbf{e}, \mathbf{t} \rangle$, $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$, and $\langle \mathbf{e}, \langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle \rangle$, and so on, which we will represent by possibly-subscripted letters from the middle of the alphabet, such as $P, Q, P_1, Q_2 \ldots$ These variables will have the same syntactic distribution as predicate constants, hence $P(d)$ is a well-formed formula. It is also possible to have variables of other types and hence other syntactic distributions too.

In fact, in this chapter we only use individual variables (type **e**). The reason we have mentioned the existence of other variables is that our syntactic definitions of lambda expressions (below) are completely general, allowing the construction of lambda expressions that contain variables of any type, and this generality will be needed in future chapters.[15]

1.6.1 The syntax of lambda expressions

Equipped with variables, we define a new kind of logical expression which makes use of them.

1. If v is a variable of type a and ϕ is an expression of type b, then $\lambda v[\phi]$ is an expression of type $\langle a, b \rangle$.[16]
   (This rule introduces the new kind of logical expression.)

[15] Readers with considerable familiarity with logic will realise that by allowing variables other than simple individual variables we are extending our logical language from being a fragment of *First-Order Logic* (FOL) to being a variety of *Higher-Order Logic* (HOL). We will continue to refer to the logic as 'FOL' in this chapter, in recognition of the fact that we are not yet using the generality we have afforded ourselves.

[16] λ is the Greek letter 'lambda'.

2. If α is an expression of type a and β is an expression of type $\langle a, b\rangle$, then $\beta(\alpha)$ is an expression of type b.
   (This rule allows us to make use of the expressions defined in 1.)

For example, $died1(x)$ is an expression of type $\mathbf{t}$ (since it is a formula) and, since x is a variable of type $\mathbf{e}$ (appearing, as it does in $died1(x)$, where an individual constant would normally appear), $\lambda x[died1(x)]$ is an expression of type $\langle \mathbf{e}, \mathbf{t}\rangle$ (by rule 1). Rule 2 allows us to take such expressions defined by rule 1 and combine them with other expressions of the logic, so we can take $\lambda x[died1(x)]$ and combine it, as licensed by rule 2, with an expression of type $\mathbf{e}$, such as d, to give a new expression: $\lambda x[died1(x)](d)$. By virtue of rule 2, we know that the type of this expression is $\mathbf{t}$, in other words, a formula.[17]

With these new lambda-expressions, we have now introduced functional expressions into an extended FOL. What we need to turn to next is how their semantics is defined.

1.6.2 The semantics of lambda expressions

For the purposes of this chapter, we will give a highly informal presentation of the semantics of our new expressions. A more formal treatment is given in chapter 2.

The (informal) semantic counterpart of syntax rule 1 given in subsection 1.6.1 is that if v is a variable of type a, and ϕ is an expression of type b, then $[\![\lambda v[\phi]]\!]^M$ is a function of an appropriate type. As an example, if x is an individual variable (type $\mathbf{e}$), and $died1(x)$ is a formula (type $\mathbf{t}$), then rule 1 says that $\lambda x[died1(x)]$ is an expression of type $\langle \mathbf{e}, \mathbf{t}\rangle$ and so its semantic value is some function from individuals to truth values.

The (informal) semantic counterpart of syntax rule 2 given in subsection 1.6.1 is that if v is a variable of type a, α is also an expression of type a, and ϕ is an expression of type b, then

$$[\![\lambda v[\phi](\alpha)]\!]^M \equiv [\![\phi^{\alpha/v}]\!]^M$$

where $\phi^{\alpha/v}$ is the result of replacing all occurrences of v in ϕ by α. That is to say, given expressions of appropriate types, we find the semantic value of a lambda expression and its argument by replacing all instances of the variable in the body of the expression by the argument.[18] As an example, given that $\lambda x[died1(x)]$ is an expression of type $\langle \mathbf{e}, \mathbf{t}\rangle$ and the individual constant d is of type $\mathbf{e}$, then $\lambda x[died1(x)](d)$ is an expression

[17] To emphasise the generality of the new syntax rules, we show that $\lambda P[P(d)]$ and $\lambda P[P(d)](\lambda x[died1(x)])$ are also licensed by the above rules (even though they are not examples used elsewhere in this chapter). $P(d)$ is an expression of type $\mathbf{t}$ (a formula), with P being a variable of type $\langle \mathbf{e}, \mathbf{t}\rangle$. Hence, $\lambda P[P(d)]$ is an expression of type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$ by rule 1. Given that $\lambda x[died1(x)]$ is an expression of type $\langle \mathbf{e}, \mathbf{t}\rangle$ (the same type as P), then $\lambda P[P(d)](\lambda x[died1(x)])$ is an expression of type $\mathbf{t}$ by rule 2.

[18] Readers with considerable familiarity with logic will note the informality of this treatment. We ought to add to the semantics some apparatus for assigning semantic values to variables and use this in giving semantic values to lambda expressions and uses of those expressions. We give this reformulation in chapter 2. The simpler treatment given here assigns the same semantic value as chapter 2's treatment (provided ϕ contains only free occurrences of the variable v), but takes a more 'syntactic' route (by textually substituting arguments for variables in the body of the lambda expression).

of type **t**, and to obtain the semantic value of this expression, $[\![\lambda x[died1(x)](d)]\!]^M$, we replace all occurrences of x within the body of the lambda expression by d, and take the semantic value of the resulting expression. This means that $[\![\lambda x[died1(x)](d)]\!]^M \equiv [\![died1(d)]\!]^M$. Referring to our model from section 1.4, $[\![died1(d)]\!]^M$ is true if and only if $[\![d]\!]^M \in [\![died1]\!]^M$, i.e. if and only if $\mathbf{i_1} \in \{\mathbf{i_1}\}$, which in this case is true. So both expressions are true.

Note that in saying that $[\![\lambda x[died1(x)](d)]\!]^M \equiv [\![died1(d)]\!]^M$, we are saying that these two expressions have exactly the same truth values in all models: they **entail** one another. (One expression entails another if, in all models in which the first expression is true, the second is also true; obviously, if two expressions entail one another, then they must be true in exactly the same models.) This equivalence means that these two expression are interchangeable.

The equivalence licences two operations, known as lambda abstraction and lambda conversion.

1.6.3 Lambda abstraction

We show the way **lambda abstraction** works very schematically below.

From any expression of the form:

$$\ldots\alpha\ldots$$

where α is some expression of type a, we can construct a semantically equivalent expression of the form:

$$\lambda v[\ldots v \ldots](\alpha)$$

where v is a variable of the same type as α, by replacing the original α by v.

For example, from $died1(d)$, we can, by lambda abstraction, construct the semantically equivalent expression $\lambda x[died1(x)](d)$. (Here both x and d are expressions of type **e**.)

1.6.4 Lambda conversion

The semantic equivalence also licences the inverse operation, **lambda conversion**.[19]

From any expression of the form:

$$\lambda v[\ldots v \ldots](\alpha)$$

where α is some expression of type a and v is a variable of the same type, we can construct a semantically equivalent expression of the form:

$$\ldots\alpha\ldots$$

by replacing v with α.

So, given $\lambda x[died1(x)](d)$, we can replace it with the semantically equivalent $died1(d)$. (Here, again, x and a are both of type **e**.)

[19]This is also known as **beta-reduction** in the computer science literature. And, indeed, it is this latter term that we will use below in the computational implementation of the ideas being presented in this section.

English	Category	FOL Translation	Type	Denotation
"Duncan"	NP	d	$\mathbf{e}$	Individual
"Macbeth"	NP	m	$\mathbf{e}$	Individual
"died"	Vi	$\lambda x[died1(x)]$	$\langle \mathbf{e}, \mathbf{t} \rangle$	characteristic function (of the set of individuals that died)
"Duncan died"	S	$died1(d)$	$\mathbf{t}$	truth value

Table 1.2: Some correspondences between English and FOL

1.7 From English to logic

Now that we have introduced functional expressions into an extension of FOL, we are in a position to reconstruct the earlier 'direct' interpretations of English sentences using logical translations as an intermediary between English and semantic values. Table 1.2 gives some of the relevant correspondences between English and FOL, where, in pairing an expression of English with a FOL translation, we have ensured that the denotations and semantic types of the two are the same. A fuller explanation is given in the rest of this section.

We pair each syntactic expression of English with a FOL translation by defining a function that translates each English expression into a logical one. For example, the word *"Duncan"* translates as d and *"died"* as $\lambda x[died1(x)]$:

$$\begin{array}{lcl} \text{Translation}(\textit{"Duncan"}) & = & d \\ \text{Translation}(\textit{"died"}) & = & \lambda x[died1(x)] \end{array}$$

We are here using the standard function-argument notation for the translation function (e.g. Translation(*"died"*)). When expressions become more complex, however, this notation becomes rather hard to read, so we will define an abbreviative notation: we will represent Translation(α) by α' (the expression α plus a prime). In this notation, the above translations look like this:[20]

$$\begin{array}{lcl} Duncan' & = & d \\ died' & = & \lambda x[died1(x)] \end{array}$$

As in sections 1.2 and 1.3, each rule in the grammar and lexicon consists of two pieces of information. In the earlier versions, this was a pairing of syntactic information with its semantic value; now it is the pairing of syntactic information with its logical translation.

The lexicon looks like this:

- English syntax — FOL translation
  Duncan : NP; — NP′ = $Duncan' = d$

[20] We will drop the quotation marks round the English expressions in these translations for the sake of tidiness, so, e.g., *"died"′* becomes simply *died′*.

- English syntax: *Macbeth* : NP; FOL translation: $\mathsf{NP}' = \mathit{Macbeth}' = m$

- English syntax: *died* : Vi; FOL translation: $\mathsf{Vi}' = \mathit{died}' = \lambda x[\mathit{died}1(x)]$

We have previously assumed that the semantic value of a VP containing only an intransitive verb is simply that of the verb. In a grammar making use of logical translations, this equates to assigning the VP the same translation as the verb it dominates. We represent this as follows:

- English syntax: VP → Vi; FOL translation: $\mathsf{VP}' = \mathsf{Vi}'$

The question of how to translate the S rule is more interesting. An S consists of an NP and a VP. We know from the lexicon that NPs translate as expressions of type **e**, and we have just seen that intransitive VPs have the same translations as intransitive verbs, expressions of type $\langle \mathbf{e}, \mathbf{t} \rangle$. Since sentences denote truth values (and therefore have translations which are expressions of type **t**), the only available way to combine the two children which will preserve these type assignments is for the function-denoting expression (the translation of the VP) to take the translation of the NP as its argument:

- English syntax: S → NP VP; FOL translation: $\mathsf{S}' = \mathsf{VP}'(\mathsf{NP}')$

Here, the expression $\mathsf{S}' = \mathsf{VP}'(\mathsf{NP}')$ is to be understood as 'the (logical) translation of an S constituent consists of the application of the translation of its VP child to the translation of its NP child'. Note that the use of type theory severely constrains the way that the translations (and hence semantic values) of constituents can be combined, reducing essentially to permissible function/argument applications.

The translations of transitive verbs simply use the same principles as for intransitives, but since they are more complex, it is worth working through the procedure step by step.

We know that the translation of *"Macbeth killed Duncan"* will be the formula $\mathit{killed}1(m, d)$. We know also that the subject NP (*"Macbeth"*) will contribute the translation of *"Macbeth"*, m, and we also know that the VP must denote a function, whose semantic value must be the set of individuals that killed Duncan. This means that to get the translation of the VP we need to abstract over m in $\mathit{killed}1(m, d)$ to get the semantically equivalent expression $\lambda x[\mathit{killed}1(x, d)](m)$. From this we remove the argument m, to get the translation of the VP alone: $\lambda x[\mathit{killed}1(x, d)]$. Having arrived at the translation of the VP, we now remove from it the contribution made by the object NP *"Duncan"*, d, by abstracting over the d to give $\lambda y[\lambda x[\mathit{killed}1(x, y)]](d)$. (Note that the rules for lambda expressions given above do not in any way block the application of lambda abstraction to expressions which are themselves lambda expressions.) From this, we remove the argument d, leaving the translation of the transitive verb: $\lambda y[\lambda x[\mathit{killed}1(x, y)]]$. Note that the type of this expression is $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$: semantically, a function from individuals to characteristic functions of sets of individuals. In other words, this expression combines with two individual constants to give a formula, just as a transitive verb in English combines with two NPs to give a sentence.

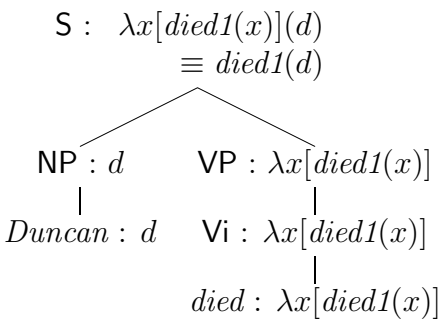

Figure 1.5: Annotated tree for *"Duncan died"*

We are now in a position to add a lexical entry for the transitive verb *"killed"* to the grammar given above, and to add a grammar rule for transitive verbs. Here is the lexical entry:

	English syntax	FOL translation
•	*killed* : Vt;	$\mathsf{Vt}' = killed' = \lambda y[\lambda x[killed1(x, y)]]$

Supplying the translation for a VP containing a transitive verb proves to be tightly constrained, as a consequence of the types assigned to the children. The translation of the transitive verb is of type $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$ and its NP sibling is of type $\mathbf{e}$. This means that the translation of the VP parent is determined by the function application of the verb's translation to that of the object NP:

	English syntax	FOL translation
•	$\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$;	$\mathsf{VP}' = \mathsf{Vt}'(\mathsf{NP}')$

Note that we do not need to revise the translation of the S rule, because, although we have two kinds of VP which differ in their internal structure, the type of the translation of the VP itself remains $\langle \mathbf{e}, \mathbf{t} \rangle$, irrespective of its internal structure.

So, we have now re-worked the same grammar fragment as that introduced earlier: now the semantic correlates of the syntactic rules are replaced by instructions on how to build up logical translations which provide the basis for semantic value. As before, function application is used to combine translations of children constituents into the translation of the parent, where previously function application was used to construct the semantic value directly.

We can show the logic translations of English expressions in the form of **annotated trees**. ('Annotated trees' show the logic translations associated with the English expressions that label each node, in contrast to 'interpreted trees', which show the semantic values associated with the English expressions that label each node.)

This grammar produces the annotated tree shown in figure 1.5 for the sentence *"Duncan died"*. The equivalence between the two formulae given as the translation of the S node is guaranteed by lambda conversion.

In order to evaluate the truth of the formula $died1(d)$ in this tree, we need to calculate its semantic value: $[\![died1(d)]\!]^M$. This is true if and only if $[\![d]\!]^M \in [\![died1]\!]^M$. By referring back to the model, you should be able to confirm that this is indeed the

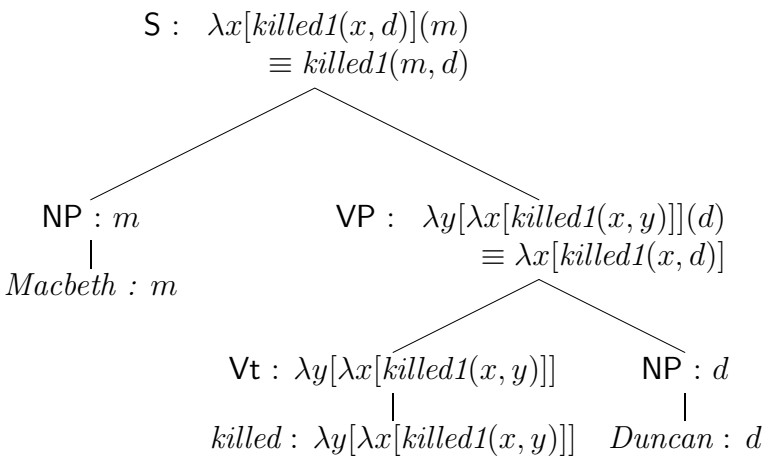

Figure 1.6: Annotated tree for *"Macbeth killed Duncan"*

case. Therefore, $[\![died1(d)]\!]^M = \mathbf{1}$. To complete the story, and assign a semantic value to the English sentence, *"Duncan died"*, we need to add to the analysis the statement that:

- if E is an expression of English and E' is its translation into logic, then $[\![E]\!]^M = [\![E']\!]^M$.

This last requirement ensures that if $[\![died1(d)]\!]^M = \mathbf{1}$, then so does $[\![$ *"Duncan died"* $]\!]^M$.

It should be emphasised very strongly that the logical translation of an English sentence is *not* the semantic value of that sentence. The semantic value for both English expressions and for expressions of a logical language consists of the semantic value assigned with respect to a model. Importantly too, we emphasise that the logical translations are not essential to the treatment. They are merely a convenient intermediate notation used in assigning semantic values to English expressions; we can assign semantic values to English expressions directly, without using logic, as we were doing earlier in this chapter, but it gives a less perspicuous treatment. We will see in chapter 2 that not all theories of semantics give the logical translations this optional status; in some theories, the logic is an essential level of meaning representation, without which correct semantic values would not be assigned.

For completeness, we give, as figure 1.6, an annotated tree showing how the translation of the sentence *"Macbeth killed Duncan"* is constructed. You will see that this mirrors the account we gave above of the reasoning that led to the translation of *"killed"*.

Exercise 1 *In this exercise, you should assume that all the grammar rules and lexical entries (with their FOL translations) given so far in this section of the book remain unchanged.*

The grammar is extended with the following rules:

$$\mathsf{VP} \rightarrow \mathsf{VPd\ NP}$$
$$\mathsf{VPd} \rightarrow \mathsf{Vd\ NP}$$

and the lexicon is extended with the following two lexical entries:

$$\begin{aligned} \text{Caliban} &: \mathsf{NP} \\ \text{sold} &: \mathsf{Vd} \end{aligned}$$

Given that the translation of "Duncan sold Macbeth Caliban" *is* $sell1(d, m, c)$ *give the FOL translations for the new grammar rules and lexical entries.*

Exercise 2 *Suppose that in exercise 1 we had added just the following grammar rule (instead of the two binary rules):*

$$\mathsf{VP} \rightarrow \mathsf{Vd\ NP\ NP}$$

In the light of your answer to exercise 1 (and aiming to achieve the same logical translation for "Duncan sold Macbeth Caliban"*), which of the following would be the semantic rule that would correspond to the above grammar rule:*

$$\begin{aligned} \mathsf{VP}' &= (\mathsf{Vd}'(\mathsf{NP}'_1))(\mathsf{NP}'_2) \\ \mathsf{VP}' &= (\mathsf{Vd}'(\mathsf{NP}'_2))(\mathsf{NP}'_1) \\ \mathsf{VP}' &= \mathsf{Vd}'(\mathsf{NP}'_1(\mathsf{NP}'_2)) \\ \mathsf{VP}' &= \mathsf{Vd}'(\mathsf{NP}'_2(\mathsf{NP}'_1)) \end{aligned}$$

(where NP'_1 *is the translation of the first* NP *and* NP'_2 *is the translation of the second* NP *in the rule* $\mathsf{VP} \rightarrow \mathsf{Vd\ NP\ NP}$*).*

Justify your answer.

Exercise 3 *In this exercise, you again assume that all the grammar rules and lexical entries (with their FOL translations) given so far in this section of the book remain unchanged.*

The grammar is extended with the following rules:

$$\begin{aligned} \mathsf{VP} &\rightarrow \mathsf{Vaux\ Vtpass\ PPby} \\ \mathsf{PPby} &\rightarrow \mathsf{Pby\ NP} \end{aligned}$$

and the lexicon is extended with the following three lexical entries:

$$\begin{aligned} \text{was} &: \mathsf{Vaux} \\ \text{killed} &: \mathsf{Vtpass} \\ \text{by} &: \mathsf{Pby} \end{aligned}$$

These extensions allow generation of simple sentences whose VP*s have* passive *mood, e.g.:*

"Duncan was killed by Macbeth."

It is fairly widely accepted that such a sentence will have identical truth-conditions to the corresponding sentence whose VP *has* active *mood:*

"Macbeth killed Duncan."

(While we assign identical semantics to the two sentences, this leaves open the possibility that they do differ in meaning from a pragmatic point of view, e.g. the passive is 'about' Duncan whereas the active is 'about' Macbeth, and this may determine the contexts in which they may felicitously be uttered.)

Give the FOL translations for the new grammar rules and lexical entries. (Be warned that the answer can involve variables of types other than $\mathbf{e}$*.)*

1.8 Summary

- In truth-conditional semantics, the (minimal) meaning of a (declarative) sentence is a truth value. In model-theoretic semantics, we evaluate the truth or falsity of a sentence against a model of the world. It is normal for expressions to have individuals and functions as their semantic values, which means that the only operation needed for putting semantic values together is function application.
- We assign semantics compositionally, which means that the meaning of a sentence is computed as a function of the meaning of its parts. We achieve this by the rule-to-rule hypothesis, where each syntactic rule is paired with a semantic rule that gives instructions for computing the semantic value.
- Rather than compute semantic values of English expressions directly, it is common to use logic to mediate between English expressions and their semantic values. This requires adding to standard logics expressions that denote functions: lambda expressions.
- Type theory provides a way of ensuring that the logic translations paired with English expressions denote the same kinds of objects. It is also a constraint on the definition and use of the lambda expressions added to the logic language.

The semantic coverage provided in this chapter is obviously grossly incomplete; we have given no discussion of tense (so we cannot handle the difference between *"Duncan sleeps"* and *"Duncan slept"*), we have presented no treatment of the semantics of modifiers (so we cannot handle *"Toby drinks scotch on ice"*, for example). The treatment of the semantics of NPs given in this chapter is deliberately oversimplified. We have made no attempt to consider NPs consisting of anything more than a name. The analysis of NPs as denoting individuals will clearly not extend to more complex cases such as *"each king"* and *"a murderer with a dagger"*. Nonetheless, the material presented here will form the basis of any treatment of more complex issues and although considerations of space will mean that some of them will be left untouched, we will extend the semantic coverage in subsequent chapters.

1.9 Further reading

In this chapter we have provided an introduction to model theoretic semantics and have shown how a logical representation using the lambda operator can be used to mediate between expressions of a natural language and their model-theoretic interpretation. Because of considerations of space, the treatment of both semantics and logic has been informal and incomplete. There are now several very good textbook level treatments of these topics which should be consulted for further details. In particular Chierchia and McConnell-Ginet (1990), Dowty et al. (1981), Gamut vols. I and II (1991), and Cann (1993) cover much of the same territory as that dealt with in this chapter, as well as many additional topics. Partee et al. (1990) also contains much useful textbook level material about natural language semantics. Klein

and Sag (1985) is a journal paper which sets out the compositional semantics programme. Janssen (1996) provides a comprehensive overview of the issues raised by compositionality.

Much of the credit for this approach to natural language semantics is awarded to Richard Montague, although his writings, collected in (Montague 1974), can be difficult work for the reader new to the subject.

Alternative model-theoretic frameworks, that are similar in many respects to the one we have developed in this chapter, but different in many other respects, include Discourse Representation Theory (e.g. Kamp 1981, Kamp and Reyle 1993) and Situation Semantics (e.g. Barwise and Perry 1983). Discourse Representation Theory is an example of a theory whose initial presentation was non-compositional (Kamp 1981) but which has subsequently been re-formulated in compositional terms (e.g. Zeevat 1989). Wide-ranging discussions of Discourse Representation Theory, Situation Semantics and other formalisms are given in the deliverables produced by the FraCaS project consortium, e.g. Cooper et al. (1994a, 1994b).

Chapter 2

The Semantics of Quantified Noun Phrases

The noun phrases whose semantics we developed in chapter 1 were all simple proper nouns (names), such as *"Duncan"* and *"Macbeth"*. We took the denotation of these names to be individuals, thus having semantic type **e**; when we gave their logic translations, we mapped them to individual constant symbols of the logic (which are also of type **e**). But there are, of course, many other types of noun phrase, and it is the semantics of some of these that we examine in this chapter.

Consider the following sentences:

(1) *"Every soldier died."*

(2) *"No witch died."*

(3) *"Some king died."*

which contain the NPs *"every soldier"*, *""no witch"* and *"some king"*. These NPs are generated by the CF-PSG fragment given as grammar 2.1. Although we will work with this grammar initially in this chapter, we note from the outset that it is not a descriptively adequate grammar for noun phrases. In particular it is normal to introduce the category Nbar and to use two grammar rules (NP $\rightarrow$ Det Nbar and Nbar $\rightarrow$ N) in place of the single grammar rule given in grammar 2.1. Nbar is then a suitable category for modification by adjectival phrases, prepositional phrases and relative clauses, for example. We will work with the simpler grammar for now, and then introduce Nbar when it is needed, later in the chapter.

The words *"every"*, *"some"* and *"no"* are determiners. Other determiners include *"the"*, *"a"*, *"all"*, *"each"*, *"most"*, *"many"*, *"few"*, etc.[1] At a semantic level, determiners have a *quantificational* role. The NPs formed using the rule given in grammar 2.1 are referred to as **quantified noun phrases**. Informally, the determiner quantifies the number of individuals being referred to, and the nominal expression (*"soldier"*, *"witch"*, *"king"*, etc.) restricts the quantification to only certain individuals (those which the nominal denotes).[2]

[1]Determiner phrases are also possible, e.g. *"all three witches"*, *"all of the witches"*, *"less than half the witches"*, *"almost all witches"*, etc. We shall ignore these.

[2]Although ignored in the rest of this chapter, certain *pronouns* can also constitute quantified

$\mathsf{NP} \rightarrow \mathsf{Det}\ \mathsf{N}$	*every* : Det	*soldier* : N
	no : Det	*witch* : N
	some : Det	*king* : N

Grammar 2.1: A grammar for some noun phrases

We want to work out the denotations, semantic type, and translations into logic of these quantified NPs. We want to show how their denotations (and logic translations) are composed from the denotations (and logic translations) of their constituents (Det and N), and we want to present the corresponding implementation of the computation of their logical translations using Prolog. This is the subject of this chapter.

In the previous chapter, we considered denotations first and only later moved on to consider the mediating role that logic could play. In this chapter, we will concentrate on giving logical translations (though, inevitably, some discussion of types and denotations will also be necessary). Therefore, as a preliminary, we need to extend the logical language given in the previous chapter so that it contains expressions suitable as the target in our translation, where, by 'suitable', we mean that these new expressions will have the kinds of denotations we want to give to quantified noun phrases of English.

2.1 Logic

As in the previous chapter, we continue to assume that the reader has some familiarity with logic, and so our presentation in this section serves only as a summary of notation and terminology. Table 2.1 summarises the basic expressions and formation rules of the syntax of our extended logic in the left-hand column, and, in the right-hand column, states how to assign semantic values to the expressions in the left-hand column. (In the table, 'iff' abbreviates 'if and only if'.)

As far as the syntax is concerned, note that this time we include references to variables and lambda expressions, exactly as defined in the previous chapter, from the outset. Additionally, definitions 1.2.3 (i) and (ii) introduce the symbols $\forall$ and $\exists$, which are referred to as **quantifiers**, and thus add quantified expressions to the logic. The symbol $\forall$ is referred to as the **universal quantifier**, and the symbol $\exists$ is referred to as the **existential quantifier**. In an expression of the form $\forall v(\phi)$ or $\exists x(\phi)$, the variable v is said to be **bound** by the quantifier; any occurrences of v in ϕ are **bound occurrences** of v; the expression ϕ is said to be the **scope** of the quantifier; variables appearing in an expression that are not bound by a quantifier or the lambda operator are said to be **free occurrences**.

Definitions 1.2.4 (i)-(v) introduce the symbols $\neg, \wedge, \vee, \supset$ and $\leftrightarrow$, which are referred to as **connectives**; they define a set of compound formulae. $\neg(\phi)$ is the **negation** of

noun phrases, e.g. *"someone"*, *"everyone"*, *"something"*, *"everything"*, etc. One way of thinking of these is that the determiner (*"some"* and *"every"* in these cases) and the nominal (*"one"* and *"thing"* here) are stuck together to form a single word.

Syntax	Semantics
Basic expressions	
1.1.1. A set of individual constants (type **e**), here represented by expressions such as d, m.	2.1.1. If α is an individual constant, then $[\![\alpha]\!]^{M,g} = \mathcal{I}(\alpha)$. (That is, the semantic value of an individual constant is simply that which is assigned to it by the interpretation function.)
1.1.2. A set of n-place predicate constants (where $1 \leq n$), here represented by expressions such as *died*1 (1-place, type $\langle \mathbf{e}, \mathbf{t} \rangle$), *killed*1 (2-place, type $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$) and *send*1 (3-place, type $\langle \mathbf{e}, \langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle \rangle$).	2.1.2. If α is a predicate constant, then $[\![\alpha]\!]^{M,g} = \mathcal{I}(\alpha)$. (That is, the semantic value of a predicate constant is simply that which is assigned to it by the interpretation function.)
1.1.3. A set of variables of various types. Where they are of type **e** (individual variables), we will represent them by possibly-subscripted lower case letters from the end of the alphabet (e.g. x, y, z, x_1, y_2); where they are of types $\langle \mathbf{e}, \mathbf{t} \rangle$, $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$, $\langle \mathbf{e}, \langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle \rangle$, etc., we will represent them by possibly-subscripted upper case letters from the middle of the alphabet (e.g. P, Q, P_1, Q_2); where they are of type $\langle \langle \mathbf{e}, \mathbf{t} \rangle, \mathbf{t} \rangle$, we will represent them by possibly-subscripted calligraphic letters from the middle of the alphabet (e.g. $\mathcal{P}, \mathcal{Q}, \mathcal{P}_1, \mathcal{Q}_2$).	2.1.3. If α is a variable, then $[\![\alpha]\!]^{M,g} = g(\alpha)$. (That is, the semantic value of a variable is simply that which is assigned to it by the value assignment function.)

Table 2.1: The syntax and semantics of an extended logic

Syntax	**Semantics**
	Formation rules
1.2.1. If v is a variable of type a, and ϕ is an expression of type b, then $\lambda v[\phi]$ is an expression of type $\langle a, b\rangle$.	2.2.1. If v is a variable of type a, and ϕ is an expression of type b, then $[\![\lambda[\phi]]\!]^{M,g}$ = a function which maps each object $\mathbf{u}$ of type a in $\mathcal{U}$ to the object $[\![\phi]\!]^{M,g^{\mathbf{u}/v}}$ (this being an object of type b).
1.2.2. If α is an expression of type a and β is an expression of type $\langle a, b\rangle$, then $\beta(\alpha)$ is an expression of type b. Specifically,	2.2.2.
(i) If v is a variable of type a, α is an expression of type a, and ϕ is an expression of type b, then $\lambda v[\phi](\alpha)$ is an expression of type b;	(i) If v is a variable of type a, α is an expression of type a, $[\![\alpha]\!]^{M,g} = \mathbf{u}$, and ϕ is an expression of type b, then $[\![\lambda v[\phi](\alpha)]\!]^{M,g} = [\![\phi]\!]^{M,g^{\mathbf{u}/v}}$;
(ii) If P is an n-place predicate constant or predicate variable and each of $a_1, \ldots, a_n$ is an individual constant or individual variable (where $1 \leq n$), then $P(a_1, \ldots, a_n)$ is a formula.	(ii) If P is an n-place predicate constant or predicate variable and each of $a_1, \ldots, a_n$ is an individual constant or individual variable, then $[\![P(a_1, \ldots, a_n)]\!]^{M,g} = \mathbf{1}$ iff $\langle [\![a_1]\!]^{M,g}, \ldots, [\![a_n]\!]^{M,g}\rangle \in [\![P]\!]^{M,g}$

Table 2.1: The syntax and semantics of an extended logic, cont'd

Syntax	**Semantics**
Formation rules, cont'd	
1.2.3. If v is an individual variable, and ϕ is a formula, then	2.2.3. If v is an individual variable, and ϕ is a formula, then
(i) $\forall v(\phi)$ is a formula;	(i) $[\![\forall v(\phi)]\!]^{M,g} = \mathbf{1}$ iff for all $\mathbf{u} \in \mathcal{U}$, $[\![\phi]\!]^{M,g^{\mathbf{u}/v}} = \mathbf{1}$;
(ii) $\exists v(\phi)$ is a formula.	(ii) $[\![\exists v(\phi)]\!]^{M,g} = \mathbf{1}$ iff there is at least one $\mathbf{u} \in \mathcal{U}$ such that $[\![\phi]\!]^{M,g^{\mathbf{u}/v}} = \mathbf{1}$.
1.2.4. If ϕ and ψ are formulae, then	2.2.4. If ϕ and ψ are formulae, then
(i) $\neg(\phi)$ is a formula;	(i) $[\![\neg(\phi)]\!]^{M,g} = \mathbf{1}$ iff $[\![\phi]\!]^{M,g} = \mathbf{0}$;
(ii) $(\phi \wedge \psi)$ is a formula;	(ii) $[\![(\phi \wedge \psi)]\!]^{M,g} = \mathbf{1}$ iff $[\![\phi]\!]^{M,g} = \mathbf{1}$ and $[\![\psi]\!]^{M,g} = \mathbf{1}$;
(iii) $(\phi \vee \psi)$ is a formula;	(iii) $[\![(\phi \vee \psi)]\!]^{M,g} = \mathbf{1}$ iff $[\![\phi]\!]^{M,g} = \mathbf{1}$ or $[\![\psi]\!]^{M,g} = \mathbf{1}$ or both $= \mathbf{1}$;
(iv) $(\phi \supset \psi)$ is a formula;	(iv) $[\![(\phi \supset \psi)]\!]^{M,g} = \mathbf{1}$ iff $[\![\phi]\!]^{M,g} = \mathbf{0}$ or $[\![\psi]\!]^{M,g} = \mathbf{1}$;
(v) $(\phi \leftrightarrow \psi)$ is a formula.	(v) $[\![(\phi \leftrightarrow \psi)]\!]^{M,g} = \mathbf{1}$ iff $[\![\phi]\!]^{M,g} = [\![\psi]\!]^{M,g}$.
1.2.5. Nothing else is a formula.	

Table 2.1: The syntax and semantics of an extended logic, cont'd

ϕ; $(\phi \wedge \psi)$ is a **conjunction**, ϕ and ψ being the **conjuncts**; $(\phi \vee \psi)$ is a **disjunction**, ϕ and ψ being the **disjuncts**; $(\phi \supset \psi)$ is an **implication**, ϕ being the **antecedent** and ψ being the **consequent**; and, $(\phi \leftrightarrow \phi)$ is a **bi-implication**.[3]

The definitions of basic expressions and the formation rules license expressions of the following kinds as formulae. (We take liberties over parentheses where we can do so without ambiguity, e.g. we write $\neg died1(d)$ rather than $\neg(died1(d))$.)[4]

$died1(d)$	$\neg died1(d)$	$\forall x\ died1(x)$
$died1(m)$	$\lambda x[died1(x)](d)$	$\exists y\ killed(y, m)$
$killed1(d, m)$	$died1(m) \vee died1(d)$	$\forall y \forall x(killed1(x, y) \supset died1(y))$
$killed1(m, m)$	$killed1(d, m) \supset died1(m)$	$\forall y(\exists x(killed1(x, y)) \supset died1(y))$

Because our revised logic contains variables, it was necessary to add to the semantics some apparatus for assigning semantic values to variables. We avoided doing this in the previous chapter, finessing their treatment and that of the semantics of lambda expressions for the sake of clarity. Now that we have variables appearing in both lambda expressions and in quantified expressions, the informal semantics is no longer tenable. Their proper treatment is accomplished by an **assignment of values to variables** (or **value assignment**), usually written as a function g. This simply assigns to a variable a semantic value of the appropriate type from the universe of discourse of the model: if the variable is an individual variable, then g assigns to it an individual in $\mathcal{U}$; if the variable is of type $\langle \mathbf{e}, \mathbf{t} \rangle$, then g assigns to it a function from individuals from $\mathcal{U}$ to truth-values, and so on. Semantic values are now computed with respect both to M and to g, so that the semantic value of an expression α will be written $[\![\alpha]\!]^{M,g}$, read 'the semantic value of α with respect to the model M and the value assignment g'. As can be seen in table 2.1, this makes no substantive difference to the way we compute the semantic values of most expressions. It affects only expressions that may contain variables. In the semantic rules for quantified expressions and the revised semantic rules for lambda expressions, we are now asked to obtain the semantic value of an expression ϕ with respect to the model M and an assignment function $g^{\mathbf{u}/v}$. This is an assignment function just like g except that it assigns the object $\mathbf{u}$ from $\mathcal{U}$ to the variable v. This is the most common way that the value assignment is used: if you need to get a simpler grip on the semantic rules in the table, you can just think of $[\![\phi]\!]^{M,g^{\mathbf{u}/v}}$ as the semantic value of the expression ϕ taking any occurrences of variable v in ϕ to denote $\mathbf{u}$.

To illustrate the definitions in the table, we use the same basic expressions as we did in the previous chapter, and we define a universe and an interpretation function as here:

$$\mathcal{U} = \{\mathbf{i_1}, \mathbf{i_2}\}$$

[3]Some readers may be more used to seeing the symbols $\rightarrow$ or $\Rightarrow$ where we have used the symbol $\supset$, i.e. $(\phi \rightarrow \psi)$ or $(\phi \Rightarrow \psi)$ in place of $(\phi \supset \psi)$. None of these symbols is ideal: $\rightarrow$ is also used in grammar rules; $\Rightarrow$ is also used in showing derivations; $\supset$ is also used in set theory. Since there is less set theory than grammar in this book, we have chosen to use $\supset$ in our logic, in the hope that this reduces the risk of notation confusion. Similarly, some readers may be more used to the symbol $\equiv$ where we have used $\leftrightarrow$. We have already been using the symbol $\equiv$ as part of our meta-language, when we show equivalence of two formulae, and we continue to reserve it for that use.

[4]To allow omission of parentheses without ambiguity, we should properly give a definition of the **precedence** and **associativity** of the operators. For the simple formulae used in this book, a pragmatic use of parentheses allows us to ignore this.

$$\begin{array}{lcl} \mathcal{I}(d) & = & \mathbf{i_1} \\ \mathcal{I}(m) & = & \mathbf{i_2} \end{array}$$

$$\begin{array}{lcl} \mathcal{I}(died1) & = & \{\mathbf{i_1}\} \\ \mathcal{I}(killed1) & = & \{\langle \mathbf{i_2}, \mathbf{i_1} \rangle\} \end{array}$$

We also need a value assignment, g. For our purposes, this can be pretty much an arbitrary assignment of objects to variables. We will use just the *individual* variables x and y, to which g will assign individuals from the universe; to emphasise the arbitrariness we will define g as mapping both (individual) variables to the same object:

$$\begin{array}{lcl} g(x) & = & \mathbf{i_2} \\ g(y) & = & \mathbf{i_2} \end{array}$$

$\forall x\, died1(x)$ is a well-formed formula of the logic according to the syntax rules given in the table. The corresponding semantic rule (2.2.3 (i)) requires a consideration of all objects in $\mathcal{U}$ ($\mathbf{i_1}$ and $\mathbf{i_2}$):

- For $\mathbf{i_1}$, we must evaluate $[\![died1(x)]\!]^{M,g^{\mathbf{i_1}/x}}$, i.e. using the value assignment, $g^{\mathbf{i_1}/x}$, that maps x to $\mathbf{i_1}$. This is true ($= \mathbf{1}$) if and only if $[\![x]\!]^{M,g^{\mathbf{i_1}/x}} \in [\![died1]\!]^{M,g^{\mathbf{i_1}/x}}$. $[\![x]\!]^{M,g^{\mathbf{i_1}/x}} = g^{\mathbf{i_1}/x} = \mathbf{i_1}$; $[\![died1]\!]^{M,g^{\mathbf{i_1}/x}} = \mathcal{I}(died1) = \{\mathbf{i_1}\}$; and, since $\mathbf{i_1} \in \{\mathbf{i_1}\}$, $[\![died1(x)]\!]^{M,g^{\mathbf{i_1}/x}}$ is true ($= \mathbf{1}$).
- For $\mathbf{i_2}$, we must evaluate $[\![died1(x)]\!]^{M,g^{\mathbf{i_2}/x}}$. $[\![x]\!]^{M,g^{\mathbf{i_2}/x}} = g^{\mathbf{i_2}/x} = \mathbf{i_2}$; $[\![died1]\!]^{M,g^{\mathbf{i_2}/x}} = \mathcal{I}(died1) = \{\mathbf{i_1}\}$; but, since $\mathbf{i_2} \notin \{\mathbf{i_1}\}$, $[\![died1(x)]\!]^{M,g^{\mathbf{i_2}/x}}$ is false ($= \mathbf{0}$).

Since the body of the quantified expression was not true for all assignments of objects in $\mathcal{U}$ to the variable x, $[\![\forall x\, died1(x)]\!]^{M,g}$ is false ($= \mathbf{0}$).

We leave the reader to try other examples.

Exercise 1 *With respect to the same model and value assignment, compute the following semantic values:*

1. $[\![died1(m) \vee died1(d)]\!]^{M,g}$
2. $[\![\forall y \forall x(killed1(x,y) \supset died1(y))]\!]^{M,g}$
3. $[\![\forall y(\exists x(killed1(x,y)) \supset died1(y))]\!]^{M,g}$
4. $[\![\lambda x[died1(x)](d)]\!]^{M,g}$

With the two quantifiers that we have introduced we can manage translations of only certain English determiners, most naturally *"every"* and *"some"*. *"no"* is straightforward too, and some others can also be defined. But we concentrate on *"every"*, *"some"* and *"no"* in the rest of this chapter, with only a small amount of material that considers other determiners.

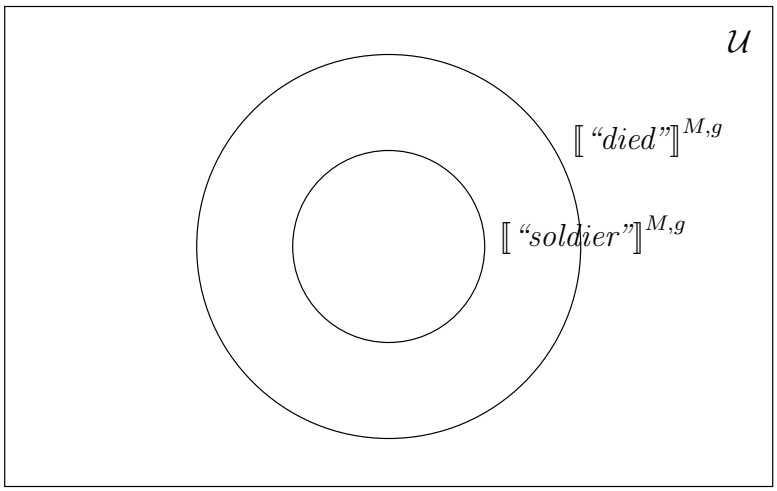

Figure 2.1: $[\![$ *"soldier"* $]\!]^{M,g} \subseteq [\![$ *"died"* $]\!]^{M,g}$

2.2 Quantified noun phrase translations

In the previous chapter we took the denotations of NPs to be individuals, which worked well for the simple names we were considering there. But, it is highly implausible to think that quantified NPs denote individuals. It seems more plausible for them to denote *sets*. Suppose quantified NPs denoted sets of individuals. In this case, the NP *"every soldier"* would presumably denote the set of soldiers. But this raises problems. For example, we would then need a denotation for, e.g., *"no soldier"*, and this is unlikely also to denote the set of soldiers. If, instead, it were to denote the empty set, then so, presumably, would the NP *"no witch"*; the result would be that $[\![$ *"no soldier died"* $]\!]^{M,g} = [\![$ *"no witch died"* $]\!]^{M,g}$: they would have identical semantics, which is not correct. This and other problems require that we abandon the idea that quantified NPs denote sets of individuals. We are going to claim that the denotation of a quantified NP is not a set of individuals, but a *set of sets of individuals*. We argue the case for this analysis in what follows.

Rather than beginning with the semantic values of the quantified NPs themselves, we will begin by informally characterising the denotations of simple *sentences* that contain such NPs; specifically, we will consider sentences (1), (2) and (3) from above (page 27):

- *"Every soldier died"* is true if and only if the set of soldiers is a subset of the set of those who died, i.e. $[\![$ *"soldier"* $]\!]^{M,g} \subseteq [\![$ *"died"* $]\!]^{M,g}$ (as shown in figure 2.1).

- *"No witch died"* is true if and only if the intersection of the set of witches with the set of those who died is empty, i.e. $[\![$ *"witch"* $]\!]^{M,g} \cap [\![$ *"died"* $]\!]^{M,g} = \emptyset$ (as shown in figure 2.2).

- *"Some king died"* is true if and only if the intersection of the set of kings with

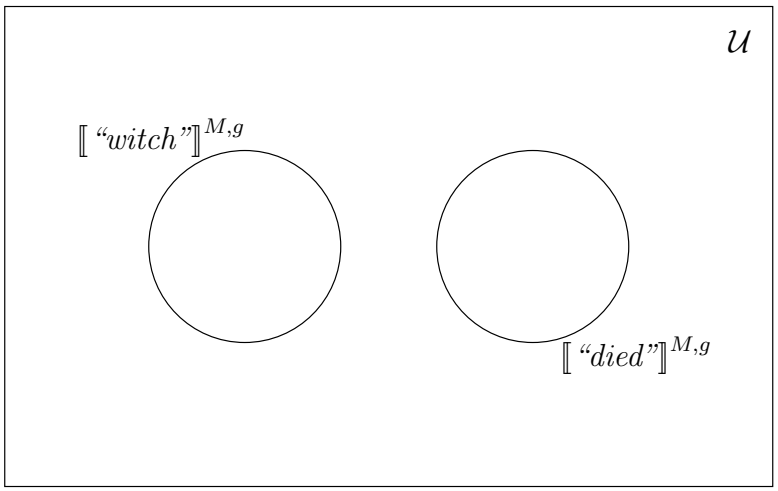

Figure 2.2: $[\![$ *"witch"* $]\!]^{M,g} \cap [\![$ *"died"* $]\!]^{M,g} = \emptyset$

the set of those who died is *not* empty, i.e. $[\![$ *"king"* $]\!]^{M,g} \cap [\![$ *"died"* $]\!]^{M,g} \neq \emptyset$ (as shown in figure 2.3, taking the overlap to be non-empty).

If these are the informal truth-conditions, we can now pair the sentences with formulae of logic that receive exactly the same truth-conditions as those depicted above (assuming that the predicate constants $soldier1$, $witch1$, $king1$ and $died1$ denote the same sets as the English words *"soldier"*, *"witch"*, *"king"* and *"died"* respectively):

English	Logic
"Every soldier died"	$\forall x(soldier1(x) \supset died1(x))$
"No witch died"	$\neg\exists x(witch1(x) \wedge died1(x))$
"Some king died"	$\exists x(king1(x) \wedge died1(x))$

Note the different quantifiers used in the translations ($\forall, \exists$), but also note that, to get the right truth-conditions, the connectives used ($\supset$ and $\wedge$) vary from translation to translation.[5] We leave it to the reader to verify that the formal semantics given in table 2.1 will indeed have the effect of making these logic formulae true when the corresponding English sentences are true.

We must set up a systematic correspondence between the sentences and the logic formulae, and we must do this in a way that meets our constraint of compositionality. Every constituent of the sentences must have a well-formed expression of logic associated with it, and these logic expressions must combine, using function application, in a way determined by the phrase-structure of the sentences. In particular, this requires that we say what the logical translation of the NP is and how this combines with the translation of the VP.

[5]There are (infinitely) many formulae with equivalent truth-conditions that could be used in place of the formulae given here. In particular, we note that *"no witch died"* could just as well have been translated as $\forall x(witch1(x) \supset \neg died1(x))$.

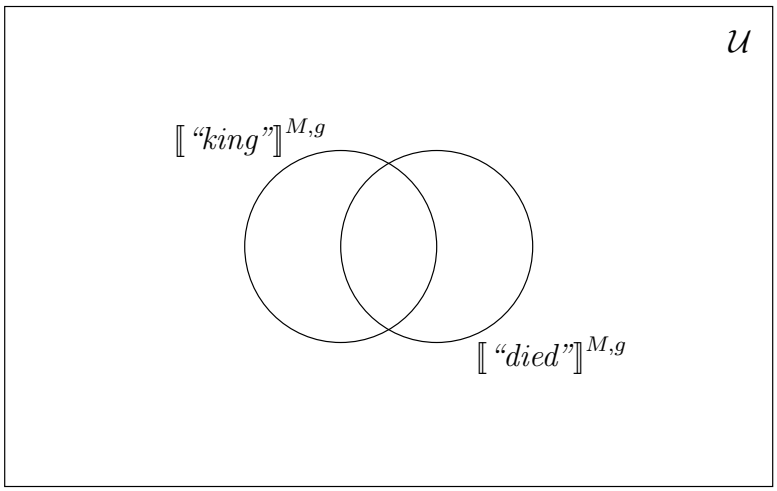

Figure 2.3: $[\![$ *"king"* $]\!]^{M,g} \cap [\![$ *"died"* $]\!]^{M,g} \neq \emptyset$

But the formulae given above present an apparent problem for compositionality, since there are no well-formed expressions in the logical translations that correspond to the English NPs *"every soldier"*, *"no witch"*, and *"some king"*. For example, the contribution of *"every soldier"* to the formula $\forall x(soldier1(x) \supset died1(x))$ cannot be either of the following:

$$\forall x(soldier1(x)$$
$$\forall x(soldier1(x) \supset$$

as these are not well-formed subexpressions within the formula. This problem has a solution, first presented by the American philosopher and logician Richard Montague.

To proceed, we will make an assumption about the translations of VPs: we will assume (for now) that these are unchanged from what they were in chapter 1. (In some extensions of this work, this assumption is, in fact, abandoned — see, in particular, subsection 2.4.4.) In chapter 1, the type of VPs was $\langle \mathbf{e}, \mathbf{t} \rangle$, a (characteristic function of a) set of individuals, and a translation of the VPs in the sentences above might be $\lambda x[died1(x)]$.

In chapter 1, the logic translation rule that corresponded to the syntax rule $\mathsf{S} \rightarrow \mathsf{NP\ VP}$ was $\mathsf{S}' = \mathsf{VP}'(\mathsf{NP}')$, i.e. the translation of the VP is applied to that of the NP. But we cannot retain this translation rule. The type of VP is a function from individuals, but we have decided that quantified NPs do not denote individuals; they denote sets of some kind; the NP translations can therefore not be the argument of the VP translations.

There are several ways of repairing this. We could, for example, abandon the assumption that VP translations stay the same as they were in the previous chapter. But for now we will take the option of revising the semantic translation rule: if the VP translation cannot be the functor, then the functor must be the NP translation:

- English syntax: $\mathsf{S} \rightarrow \mathsf{NP\ VP}$; Logic translation: $\mathsf{S}' = \mathsf{NP}'(\mathsf{VP}')$

We can now find out the logical translation of quantified NPs. We will take *"Every soldier died"* as our example. Its translation is $\forall x(soldier1(x) \supset died1(x))$. We can split this into a functor and its argument by lambda abstraction over the argument. Since we have just decided that the NP translation is the functor, we must abstract over the contribution made to the semantic translation by the VP. The contribution that the VP *"died"* makes to the translation is presumably the predicate $died1$. If we lambda abstract over this predicate, we get the following:

$$\lambda P[\forall x(soldier1(x) \supset P(x))](died1)$$

This might look a bit strange at first. Up to now, when we have formed a lambda abstract, we have abstracted over an individual constant symbol, e.g.:

$$died1(d) \equiv \lambda x[died1(x)](d)$$

But there is surely nothing to stop us from lambda abstracting over predicate symbols:

$$died1(d) \equiv \lambda P[P(d)](died1)$$

or, as in the example:

$$\forall x(soldier1(x) \supset died1(x)) \equiv \lambda P[\forall x(soldier1(x) \supset P(x))](died1)$$

These expressions have the same truth-conditions.

A further worry that you might have is that we have said in the above that the contribution made by the verb phrase is the predicate $died1$, whereas we have previously taken VP translations to be lambda expressions, e.g. $\lambda x[died1(x)]$. $died1$ and $\lambda x[died1(x)]$ have the same semantic type (both $\langle \mathbf{e}, \mathbf{t} \rangle$) and so, in fact, we can lambda abstract over $\lambda x[died1(x)]$ in $\forall x(soldier1(x) \supset died1(x))$ as follows:

$$\forall x(soldier1(x) \supset died1(x)) \equiv \lambda P[\forall x(soldier1(x) \supset P(x))](\lambda y[died1(y)])$$

(We have used the variable y in the argument expression to make expressions with multiple variables more readable.) This is still equivalent to the original formula, as demonstrated by the following *two* lambda conversions:

$$\begin{aligned} \lambda P[\forall x(soldier1(x) \supset P(x))](\lambda y[died1(y)]) &\equiv \forall x(soldier1(x) \supset \lambda y[died1(y)](x)) \\ &\equiv \forall x(soldier1(x) \supset died1(x)) \end{aligned}$$

In the first lambda conversion, we replace P in the body of the leftmost lambda expression by $\lambda y[died1(y)]$; in the second conversion, we replace y in the body of the remaining lambda expression by x; and this gives us the original formula. We will, in fact, use the second of the lambda abstractions $(\lambda P[\forall x(soldier1(x) \supset P(x))](\lambda y[died1(y)]))$ henceforth.

You will note that our lambda abstract has a variable in place of a predicate symbol. (Following our convention, we used a variable name P, i.e. an upper case letter from the middle of the alphabet, rather than a variable name such as x which we have used for individual variables.) It follows that we are now using **Higher-Order Logic**. Much of this chapter is going to be taken up with higher-order expressions

like these. In general, there are computational difficulties in computing with higher-order logics. However, although the translations of certain constituents such as NPs will be higher-order, we are going to arrange things so that the translation of Ss will always be first-order. In other words, by the time we have combined the possibly higher-order translations of the constituents of an S in appropriate ways and lambda converted as much as possible, we will always arrive at a first-order translation for the S. Since S is the category whose semantics concern us most, we will not need to be too worried about the higher-order translations of other constituents: we will not be doing any significant computation with them; we will only be functionally applying them to other expressions and lambda converting the result.

Now, if $\lambda P[\forall x(soldier1(x) \supset P(x))](\lambda y[died1(y)])$ is the result of lambda abstracting over the VP contribution, then, by dropping the argument $\lambda y[died1(y)]$, we are left with the logical translation of the NP *"every soldier"*:

$$\lambda P[\forall x(soldier1(x) \supset P(x))]$$

By carrying out this reasoning on all the NPs that we have been looking at, we get the following equivalences:

English	Logic
"every soldier"	$\lambda P[\forall x(soldier1(x) \supset P(x))]$
"no witch"	$\lambda P[\neg\exists x(witch1(x) \wedge P(x))]$
"some king"	$\lambda P[\exists x(king1(x) \wedge P(x))]$

Having arrived at these logical translations for the NPs, we can now be more precise about quantified NP denotations by saying what the logical translations denote. We will start by considering their semantic type.

It is easy to show that quantified NPs denote *sets of sets of individuals*. We can show this two (equivalent) ways. Firstly, we revised the semantic translation rule for the syntax rule $\mathsf{S} \rightarrow \mathsf{NP\ VP}$ to $\mathsf{S'} = \mathsf{NP'(VP')}$. The NP translation is a function from VP translations (type $\langle \mathbf{e}, \mathbf{t} \rangle$) to S translations (type $\mathbf{t}$), hence it is of type $\langle \langle \mathbf{e}, \mathbf{t} \rangle, \mathbf{t} \rangle$. Secondly, we can reach the same result with reference to the logic. We started with $\forall x(soldier1(x) \supset died1(x))$, which was of type $\mathbf{t}$. We abstracted over $\lambda y[died1(y)]$, which is of type $\langle \mathbf{e}, \mathbf{t} \rangle$. After dropping the argument $\lambda y[died1(y)]$, we got $\lambda P[\forall x(soldier1(x) \supset P(x))]$, which denotes a function from the thing we abstracted over, type $\langle \mathbf{e}, \mathbf{t} \rangle$, to what we had to begin with, type $\mathbf{t}$, in other words it has type $\langle \langle \mathbf{e}, \mathbf{t} \rangle, \mathbf{t} \rangle$. The type $\langle \langle \mathbf{e}, \mathbf{t} \rangle, \mathbf{t} \rangle$ is, informally, a set of sets of individuals: it is a function from sets (type $\langle \mathbf{e}, \mathbf{t} \rangle$ — the characteristic function of a set of individuals) to truth values (type $\mathbf{t}$) and hence is a characteristic function of sets (type $\langle \langle \mathbf{e}, \mathbf{t} \rangle, \mathbf{t} \rangle$).

Knowing the type is useful, but we would like also to say what particular sets of sets of individuals the noun phrases (and their logical translations) will denote in a model. Let us consider this for the NP *"every soldier"*.

We know the following:

- *"Every soldier died"* is true if and only if the set of soldiers is a subset of the set of those who died.
- *"Every soldier slept"* is true if and only if the set of soldiers is a subset of the set of those who slept.

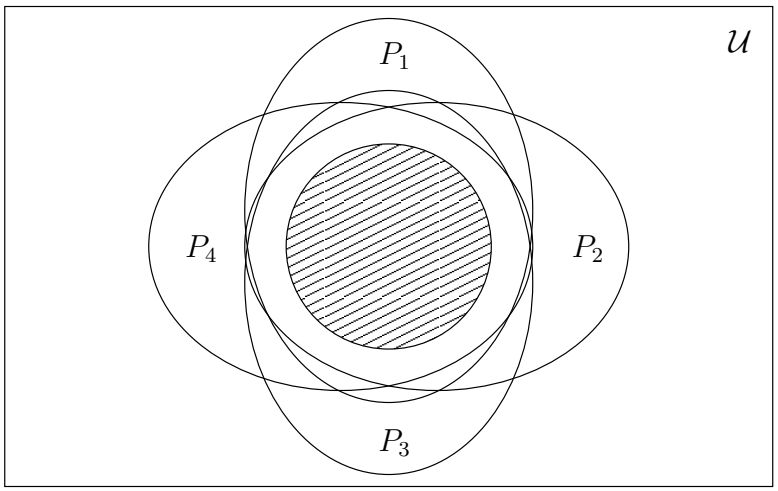

Figure 2.4: $[\![$ *"every soldier"* $]\!]^{M,g}$

- *"Every soldier dreamt"* is true if and only if the set of soldiers is a subset of the set of those who dreamt.

In general, *"Every soldier Ps"* is true if and only if the set of soldiers is a subset of the set of those who P. The pattern is that the semantic contribution of *"every soldier"* depends only on the sets of which the set of soldiers is a subset. Then, supplied with a particular set, P (the denotation of some VP), we can determine whether the set of soldiers is a subset of P; if it is, then the sentence *"Every soldier Ps"* is true. In figure 2.4, the set of soldiers is the shaded area. The denotation of *"every soldier"* is the set $\{P_1, P_2, P_3, P_4\}$, the sets of which the set of soldiers is a subset.[6]

Similarly, *"No witch Ps"* is true if and only if the set of witches has a null intersection with the set of those who P. In other words, *"no witch"* also denotes a set of sets: the set of sets whose intersection with the set of witches is empty.[7] This is shown in figure 2.5; the shaded set is the set of witches; the denotation of *"no witch"* is the set $\{P_1, P_2, P_3, P_4, P_5, P_6\}$.

Similarly too, *"Some king Ps"* is true if and only if the set of kings has a non-null intersection with the set of those who P. In other words, *"some king"* denotes a set of sets: the set of sets whose intersection with the set of kings is not empty.[8] Figure 2.6 shows this; the shaded set is the set of kings; the denotation of *"some king"* is the set $\{P_1, P_2, P_3, P_4, P_5\}$.

We have successfully provided an intuitively appealing account of the denotations of quantified NPs (and their logical translations), the denotations being of the semantic type that we deduced earlier.

[6]The set of sets of which some other set is a subset is said to be the **universal sublimation** of that set.

[7]The set of sets which have a null intersection with some other set is said to be the **negative sublimation** of that set.

[8]The set of sets which have a non-null intersection with some other set is said to be the **existential sublimation** of that set.

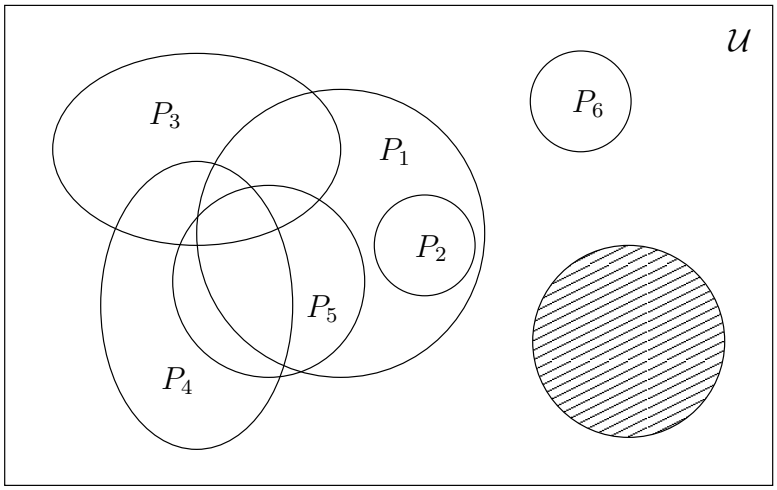

Figure 2.5: $[\![$ *"no witch"* $]\!]^{M,g}$

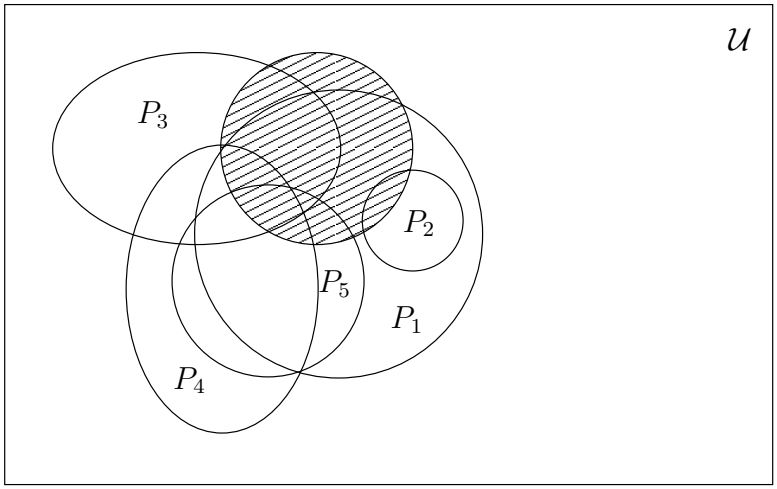

Figure 2.6: $[\![$ *"some king"* $]\!]^{M,g}$

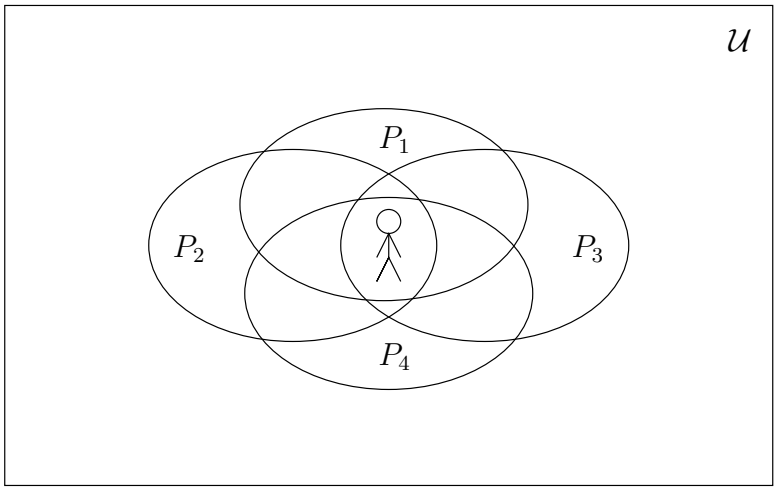

Figure 2.7: $[\![$ *"Duncan"* $]\!]^{M,g}$

There remains a very important question: what is the appropriate translation for those NPs for which an individual constant seemed initially to be an appropriate translation (i.e. names like *"Duncan"*)? It would be unfortunate if NPs were to have one type when they are simple names and another type when they are made up of a determiner and a noun. Ideally, NPs will always have the same type. (If they have different types, we may need multiple semantic rules to correspond to the syntax rule S $\rightarrow$ NP VP, which would also be unfortunate.) The ingenious thing about Montague's approach is that it extends to these NPs too, without modification. If we take the translation of a sentence, such as *"Duncan died"*, and abstract over the VP contribution, we get the following result:

$$died1(d) \equiv \lambda P[P(d)](\lambda y[died1(y)])$$

Removing $\lambda y[died1(y)]$, as before, leaves $\lambda P[P(d)]$ as the translation of the NP *"Duncan"*. This has the same type as *"every soldier"* and other quantified NPs, namely a set of sets. In this case though, the denotation is the set of sets of which Duncan is a member.[9] This makes intuitive sense too (surprisingly) as we illustrate with the following *informal* argument. If you want to identify an individual to someone, how do you do it? You give enough properties of the individual (i.e. enough sets of which the individual is a member) until you have distinguished that individual from all others. This illustrates that even NPs that are names denote sets of sets of individuals. (The argument is informal and possibly misleading for the following reason: to identify an individual some contextually-determined, probably minimal, number of properties is needed; but the denotation of a name is the set of *all* sets of which the individual is a

member.) Figure 2.7 shows this; the stick figure in the centre region should be taken to represent the individual, Duncan; the denotation of the NP *"Duncan"* is then the set of sets of which Duncan is a member, i.e. $\{P_1, P_2, P_3, P_4\}$.

2.3 Determiner and noun translations

2.3.1 Simple determiners

The quantified NPs we have been looking at obviously have some constituent structure to them. Specifically, grammar 2.1 shows them as comprising a determiner and a nominal constituent: NP $\rightarrow$ Det N. In this section, we consider the semantic type, denotation and translation of determiners and nouns.

The logical translation of the NP *"every soldier"*, we have shown, is $\lambda P[\forall x(soldier1(x) \supset P(x))]$. We need to split this into the contribution made by the determiner and that made by the noun, combined through function application. The noun *"soldier"* presumably denotes the set of soldiers; this being a set of individuals, it will be of type $\langle \mathbf{e}, \mathbf{t} \rangle$ and its translation will therefore be, e.g., $\lambda y[soldier1(y)]$. We can lambda abstract over this in $\lambda P[\forall x(soldier1(x) \supset P(x))]$, thus abstracting away the contribution made by the N, to give:

$$\lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]](\lambda y[soldier1(y)])$$

which has the same truth-conditions as $\lambda P[\forall x(soldier1(x) \supset P(x))]$. (You can verify this by carrying out *two* lambda conversions on the above to get back to the original.) In this lambda abstract, since the argument $\lambda y[soldier1(y)]$ is the translation of *"soldier"*, the functor must be the translation of the determiner *"every"*. By this approach, we get the translations of all the determiners:

English	Logic
"every"	$\lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]]$
"no"	$\lambda Q[\lambda P[\neg\exists x(Q(x) \wedge P(x))]]$
"some"	$\lambda Q[\lambda P[\exists x(Q(x) \wedge P(x))]]$

Determiners, therefore, denote functions from sets (the semantic values of Ns) to sets of sets (the semantic values of NPs), i.e. functions from sets of individuals to sets of sets of individuals, $\langle\langle \mathbf{e}, \mathbf{t} \rangle, \langle\langle \mathbf{e}, \mathbf{t} \rangle, \mathbf{t} \rangle\rangle$. Recall that $\langle a, \langle a, \mathbf{t} \rangle\rangle$ is the semantic type for relations between two things of type a. Taking a to be $\langle \mathbf{e}, \mathbf{t} \rangle$, we see that determiners are relations between two sets of individuals. This seems sensible, as we can demonstrate schematically: if Q and P are these sets, then *"every Q Ps"* is true if and only if Q is a subset of P; *"no Q Ps"* is true if and only if Q and P have a null intersection; and *"some Q Ps"* is true if and only if Q and P have a non-null intersection.

A sample lexicon (for just one of the determiners and one of the nouns, with the others being defined analogously) now looks like this:

[9] The set of sets of which some individual is a member is said to be the **individual sublimation** of that individual.

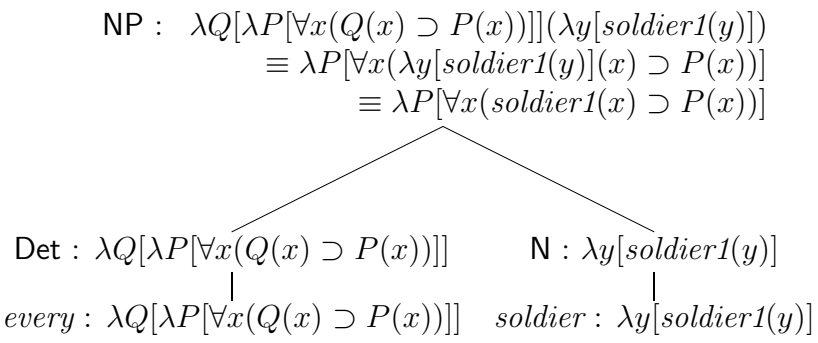

Figure 2.8: Annotated tree for the NP *"every soldier"*

- English syntax: *every* : Det; Logic translation: $\mathsf{Det}' = every' = \lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]]$
- English syntax: *soldier* : N; Logic translation: $\mathsf{N}' = soldier' = \lambda y[soldier1(y)]$

The grammar rule's corresponding translation rule is:

- English syntax: NP → Det N; Logic translation: $\mathsf{NP}' = \mathsf{Det}'(\mathsf{N}')$

An annotated tree for the NP *"every soldier"* is given as figure 2.8.

Exercise 2 *In the formulation used above we have simplified by using the rule* NP → Det N. *This simplification is easily remedied:*

- *English syntax:* NP → Det Nbar*;* *Logic translation:* $\mathsf{NP}' = \mathsf{Det}'(\mathsf{Nbar}')$
- *English syntax:* Nbar → N*;* *Logic translation:* $\mathsf{Nbar}' = \mathsf{N}'$

An Nbar *will then have* $\langle \mathbf{e}, \mathbf{t} \rangle$ *as its semantic type.*

Now extend this revised grammar by adding the logic translations for the following lexical entry and grammar rule:

brave : Adj

Nbar → Adj Nbar

When working out your answer, do not change any of the types or translation rules of anything else.

What simplifications are inherent in this treatment of adjectives?

2.3.2 Generalised quantifier theory

The logic we have presented in table 2.1 provides suitable target translations for only a handful of English determiners: in addition to *"every"*, *"some"* and *"no"*, we can translate, e.g., *"two"*:[10,11]

English	Logic
"Two kings died"	$\exists x_1 \exists x_2 (king1(x_1) \wedge king1(x_2) \wedge x_1 \neq x_2 \wedge died1(x_1) \wedge died1(x_2))$

The logic is paraphrasable as: there exists an x_1 who is a king and an x_2 who is a king, where x_1 is not the same as x_2; x_1 died and x_2 died. (In fact, strictly, the logic translates the English determiner phrase *"at least two"*.) There are two problems: (i) this translation, and even the translations of *"every"*, *"some"* and *"no"* given in the previous subsection, introduce connectives (the uses of $\supset$ and $\wedge$) and other logical expressions (e.g. the use of $x_1 \neq x_2$ in the above) that have no syntactic correlate in the original English sentences they purport to translate; and (ii) the treatment does not extend to determiners and determiner phrases such as *"most"*, *"few"*, *"(exactly) two"*, *"at least half"*, etc. These problems can be remedied through use of an alternative target logic for the translation. The resulting approach is referred to as **generalised quantifier theory**.

In a logic extended with generalised quantifiers, quantifiers such as $every1$, $some1$, $exactlytwo1$, $most1$, $few1$, etc. are basic expressions of type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle\rangle$ in the logic. By the normal formation rule, i.e. that if β is an expression of type $\langle a, b\rangle$ and α is an expression of type a, then $\beta(\alpha)$ is an expression of type b, we can form formulae such as the following:

$$(every1(\lambda x[soldier1(x)]))(\lambda y[died1(y)])$$

or, more simply:

$$(every1(soldier1))(died1)$$

An NP translation would be, e.g.:

$$every1(\lambda x[soldier1(x)])$$

or, more simply:

$$every1(soldier1)$$

We then want to assign a semantic value to the quantifiers to give the same effect as our original translation. The definitions of $every1$ and $some1$ would be:

[10] The use of plural nouns in the examples given in this subsection introduce problems that are acknowledged only through the inclusion of this footnote and some pointers to the literature in the further reading section.

[11] We include words such as *"two"* in this subsection because their semantic treatment may be accounted for by generalised quantifier theory. However, readers should note that, syntactically, words like *"one"*, *"two"* and *"three"* are not determiners (their syntactic distribution is different from that of determiners such as *"every"*).

$$[\![every1]\!]^{M,g} = \mathcal{I}(every1) = \text{a function of type } \langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle\rangle \text{ which assigns to each set of individuals } Q \subseteq \mathcal{U} \text{ the set of sets } \{P \subseteq \mathcal{U} \mid Q \subseteq P\}$$

$$[\![some1]\!]^{M,g} = \mathcal{I}(some1) = \text{a function of type } \langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle\rangle \text{ which assigns to each set of individuals } Q \subseteq \mathcal{U} \text{ the set of sets } \{P \subseteq \mathcal{U} \mid Q \cap P \neq \emptyset\}$$

You can see that these denotations are effectively the same as the ones we described in subsection 2.3.1.

The denotations of *exactlytwo*1 and *most*1 might be:

$$[\![exactlytwo1]\!]^{M,g} = \mathcal{I}(exactlytwo1) = \text{a function of type } \langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle\rangle \text{ which assigns to each set of individuals } Q \subseteq \mathcal{U} \text{ the set of sets } \{P \subseteq \mathcal{U} \mid card(Q \cap P) = 2\}$$

$$[\![most1]\!]^{M,g} = \mathcal{I}(most1) = \text{a function of type } \langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle\rangle \text{ which assigns to each set of individuals } Q \subseteq \mathcal{U} \text{ the set of sets } \{P \subseteq \mathcal{U} \mid card(Q \cap P) > card(Q \cap (\mathcal{U} - P))\}$$

where $card(S)$ represents the cardinality (i.e. the number of elements in) the set S. You can see that for the formula $(exactlytwo1(soldier1))(died1)$ to be true, the intersection of the VP denotation, *died*1, (which is the set P above) and the nominal denotation, *soldier*1, (which is the set Q above) must have cardinality of two; for the formula $(most1(soldier1))(died1)$ to be true, the cardinality of those soldiers who died $(Q \cap P)$ must be greater than the cardinality of those soldiers who did not die $(Q \cap (\mathcal{U} - P))$. (Whether this is the correct denotation need not concern us; it suffices to illustrate the principle.)

We see that the introduction of generalised quantifiers solves the two problems mentioned earlier: (i) we no longer need have expressions in the logical translation that have no syntactic correlates in the original English expression: their contribution has been absorbed into the semantics of the generalised quantifier; and (ii) we now have a treatment of a wide range of English determiners and determiner phrases. In fact, further advantages follow too.

One advantage is that the theory allows the statement of a number of universals about natural language determiners.

To illustrate this, we note that if the cardinality of the universe of discourse $\mathcal{U}$ is n, then there are 2^{4^n} possible binary relations between $\mathcal{U}$'s subsets, and hence we have 2^{4^n} generalised quantifiers. However, only some of these are the translation of natural

language determiners. One empirical claim, for example, is that all natural language determiners translate as **conservative** quantifiers, where a quantifier is conservative if and only if for every set P and every set Q,

$$(\mathsf{Det}'(Q))(P) \equiv (\mathsf{Det}'(Q))(Q \wedge P)$$

For example, *"every"* (and its translation *every*1) is conservative: *"Every Q Ps"* is true if and only if *"Every Q is a Q who Ps"*. Informally, you would agree that *"Every soldier swears"* is true if and only if every soldier is a soldier who swears; *"Every soldier died"* is true if and only if every soldier is a soldier who died, and so on.[12] Similar arguments show that *"some"*, *"no"*, *"at least two"*, and so on are all translated as conservative quantifiers.

The property of being conservative is only one property that can be stated in generalised quantifier theory. Other properties can be defined and linguistic generalisations stated in terms of them too.

Another advantage of generalised quantifier theory is that it allows the introduction of some elements of context-dependence into theories of meaning. For example, the quantifier *many*1 might have as its denotation:

$$[\![many1]\!]^{M,g} = \mathcal{I}(many1) = \text{a function of type } \langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle\rangle \text{ which assigns to each set of individuals } Q \subseteq \mathcal{U} \text{ the set of sets } \{P \subseteq \mathcal{U} \mid card(Q \cap P) > \theta \times card(Q)\}$$

where θ is some contextually-dependent constant ($0 \leq \theta \leq 1$). In other words for $(many1(Q))(P)$ to be true, the number of Qs who P must be greater than some contextually determined proportion of the number of Qs that there are.[13]

Generalised quantifier theory is of increasing importance. However, having introduced it here, for the rest of this book we continue to restrict ourselves to the determiners *"every"*, *"some"* and *"no"* and the original logic translations we gave for these in subsection 2.3.1.

2.4 Object noun phrase translations

So far, discussion and exemplification have been confined to subject NPs. We now turn to direct object NPs, which require us first to discuss type-raising.[14]

[12] Formally, we ought to show that the equivalence follows from the model-theoretic definition of *every*1. We let an informal argument suffice here.

[13] This is not the only aspect of context-dependence that we ought to deal with when considering quantified NPs: even the nominal denotation may need to be contextually constrained. For example, for *"Every soldier died"* to be true, we do not usually require that absolutely every soldier died; we require only that every member of some contextually-relevant set of soldiers died. (Cf. *"Is everybody happy?"* This can be truthfully answered *"yes"*, e.g., if everybody to whom it is addressed is happy; it does not require a universal absence of human suffering.) But this aspect of context-dependence would not be reflected in the denotation of the quantifier, and we ignore it further here.

[14] The material in this section is easily extended to cover the complements of ditransitive verbs too. However, it is worth pointing out that other uses of quantified NPs will not necessarily receive

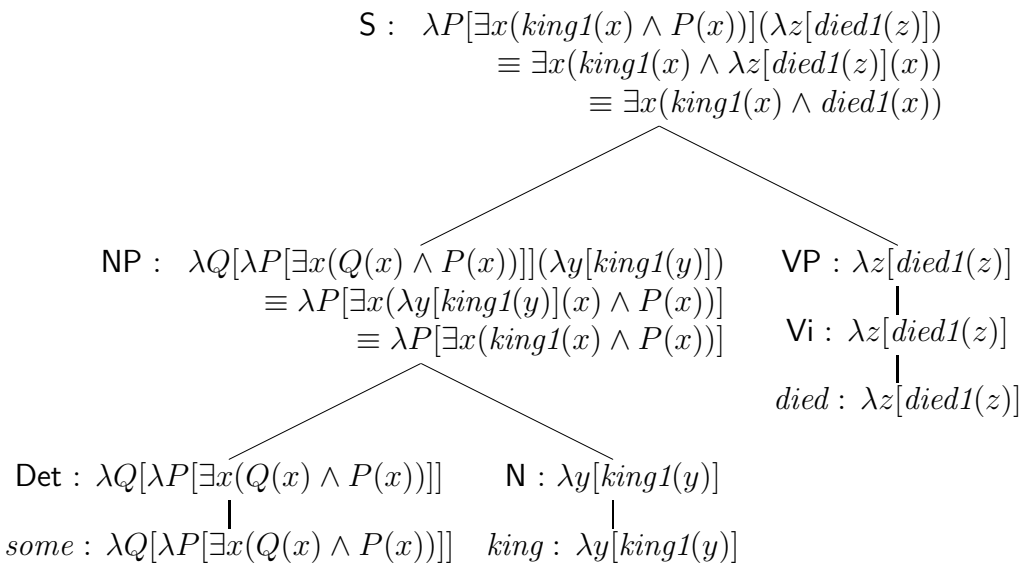

Figure 2.9: Annotated tree for *"Some king died"*

In our revised logic translation rule for the syntax rule S → NP VP, i.e. S′ = NP′(VP′), the NP translation, which in chapter 1 had been the argument, has become the functor:

- English syntax: S → NP VP; Logic translation: S′ = NP′(VP′)

An annotated tree for the S *"Some king died"* is given as figure 2.9. (Again we rename variables as necessary for clarity.) We started out in a situation in which NPs denoted individuals and VPs denoted functions from individuals to truth values. We have ended up with a situation in which NPs denote functions from VP denotations to truth values: the roles of functor and argument have been reversed. The denotation of the VP has remained the same, but the NP now has a functional type.

There is a general point to make about the technique involved in assigning these translations to NPs. This process (of making an argument expression into a functional expression which takes the former functor as argument) is known as **type-raising** (or **type-lifting**). The general rule is as follows. If we have an expression of type a and an expression of type $\langle a, b\rangle$, we can raise the type of the expression that was type a to type $\langle\langle a, b\rangle, b\rangle$: an expression which takes the original functor as argument, and returns the same value as the original functor did. We are going to use this idea again later in this section.

The syntax rule whose logical translation rule we want to develop is VP → Vt NP. The semantic type that we gave to Vt in the last chapter was $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$, NPs were of type $\mathbf{e}$ and the translation rule was VP′ = Vt′(NP′). But now, NPs are of

the same kind of treatment. For example, where an NP is predicative, as in the following sentence,

"Duncan is a king"

a different analysis may be needed.

type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$. Where previously the transitive verb translation was a function from NP translations to VP translations, this is no longer so. The rule $\mathsf{VP}' = \mathsf{Vt}'(\mathsf{NP}')$ does not work (indeed, it is no longer syntactically well-formed) because Vt' is a function from individuals, but NPs no longer denote individuals. And, the rule $\mathsf{VP}' = \mathsf{NP}'(\mathsf{Vt}')$ does not work either (and it too is not syntactically well-formed): an NP denotes a function from sets of individuals ($\langle \mathbf{e}, \mathbf{t}\rangle$), but Vts do not denote sets of individuals. Something must be changed to get this to work properly.

There is a range of solutions, each of which we will briefly consider. They vary according to whether the NP translation or the VP translation is the functor. They can also vary in the order of arguments needed in the transitive verb translation. From chapter 1 we have, e.g., *"killed"* translating as $\lambda y[\lambda x[killed1(x, y)]]$. The outermost abstracted variable (y) picked up the object NP translation and placed it into the second argument position, and the inner variable (x) was replaced by the subject NP translation and this was put into the first argument position. We may need to reverse this order in order to get things to work out.

2.4.1 Solution 1: 'Re-arrange' the translation of the transitive verb

In this solution, we have the NP as the functor and the Vt as the argument. The type of the object NP is $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$, a function whose argument is sets of individuals. For this to be the functor, the argument we supply must be of type $\langle \mathbf{e}, \mathbf{t}\rangle$. However, the type of the Vt is $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$, a function that returns sets of individuals. We therefore need to 'remove' one of the Vt's arguments to give an expression of type $\langle \mathbf{e}, \mathbf{t}\rangle$. The way we achieve this is by functionally applying the Vt translation (type $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$) to a fresh variable of type $\mathbf{e}$, not used elsewhere in the Vt or NP translations. If we use z for that variable, then the expression we want is $\mathsf{Vt}'(z)$, an expression of type $\langle \mathbf{e}, \mathbf{t}\rangle$. This expression is of the right type to be the argument to the NP translation (type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$), i.e. we want $\mathsf{NP}'(\mathsf{Vt}'(z))$, an expression of type $\mathbf{t}$. By making this expression (which will contain free occurrences of the variable z) the body of a lambda abstract whose abstracted variable is z, we arrive at an expression, $\lambda z[\mathsf{NP}'(\mathsf{Vt}'(z))]$, of type $\langle \mathbf{e}, \mathbf{t}\rangle$, the type of VPs, as desired. This solution is summarised, with some sample lexical entries, as grammar 2.2. (Note that the order of arguments in the translation of the transitive verb is the reverse of what it used to be. You should think about why this should be so.)

We can show how this works by computing the translation of *"killed Duncan"*:[15]

$$\begin{aligned}
\mathsf{VP}' &= \lambda z[\mathsf{NP}'(\mathsf{Vt}'(z))] \\
&= \lambda z[\lambda P[P(d)](\lambda y[\lambda x[killed1(y, x)]](z))] \\
&= \lambda z[\lambda P[P(d)](\lambda x[killed1(z, x)])] \\
&= \lambda z[\lambda x[killed1(z, x)](d)] \\
&= \lambda z[killed1(z, d)]
\end{aligned}$$

[15] It might seem strange to use *"killed Duncan"* as the example: *"Duncan"* is not a quantified NP, and yet the point of this section is to give ways of handling quantified NPs as transitive verb complements; *"killed every witch"* might be a more convincing example. What legitimises our choice of example is that quantified NPs and simple names have the same semantic type, $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$, and so, if the solution works on simple names, it will also work on quantified NPs. The advantage of using a simple name is that it makes the examples less cluttered.

Grammar	Logic
$\mathsf{S} \rightarrow \mathsf{NP\ VP}$	$\mathsf{NP}'(\mathsf{VP}')$
$\mathsf{VP} \rightarrow \mathsf{Vi}$	Vi'
$\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$	$\lambda z[\mathsf{NP}'(\mathsf{Vt}'(z))]$
$\mathsf{NP} \rightarrow \mathsf{Det\ N}$	$\mathsf{Det}'(\mathsf{N}')$
$\mathsf{NP} \rightarrow \mathsf{PName}$	PName'

Lexicon	Logic
every: Det	$\lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]]$
soldier: N	$\lambda x[soldier1(x)]$
Duncan: PName	$\lambda P[P(d)]$
died: Vi	$\lambda x[died1(x)]$
killed: Vt	$\lambda y[\lambda x[killed1(y, x)]]$

Grammar 2.2: 'Rearranging' the translation of the transitive verb

This is what we want. (You can see that it is of the same form as the VP translation in the annotated tree in figure 2.9.) Its type is $\langle \mathbf{e}, \mathbf{t} \rangle$, which is now suitable for use in the translation rule for $\mathsf{S} \rightarrow \mathsf{NP\ VP}$.

Exercise 3 *Use this solution to compute the translation of* "killed every witch"*. Then use your translation to compute the translation of* "Duncan killed every witch"*.*

The advantages of this approach are that it does not require any further type-raising, and that it gives a uniform role to NP translations: they act as the functor wherever they appear. The disadvantage is that the semantic rule corresponding to the $\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$ rule is now rather complicated: we have to break up the translation of the Vt in order to prevent the translation of the subject NP from being incorrectly bound. (The translation rule for the syntax rule $\mathsf{VP} \rightarrow \mathsf{Vd\ NP\ NP}$ is even more complicated as more 're-arranging' is needed.) Not only are the rules more complicated, we have also lost the uniform implementation of compositionality that we previously had: certain semantic rules now require an amount of algebraic manipulation in addition to function application of one child to the other.

2.4.2 Solution 2: Type-raise object noun phrases

Alternatively, we can raise the types of object NPs so that they are functions from Vt translations (type $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$) to VP translations (type $\langle \mathbf{e}, \mathbf{t} \rangle$), i.e. object NP types would be $\langle \langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$. This solution is summarised, with some sample lexical entries, as grammar 2.3. (Note that the order of arguments of the transitive verb is back to what it used to be in our original translation.)

We can show how this works by computing the translation of *"killed Duncan"* again (with some variable renaming for clarity):

Grammar	Logic
$\mathsf{S} \rightarrow \mathsf{NP}_{subj}\ \mathsf{VP}$	$\mathsf{NP}_{subj}{}'(\mathsf{VP}')$
$\mathsf{VP} \rightarrow \mathsf{Vi}$	Vi'
$\mathsf{VP} \rightarrow \mathsf{Vt}\ \mathsf{NP}_{obj}$	$\mathsf{NP}_{obj}{}'(\mathsf{Vt}')$
$\mathsf{NP}_{subj} \rightarrow \mathsf{Det}_{subj}\ \mathsf{N}$	$\mathsf{Det}_{subj}{}'(\mathsf{N}')$
$\mathsf{NP}_{obj} \rightarrow \mathsf{Det}_{obj}\ \mathsf{N}$	$\mathsf{Det}_{obj}{}'(\mathsf{N}')$
$\mathsf{NP}_{subj} \rightarrow \mathsf{PName}_{subj}$	$\mathsf{PName}_{subj}{}'$
$\mathsf{NP}_{obj} \rightarrow \mathsf{PName}_{obj}$	$\mathsf{PName}_{obj}{}'$

Lexicon	Logic
$every : \mathsf{Det}_{subj}$	$\lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]]$
$every : \mathsf{Det}_{obj}$	$\lambda Q[\lambda P[\lambda y[\forall x(Q(x) \supset P(x)(y))]]]$
$soldier : \mathsf{N}$	$\lambda x[soldier1(x)]$
$Duncan : \mathsf{PName}_{subj}$	$\lambda P[P(d)]$
$Duncan : \mathsf{PName}_{obj}$	$\lambda P[\lambda x[P(d)(x)]]$
$died : \mathsf{Vi}$	$\lambda x[died1(x)]$
$killed : \mathsf{Vt}$	$\lambda y[\lambda x[killed1(x, y)]]$

Grammar 2.3: Type-raising object noun phrases

$$\begin{aligned}
\mathsf{VP}' &= \mathsf{NP}_{obj}{}'(\mathsf{Vt}') \\
&= \lambda P[\lambda z[P(d)(z)]](\lambda y[\lambda x[killed1(x, y)]]) \\
&= \lambda z[\lambda y[\lambda x[killed1(x, y)]](d)(z)] \\
&= \lambda z[\lambda x[killed1(x, d)](z)] \\
&= \lambda z[killed1(z, d)]
\end{aligned}$$

This is again what we want. Its type is suitable for use in the translation rule for $\mathsf{S} \rightarrow \mathsf{NP}_{subj}\ \mathsf{VP}$.

Exercise 4 *Again compute the translations of* "killed every witch" *and* "Duncan killed every witch".

This solution has the disadvantage that subject NPs and object NPs have different types (with ramifications for their constituents such as Dets too). This scheme also becomes more complicated when we introduce ditransitive verbs: indirect object NPs will have a third type. However, there are grammatical theories, particularly versions of Categorial Grammar, that take this approach.

Within these theories of grammar, this approach to the semantics of object NPs may have the advantage of allowing a fairly straightforward constituent-coördination analysis of, e.g., *"sent a card and gave a present to every witch"*. We also note that, in these theories, it may not be necessary to give multiple grammar rules and lexical entries for subject and object NPs as we have done in grammar 2.3. Instead, type-raising might apply dynamically to some base entries during semantic translation.

Grammar	Logic
$\mathsf{S} \rightarrow \mathsf{NP\ VP}$	$\mathsf{NP}'(\mathsf{VP}')$
$\mathsf{VP} \rightarrow \mathsf{Vi}$	Vi'
$\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$	$\mathsf{Vt}'(\mathsf{NP}')$
$\mathsf{NP} \rightarrow \mathsf{Det\ N}$	$\mathsf{Det}'(\mathsf{N}')$
$\mathsf{NP} \rightarrow \mathsf{PName}$	PName'

Lexicon	Logic
every: Det	$\lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]]$
soldier: N	$\lambda x[soldier1(x)]$
Duncan: PName	$\lambda P[P(d)]$
died: Vi	$\lambda x[died1(x)]$
killed: Vt	$\lambda \mathcal{P}[\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])]]$

Grammar 2.4: Type-raising transitive verbs

While this brings computational problems of its own, it does help to overcome the major drawback of the approach as we have presented it here. Furthermore, it has the interesting advantage that it follows from a simple type-shifting rule of wider applicability, originally proposed by Peter Geach (1972) and known as the **Geach rule**:

> If an expression can occur having type $\langle a, b\rangle$, then it can also occur having type $\langle\langle c, a\rangle, \langle c, b\rangle\rangle$, for arbitrary c.

Quantified NPs ordinarily have type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$, but, by Geach's rule, taking a to be $\langle \mathbf{e}, \mathbf{t}\rangle$, b to be $\mathbf{t}$, and c to be $\mathbf{e}$, we can obtain in one step the type $\langle\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t}\rangle\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$. (An example of the rule's wider use is that it can change sentential negation, type $\langle \mathbf{t}, \mathbf{t}\rangle$ to intransitive verb negation, type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$, taking a and b to be $\mathbf{t}$ and c to be $\mathbf{e}$, and to transitive verb negation, type $\langle\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t}\rangle\rangle, \langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t}\rangle\rangle\rangle$, taking a and b to be $\langle \mathbf{e}, \mathbf{t}\rangle$ and c to be $\mathbf{e}$, and so on.)

2.4.3 Solution 3: Type-raise transitive verbs

Yet another possibility is to raise the type of transitive verbs so that they are functions from NP translations (type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$) to VP translations (type $\langle \mathbf{e}, \mathbf{t}\rangle$), i.e. Vt translations will have type $\langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$. The logical translation for *"killed"* would reflect this type by having a lambda abstraction over something of the type of NPs $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$. (Variables of this type are represented by calligraphic letters, e.g. $\mathcal{P}$.) This solution is summarised, with some sample lexical entries, as grammar 2.4. (Note that the order of arguments of the transitive verb is again the reverse of what it used to be in the original translation.)

We can show how this works by computing the translation of *"killed Duncan"* again:

$$\begin{aligned}\mathsf{VP}' &= \mathsf{Vt}'(\mathsf{NP}')\\ &= \lambda\mathcal{P}[\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])]](\lambda P[P(d)])\\ &= \lambda y[\lambda P[P(d)](\lambda x[killed1(y,x)])]\\ &= \lambda y[\lambda x[killed1(y,x)](d)]\\ &= \lambda y[killed1(y,d)]\end{aligned}$$

Again, we have a suitable VP translation.

Exercise 5 *Again compute the translations of* "killed every witch" *and* "Duncan killed every witch".

This scheme is simpler than the previous two. It suffers the disadvantage, however, that it assigns very different types to intransitive and transitive verbs. Intransitive verbs are still of type $\langle \mathbf{e}, \mathbf{t}\rangle$, but transitive verbs are of type $\langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$. This does not seem to capture the simple idea that the main difference between the two kinds of verbs is the number of NP arguments that they demand.

Following on from this, we have another disadvantage. In the translation rule that corresponds to the syntax rule $\mathsf{S} \rightarrow \mathsf{NP\ VP}$, i.e. $\mathsf{S}' = \mathsf{NP}'(\mathsf{VP}')$, the NP translation is the functor; but, in the translation rule that corresponds to the syntax rule $\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$, i.e. $\mathsf{VP}' = \mathsf{Vt}'(\mathsf{NP}')$, the NP translation is the argument. It is, perhaps, unattractive that NP translations should play these different roles (functor and argument) according to their syntactic position. Further, this solution violates a possible generalisation that can be expressed with reference to the underlying semantics concerning *syntactic* agreement across different languages. In English, for example, verb phrases must agree in number and person with the subject NP; in French, adjectives must agree with the Nbar they modify; in some languages, the transitive verb must agree with the direct object. The generalisation is (in informal terms): the category which semantically is the functor may have to agree syntactically with the category that semantically is the argument. Adjectives are functors and so may agree with their Nbars; transitive verbs are functors and so may agree with their direct objects. And if the generalisation is to work, we cannot have our first rule above. The VP must be the functor if it is to agree with the subject NP.[16]

Despite these disadvantages, this is one of the two most commonly used solutions.

2.4.4 Solution 4: Type-raise all verbal categories

Type-raising all verbal categories is the other most commonly used solution.

Suppose we let **NP** stand for the types of NPs. In our original semantic rules and types (i.e. in the rules and types we had in the previous chapter), VPs were of type $\langle \mathbf{NP}, \mathbf{t}\rangle$, intransitive verbs were of type $\langle \mathbf{NP}, \mathbf{t}\rangle$, and transitive verbs were of type $\langle \mathbf{NP}, \langle \mathbf{NP}, \mathbf{t}\rangle\rangle$, and, in these types, **NP** was **e**. But now, after the deliberations of this chapter, the type of NPs is $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$. So we can replace **NP** by $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$ in the verbal types just given. The type of VPs will become $\langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \mathbf{t}\rangle$, intransitive verbs similarly will become type $\langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \mathbf{t}\rangle$, and transitive verbs will become type

[16] Gazdar et al. (1985) present another problem with this approach (pp.189-91), concerning a proper account for the 'intensionality' of subject NPs for certain VPs. As we have little to say on intensional semantics in this book, we will not present that argument here.

Grammar	Logic
$\mathsf{S} \rightarrow \mathsf{NP\ VP}$	$\mathsf{VP}'(\mathsf{NP}')$
$\mathsf{VP} \rightarrow \mathsf{Vi}$	Vi'
$\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$	$\mathsf{Vt}'(\mathsf{NP}')$
$\mathsf{NP} \rightarrow \mathsf{Det\ N}$	$\mathsf{Det}'(\mathsf{N}')$
$\mathsf{NP} \rightarrow \mathsf{PName}$	PName'

Lexicon	Logic
every: Det	$\lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]]$
soldier: N	$\lambda x[soldier1(x)]$
Duncan: PName	$\lambda P[P(d)]$
died: Vi	$\lambda \mathcal{P}[\mathcal{P}(\lambda x[died1(x)])]$
killed: Vt	$\lambda \mathcal{P}[\lambda \mathcal{Q}[\mathcal{Q}(\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])])]]$

Grammar 2.5: Type-raising all verbal categories

$\langle\langle\mathbf{e},\mathbf{t}\rangle,\mathbf{t}\rangle, \langle\langle\langle\mathbf{e},\mathbf{t}\rangle,\mathbf{t}\rangle,\mathbf{t}\rangle\rangle$. While these look ugly, they are simply the result of putting $\langle\langle\mathbf{e},\mathbf{t}\rangle,\mathbf{t}\rangle$ in place of $\mathbf{NP}$.

Effectively, we have ensured that all verbal categories (VP, Vi and Vt) denote functions over NP types. Therefore, we can return to our original semantic rules, summarised, with some sample lexical entries, as grammar 2.5. In particular, the translation of the rule $\mathsf{S} \rightarrow \mathsf{NP\ VP}$ returns to being $\mathsf{S}' = \mathsf{VP}'(\mathsf{NP}')$ (but note also that the order of arguments is the reverse of what it used to be in the translation of the transitive verb).

To show this solution working, we compute the translation of *"Some soldier killed Duncan"*. We will assume the translations of the two NPs have already been computed. The translation of the VP is:

$$\begin{aligned}
\mathsf{VP}' &= \mathsf{Vt}'(\mathsf{NP}') \\
&= \lambda\mathcal{P}[\lambda\mathcal{Q}[\mathcal{Q}(\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])])]](\lambda P[P(d)]) \\
&= \lambda\mathcal{Q}[\mathcal{Q}(\lambda y[\lambda P[P(d)](\lambda x[killed1(y,x)])])] \\
&= \lambda\mathcal{Q}[\mathcal{Q}(\lambda y[\lambda x[killed1(y,x)](d)])] \\
&= \lambda\mathcal{Q}[\mathcal{Q}(\lambda y[killed1(y,d)])]
\end{aligned}$$

We use this in the translation of the S (with some variable renaming for clarity) as follows:

$$\begin{aligned}
\mathsf{S}' &= \mathsf{VP}'(\mathsf{NP}') \\
&= \lambda\mathcal{Q}[\mathcal{Q}(\lambda y[killed1(y,d)])](\lambda P[\exists z(soldier1(z) \wedge P(z))]) \\
&= \lambda P[\exists z(soldier1(z) \wedge P(z))](\lambda y[killed1(y,d)]) \\
&= \exists z(soldier1(z) \wedge \lambda y[killed1(y,d)](z)) \\
&= \exists z(soldier1(z) \wedge killed1(z,d))
\end{aligned}$$

This translation has the truth conditions we would like for this sentence.

Exercise 6 *Again compute the translations of* "killed every witch" *and* "Duncan killed every witch".

The main advantage of this approach is its uniformity. The criticisms of solution (3) in the previous subsection, whose subject NPs and object NPs played different semantic roles (functor and argument, respectively) and the consequences of this do not apply to this solution. The disadvantage of this solution is the complexity of the verbal category translations and the unintuitively complicated denotations assigned to VPs: they are no longer sets of individuals, they are now sets of sets of sets of individuals.

2.5 Prepositional phrase attachment

It is instructive for us to continue by introducing some simple prepositional phrases into our grammar. This will provide a further test of our treatment of quantified NPs, since PPs both contain NPs and can be contained within NPs. We can also use this extension to introduce a structural ambiguity to the grammar.

In sentence (4)

(4) *"Macbeth killed a soldier with a dagger."*

the phrase *"with a dagger"* is a prepositional phrase. *"With"*, along with words such as *"in"*, *"on"*, *"by"*, *"from"*, *"of"*, *"within"*, *"over"*, *"under"*, *"near"* and *"to"*, is a preposition (P). In a prepositional phrase (PP), the preposition is typically followed by a noun phrase (*"a dagger"* in (4)):[17]

$$\mathsf{PP} \rightarrow \mathsf{P}\ \mathsf{NP}$$

The grammar into which we will incorporate some treatment of PPs is given as grammar 2.6. This grammar uses the semantic treatment of quantified object NPs labelled solution (3) in this chapter, i.e. where transitive verbs have been type-raised so that they denote functions from NP translations to VP translations, but other verbal categories have not also been type-raised. Of solutions (3) and (4), which are the two most popular treatments of object NPs, we choose solution (3) only because it accommodates the treatment of PPs straightforwardly; the extra problems caused by using solution (4) are briefly mentioned in subsection 2.5.2.

Grammar 2.6 is, therefore, simply the same as the one presented earlier as grammar 2.4. However, you will note that the rule NP → Det N has been replaced by

[17] There are issues ignored by this. For example, (i) multiple prepositions are sometimes possible (*"Macbeth heard screams* from within *the bedroom"*); (ii) the word *"to"* is categorially ambiguous as it is also used to form infinitival verb phrases (*"Macbeth wants* to *be king"*); and (iii) most prepositions are categorially ambiguous as they can be used as 'particles' within phrasal verbs (*"Macbeth killed* off *the rumour"*); (iv) there are also cases where PPs function like ordinary NP complements to a verb (e.g. *"to"*-PPs as complements of ditransitive verbs as in *"Macbeth gave a dagger to Duncan"*, or *"by"*-PPs in passives such as *"Duncan was killed by Macbeth"*), the prepositions acting merely as 'function words', signalling the grammatical role played by the NPs in the sentence, rather than 'content words' that contribute spatial or temporal information. We ignore these and other issues here, concentrating only on the simplest PPs.

Grammar	Logic
$\mathsf{S} \rightarrow \mathsf{NP\ VP}$	$\mathsf{NP}'(\mathsf{VP}')$
$\mathsf{VP} \rightarrow \mathsf{Vi}$	Vi'
$\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$	$\mathsf{Vt}'(\mathsf{NP}')$
$\mathsf{NP} \rightarrow \mathsf{Det\ Nbar}$	$\mathsf{Det}'(\mathsf{Nbar}')$
$\mathsf{NP} \rightarrow \mathsf{PName}$	PName'
$\mathsf{Nbar} \rightarrow \mathsf{N}$	N'

Lexicon	Logic
every: Det	$\lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]]$
soldier: N	$\lambda x[soldier1(x)]$
Duncan: PName	$\lambda P[P(d)]$
died: Vi	$\lambda x[died1(x)]$
killed: Vt	$\lambda \mathcal{P}[\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])]]$

Grammar 2.6: Grammar to be extended by PPs

the rules NP → Det Nbar and Nbar → N. The introduction of the category Nbar, of semantic type $\langle \mathbf{e}, \mathbf{t} \rangle$, gives a suitable category for PP-modification.

Of course, sentence (4) is ambiguous: *"with a dagger"* can modify the NP *"a soldier"*, a reading we can paraphrase as 'a soldier with a dagger was killed by Macbeth'; or, *"with a dagger"* can modify the VP, a reading we can paraphrase as 'it was with a dagger that Macbeth killed a soldier'. This is a structural ambiguity: the sentence has two phrase-structures, and, as phrase-structures drive semantic translation, this gives it two meanings. So, in addition to adding the rule PP → P NP to grammar 2.6, we need to add the following two rules,

$$\mathsf{Nbar} \rightarrow \mathsf{Nbar\ PP}$$
$$\mathsf{VP} \rightarrow \mathsf{VP\ PP}$$

in order to give the sentence, and others like it, two distinct phrase-structure trees. (There is some evidence for rules such as NP → NP PP, Nbar → N PP and VP → V PP, but this we ignore.)

2.5.1 Noun phrase attachment

In this section, we determine semantic translations rules for the syntax rules Nbar → Nbar PP and PP → P NP, and also work out a translation for prepositions.

In the rule Nbar → Nbar PP, we know that both Nbars are of type $\langle \mathbf{e}, \mathbf{t} \rangle$. The child Nbar, therefore, denote a function, but it is a function from individuals to truth-values; it is not plausible for PPs to denote individuals, and it is also not possible for the category we choose as functor to return truth-values, since the type of the parent category in the rule is $\langle \mathbf{e}, \mathbf{t} \rangle$. It follows that the PP must be the functor category: it

must denote a function from Nbar types to Nbar types, i.e. $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$. We have arrived at the following:

- English syntax: $\mathsf{Nbar} \rightarrow \mathsf{Nbar\ PP}$; Logic translation: $\mathsf{Nbar}' = \mathsf{PP}'(\mathsf{Nbar}')$

We now need a translation for the rule $\mathsf{PP} \rightarrow \mathsf{P\ NP}$. We now know the type of these PPs to be $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$; the type of NPs is $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$. Since the latter is a function returning truth-values, it is not the functor of the translation rule, which must return PP types. It follows that the P must be the functor:

- English syntax: $\mathsf{PP} \rightarrow \mathsf{P\ NP}$; Logic translation: $\mathsf{PP}' = \mathsf{P}'(\mathsf{NP}')$

with the type of Ps being $\langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle\rangle$.

We now need a translation for prepositions. A sensible translation for an Nbar such as *"soldier with a dagger"* is

$$\lambda x_1[\exists y_1(dagger(y_1) \wedge soldier1(x_1) \wedge with1(x_1, y_1))]$$

Abstracting over the Nbar contribution gives

$$\lambda P[\lambda x_1[\exists y_1(dagger(y_1) \wedge P(x_1) \wedge with1(x_1, y_1))]](\lambda x_2[soldier1(x_2)])$$

giving the following

$$\lambda P[\lambda x_1[\exists y_1(dagger(y_1) \wedge P(x_1) \wedge with1(x_1, y_1))]]$$

as the translation of the PP. In this, if we abstract over the NP contribution, we get

$$\lambda \mathcal{P}[\lambda P_1[\lambda x_1[\mathcal{P}(\lambda y_2[P_1(x_1) \wedge with1(x_1, y_2)])]]](\lambda P_2[\exists y_1(dagger1(y_1) \wedge P_2(y_1))])$$

giving the following

$$\lambda \mathcal{P}[\lambda P_1[\lambda x_1[\mathcal{P}(\lambda y_2[P_1(x_1) \wedge with1(x_1, y_2)])]]]$$

as the translation of the preposition, i.e.:

- English syntax: *with* : P; Logic translation: $\mathsf{P}' = with' = \lambda \mathcal{P}[\lambda P_1[\lambda x_1[\mathcal{P}(\lambda y_2[P_1(x_1) \wedge with1(x_1, y_2)])]]]$

Exercise 7 *Confirm that the above works by computing the translation of the* NP "every soldier with a dagger".

2.5.2 Verb phrase attachment

By a similar argument to the one given above, the functor category in the translation of the rule $\mathsf{VP} \rightarrow \mathsf{VP\ PP}$ will be the PP: PPs denote functions from VP types to VP types.

At this point, we have obtained a reward for having chosen solution (3) for the treatment of quantified object NPs. By that solution, VPs were of type $\langle \mathbf{e}, \mathbf{t}\rangle$, making

PPs of type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$. This is the same type that we discovered for nominal modification in subsection 2.5.1 above.[18]

It is worth pointing out that if we had chosen solution (4) as our preferred solution to the treatment of quantified object NPs (solution (4) being the other widely used solution), then we would have complications at this point. By that solution, all verbal categories were type-raised to denote functions over NP types. VPs, in particular, were of type $\langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \mathbf{t}\rangle$, which would mean that VP-modifying PPs would have a different type $(\langle\langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \mathbf{t}\rangle, \langle\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle, \mathbf{t}\rangle\rangle)$ from Nbar-modifying PPs (type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \langle \mathbf{e}, \mathbf{t}\rangle\rangle$ from above). On the one hand, this looks deeply unattractive: it appears to require a separate set of syntax and semantic rules for the two forms of modification. On the other hand, although the two types for PPs look quite different, they are, in fact, both of the form $\langle a, a\rangle$ where a is either the type of Nbars or the type of VPs. If we allow type expressions to include *variables over types*, we could even say that their type *is* $\langle a, a\rangle$ (with the type-variable a constrained to range over Nbar and VP types). A type expression that contains type-variables is said to be a **polymorphic type**. The introduction of polymorphic types is by no means unmotivated in natural language semantics. For example, the type of the word *"and"* may well be polymorphic; specifically, it may be of type $\langle a, \langle a, a\rangle\rangle$, i.e. it takes two expressions of the same type and returns a third expression of that type. The word *"not"* might also be polymorphic. But, for our purposes, a polymorphic treatment is overly complex. One alternative might be to look for a general type-raising rule of which this change in PP types is an instance. However, the choice of solution (3) has given us a simpler solution.

But there is a question which we have not yet addressed: what should the denotation (and logical translation) of a VP modified by a PP be?

Consider the following sentences and possible translations of those sentences into logic (ignoring NP-modification in sentence (7)):[19]

(5) a. *"Macbeth killed Duncan."*
b. $killed1(m, d)$

(6) a. *"Macbeth killed Duncan with a dagger."*
b. $\exists x(dagger1(x) \wedge killed2(m, d, x))$

(7) a. *"Macbeth killed Duncan with a dagger in a bedroom."*
b. $\exists y(bedroom1(y) \wedge \exists x(dagger1(x) \wedge killed3(m, d, x, y)))$

In translating *"Macbeth killed Duncan"*, we used $killed1$ as a predicate denoting a relation between two individuals; in translating *"Macbeth killed Duncan with a dagger"*, we used $killed2$ as a predicate denoting a relation between three individuals (the killer, the deceased and the instrument of the killing); in translating *"Macbeth killed Duncan with a dagger in a bedroom"*, we used $killed3$ as a four-place predicate. Additional modifiers that further specify the time, place and manner of the killing would require predicates of ever greater arity.

[18] And, in fact, it is the same type as the one you should have assigned to simple nominal-modifying adjectives in exercise 2.

[19] We use the determiner *"a"* to obtain natural-sounding examples, and we assume that it translates as existential quantification.

This approach poses many practical problems. They include the difficulty for us, as system designers, to remember what each argument position signifies (e.g. that the third is the location), and the failure to capture some of the import of the original sentence (e.g. it is not clear how we would distinguish a translation of a sentence in which the killing had been *near* the bedroom from one in which the killing took place *in* the bedroom, given only a single argument position for the specification of the location).

There are other problems too. The most significant of all is that we intuitively feel that, for example, sentence (6) entails sentence (5) (in any world in which it is true that Macbeth killed Duncan with a dagger it is also true that Macbeth killed Duncan). But sentence (6)'s logic translation does *not* entail sentence (5)'s logic translation. This and other entailment relations *can* be captured (using 'meaning postulates', as described in a different context later in this subsection, but the proliferation of predicate symbols and explicitly stated entailment relationships between them is inelegant. A number of solutions has been proposed, with an overview of them being given in Parsons (1990, pp.50-61). We present a simple, elegant scheme that is credited to Donald Davidson and is now fairly widely used in computational linguistics, though we note that it does not meet with universal approval.

In this alternative treatment, we choose different translations for the sentences, as follows:

(8) a. *"Macbeth killed Duncan."*
b. $\exists e\, killed1(e, m, d)$

(9) a. *"Macbeth killed Duncan with a dagger."*
b. $\exists e(\exists x(dagger1(x) \wedge killed1(e, m, d) \wedge with1(e, x)))$

(10) a. *"Macbeth killed Duncan with a dagger in a bedroom."*
b. $\exists e(\exists y(bedroom1(y) \wedge \exists x(dagger1(x) \wedge killed1(e, m, d) \wedge with1(e, x) \wedge in1(e, y))))$

A translation such as $\exists e\, killed(e, m, d)$ can be paraphrased into pseudo-English as 'there exists an event e which was a killing by Macbeth of Duncan'. The paraphrase for the translation of sentence (9) would be 'there exists an event e and a dagger x where e was a killing by Macbeth of Duncan and the event e was carried out with the dagger x'. This style of translation is often referred to as **event semantics**, being characterised by the introduction of, and quantification over, **event variables**.[20]

Using event semantics, we no longer have the problem of remembering what a large number of argument positions signify as we did in (5b), (6b) and (7b). Similarly, we can easily represent distinctions such as the difference between *"near"* and *"in"*. Most importantly, all the translations use the same predicate, $killed1$; PP translations can be conjoined indefinitely to the rest of the translation without requiring a change of predicate symbol. This means that no explicit statement of entailment relations is needed. Simply from the meaning of logical conjunction ($\wedge$), the translation of sentence (9) entails the translation of sentence (8). (By the meaning of $\wedge$, $\Phi \wedge \Psi$ entails both Φ and Ψ on their own.) Equally correctly, the translation of sentence (8)

[20] In the case of event variables, we abandon our convention of using lower case letters from the *end* of the alphabet for individual variables; instead, we use e, e_1, e_2, etc., which are suitably mnemonic.

does not entail the translation of sentence (9), and this is as we would expect it to be from the English (just because Macbeth killed Duncan it does not follow that he killed him with a dagger, but the reverse does follow). Coupled with the fact that the solution is one that gives first-order translations to sentences (first-order logic not suffering the same computational efficiency problems suffered by higher-order logics), it is unsurprising that it has seen considerable take-up in computational linguistics.

Critics of the treatment can point to many issues of detail that are not satisfactorily worked out for all linguistic phenomena, of course. But the main substantive criticisms are 'philosophical' in nature. The scheme requires us to extend the universe of discourses of the models that we use to assign meanings to natural language sentences and formulae of logic. In addition to relatively uncontroversial individuals such as Macbeth, Duncan, various daggers and soldiers and the like, we also have to admit 'events' into the universe of discourse. In the interests of giving some semantic treatment to VP modifiers, we will brush aside all such concerns, and proceed with a solution that uses event variables.

In fact, adopting this solution, the types of all verbal categories change. Intransitive verbs, for example, are no longer one-place relations ($\langle \mathbf{e}, \mathbf{t} \rangle$), they are now two-place relations ($\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$), the extra individual being the event. Transitive verbs will now have type $\langle \langle \langle \mathbf{e}, \mathbf{t} \rangle, \mathbf{t} \rangle, \langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle \rangle$. Here are the revised lexical entries:

- English syntax: *died* : Vi; Logic translation: $\mathsf{Vi}' = died' = \lambda x[\lambda e[died1(e, x)]]$
- English syntax: *killed* : Vt; Logic translation: $\mathsf{Vt}' = killed' = \lambda \mathcal{P}[\lambda y[\lambda e[\mathcal{P}(\lambda x[killed1(e, y, x)])]]]$

The translations for the rules VP → Vi and VP → Vt NP do not change.

Now let us consider VP-modification. VPs will now have type $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$. For consistency with those PPs that modify Nbars, we want to retain the semantic type of PPs as $\langle \langle \mathbf{e}, \mathbf{t} \rangle, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$. For the rule VP → VP PP, therefore, we need a solution akin to solution (1) from subsection 2.4.1 for the treatment of quantified object NPs, where we 're-arranged' the translation of the verb phrase. Here, we will take the PP to be the functor, but it will need to 'break into' the translation of the child VP in order to apply to something of type $\langle \mathbf{e}, \mathbf{t} \rangle$, the remainder (i.e. the remaining $\mathbf{e}$ of the VP translation and the second $\langle \mathbf{e}, \mathbf{t} \rangle$ of the PP translation) forming the result, this, correctly, being a VP type, i.e. $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$:

- English syntax: VP → VP PP; Logic translation: $\mathsf{VP}' = \lambda x[\mathsf{PP}'(\mathsf{VP}'(x))]$

The translation of a VP such as *"killed Duncan with a dagger"* will be

$$\lambda x_1[\lambda e[\exists y_1(dagger1(y_1) \wedge killed1(e, x_1, d) \wedge with1(e, y1))]]$$

Exercise 8 *Confirm that this is the translation.*

Since VPs are now of type $\langle \mathbf{e}, \langle \mathbf{e}, \mathbf{t} \rangle \rangle$, the translation rule for S → NP VP also needs modification. It is at this point that the event variable should be existentially quantified. We will existentially quantify it prior to applying the NP translation (thus

Grammar	Logic
$\mathsf{S} \rightarrow \mathsf{NP\ VP}$	$\mathsf{NP}'(\lambda x[\exists e(\mathsf{VP}'(x)(e))])$
$\mathsf{VP} \rightarrow \mathsf{Vi}$	Vi'
$\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$	$\mathsf{Vt}'(\mathsf{NP}')$
$\mathsf{VP} \rightarrow \mathsf{VP\ PP}$	$\lambda x[\mathsf{PP}'(\mathsf{VP}'(x))]$
$\mathsf{NP} \rightarrow \mathsf{Det\ Nbar}$	$\mathsf{Det}'(\mathsf{Nbar}')$
$\mathsf{NP} \rightarrow \mathsf{PName}$	PName'
$\mathsf{Nbar} \rightarrow \mathsf{N}$	N'
$\mathsf{Nbar} \rightarrow \mathsf{Nbar\ PP}$	$\mathsf{PP}'(\mathsf{Nbar}')$
$\mathsf{PP} \rightarrow \mathsf{P\ NP}$	$\mathsf{P}'(\mathsf{NP}')$

Lexicon	Logic
every: Det	$\lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]]$
soldier: N	$\lambda x[soldier1(x)]$
Duncan: PName	$\lambda P[P(d)]$
died: Vi	$\lambda x[\lambda e[died1(e,x)]]$
killed: Vt	$\lambda \mathcal{P}[\lambda y[\lambda e[\mathcal{P}(\lambda x[killed1(e,y,x)])]]]$
with: P	$\lambda \mathcal{P}[\lambda P_1[\lambda x_1[\mathcal{P}(\lambda y_2[P_1(x_1) \wedge with1(x_1,y_2)])]]]$

Grammar 2.7: Grammar extended by PPs

the existential quantification of the event variable will, by default, lie within the scope of any quantification introduced by the subject NP —see section 2.6.4 for a discussion of this).

- English syntax: $\mathsf{S} \rightarrow \mathsf{NP\ VP}$; Logic translation: $\mathsf{S}' = \mathsf{NP}'(\lambda x[\exists e(\mathsf{VP}'(x)(e))])$

Exercise 9 *Use your answer to exercise 8 to compute a translation for* "Macbeth killed Duncan with a dagger"*.*

Making use of all the new rules introduced in this subsection and the previous one, sentence (4) (*"Macbeth killed a soldier with a dagger"*) will now be assigned two phrase-structure trees, these licensing two semantic translations:[21]

$$\exists e(\exists x(\exists y(dagger1(y) \wedge soldier1(x) \wedge with1(x,y)) \wedge killed1(e,m,x)))$$

$$\exists e(\exists y(dagger1(y) \wedge \exists x(soldier1(x) \wedge killed1(e,m,x)) \wedge with1(e,y)))$$

We summarise the resulting grammar as grammar 2.7.

[21] *"with"* might more appropriately be translated as different predicate symbols in the two translations: *with*2 or *accomp* to denote the relation of 'accompaniment' between two individuals in the first translation, and *with*1 or *instr* to denote the relation between an event and its instrument in the second translation.

A common refinement is to make use of an idea that was introduced independently, usually credited to Charles Fillmore, but which can be introduced into an event-based semantics quite easily. The idea is that there is a small number of abstract relations that can hold between events and the objects that participate in events. These relations have been called **cases**, **case roles** and **thematic roles**, among other names. We will use the term 'thematic roles'.[22]

There is only limited agreement on what the roles are and how many there might be, though it is generally accepted that there is a small number of them. Let us introduce three of the more common thematic roles.

- **Agent**: the instigator of the event; usually taken to be a participant who acts with intent;
- **Patient** (also referred to in some books as the Object or Theme): the object acted upon in the event;
- **Instrument**: the tool, material or force used to perform the event.

Other roles that have been proposed include the Experiencer, the Beneficiary, the Location, the Possessor, the Destination, the Recipient, the Source and the Accompanier.

We illustrate the three roles (agent, patient and instrument) with the following sentences, which we assume are being used to describe the same event in different ways:

(11) *"Banquo broke the window with the hammer."*

(12) *"The hammer broke the window."*

(13) *"The window was broken by Banquo."*

(14) *"The window broke."*

In all of these, Banquo (where mentioned) is the agent of the breaking event described by the sentence, the window is the patient of the event, and the hammer (where mentioned) is the instrument in the event. Notice that there is some, but not a one-to-one, correlation with traditional syntactic functions, e.g. the subject NP may be the agent (as in (11)), the instrument (as in (12)), or the patient (as in (13) and (14)); the object NP may be the patient (as in (11) and (12)); and various PPs can fill roles too (such as the instrument in (11) and the agent in (13)). It follows, therefore, that some roles will be filled by the entities described by the arguments of the verb, e.g. the subject and object NPs; other roles will be filled by the entities described by the NPs mentioned in various prepositional phrases (PPs). (This is a simplification: other parts of speech, e.g. adverbs, might describe thematic role fillers too).

If thematic roles are to be more than just a descriptive device, we need to state (model-theoretic) properties about them. One way to do this is to use **meaning postulates**. In their original conception, meaning postulates are *constraints* on models,

[22]Use of the word 'case' can conflict with the use of this word to refer to the way that noun morphology can reflect the syntactic function of a noun in a sentence; in this latter usage, cases include nominative (subject position), accusative (object position), and so on.

models that satisfy the constraints being called **admissible models**. The evaluation of the denotation of expressions is then limited only to these admissible models.

For example, consider sentence (10), *"Macbeth killed Duncan with a dagger in a bedroom"*, whose logic translation is:

$$\exists e(\exists y(bedroom1(y) \wedge \exists x(dagger1(x) \wedge killed1(e, m, d) \wedge with1(e, x) \wedge in1(e, y))))$$

We would write meaning postulates that describe the properties of the objects that fill argument positions of the predicates. Since, for example, the second argument of $killed1/3$ is the agent of the killing, we might write postulates that say (i) that whatever object is denoted by the term that occupies the second argument position of a $killed1/3$ predicate symbol is its agent, and (ii) that agents are objects that act with intent in events.

$$\forall e \forall x \forall y(killed1(e, x, y) \supset agt(e, x))$$

$$\forall e \forall x(agt(e, x) \supset acted\_intentionally(x, e))$$

In fact, it would be more common to make the roles explicit in the logical translations of utterances themselves using binary predicates such as *agt*, *pat* and *instr*. For example, the translation of sentence (10) might become

$$\exists e(\exists y(bedroom1(y) \wedge \exists x(dagger1(x) \wedge killed1(e) \\ \wedge agt(e, m) \wedge pat(e, d) \wedge instr(e, x) \wedge in1(e, y))))$$

Note that the predicate $killed1$ is now a one-place predicate and its relationships with the killer and killed person are expressed using the binary predicates agt, pat and $instr$. This is achieved by revising the lexical semantics of the words (see exercise 10). We would then include meaning postulates, as before, to state the properties of the relations denoted by the predicates agt, pat, etc.

The status of meaning postulates needs further discussion. We have shown meaning postulates as if they were formulae of logic. But, in fact, meaning postulates, in their original conception, are statements *about* the logic, specifically about the logical interpretation of the symbols of the logic. They are, therefore, statements to be made in our metalanguage, not our object language. Using object language symbols, such as $\forall$ and $\supset$, is thus misleading.[23]

For the rest of this chapter, not only will we not consider meaning postulates any more, we shall not even use thematic roles, as they clutter logical translations. Therefore, we will use translations such as those given as (8b), (9b) and (10b) above.

Exercise 10 *Revise the lexicon given in grammar 2.7 so that translations contain explicit thematic roles.*

[23] Some texts write meaning postulates prefixed by the symbol $\Box$, e.g.

$$\Box(\forall e \forall x(agt(e, x) \supset acted\_intentionally(x, e)))$$

The box is (informally) read as 'it is necessarily the case that', which helps to emphasise that the statement constrains models. It still does not avoid the confusion between object language and metalanguage uses of symbols.

2.6 Scope ambiguity

Our consideration of quantified NPs is still not at an end, as they give rise to a form of ambiguity, and we have not yet accounted for this.

The following sentence is ambiguous:

(15) *"Every soldier killed some witch."*

On one reading, every soldier killed some witch or another, i.e. they all killed at least one witch, and the witch killed by one soldier is quite possibly different from that killed by another soldier. The logic translation corresponding to this reading is:[24]

$$\forall x(soldier1(x) \supset \exists y(witch1(y) \wedge killed1(x, y)))$$

In this formula, the universal quantifier has the existential quantifier in its scope. We say that the universal quantifier has wide scope and the existential quantifier has narrow scope, or that the universal quantifier outscopes the existential quantifier. (Generally in the formulae in this book, quantifiers to the left have quantifiers to the right in their scope, unless brackets dictate otherwise.)

On the other reading, there is (at least) one witch who was killed by all the soldiers: they all killed the same witch. The corresponding translation is:

$$\exists y(witch1(y) \wedge \forall x(soldier1(x) \supset killed1(x, y)))$$

In this formula, the existential quantifier has the universal quantifier in its scope. We say that the existential has wide scope and the universal has narrow scope, or that the existential has the universal in its scope.

More generally, most sentences which contain more than one quantified NP will be ambiguous in this way. In recognition of the fact that such ambiguities are characterised by the scope of the quantifiers, these ambiguities are referred to as **scope ambiguities**.[25]

Of course, utterances of these sentences often have *preferred* readings, but our task for now is to show how all the possible readings can be computed. (We briefly discuss preferred readings in subsection 2.6.4.)

The syntactic and semantic rules that we developed in the earlier parts of this chapter (irrespective of which solution we use to compute the translation of object NPs) fail to compute multiple translations for sentences which contain more than one quantified NP. Clearly, if our grammar assigns only one phrase-structure tree to

[24] To avoid unnecessary clutter, where possible we exclude event variables from logic translations in the rest of this chapter.

[25] Scope ambiguities also arise due to the interaction of logical operators such as tense, modal and intensional operators with quantified NPs. For example, the following sentence

"Macbeth sought a witch."

is ambiguous between a 'non-specific' reading, in which any witch will do, and a 'specific' reading, where a certain witch is sought. This ambiguity can be accounted for by having readings in which the existential quantification takes narrow or wide scope relative to the intensional operator that is introduced by use of the verb *"sought"*. We do not pursue these other causes of scope ambiguities any further, because we have not addressed the semantics of these operators in this book.

such sentences, and the lexicon assigns the words of the sentence only one semantic translation, then we will not get multiple readings.

Sentence (15), for example, will have only one phrase-structure tree and, assuming it has no semantically lexically ambiguous words, will hence have only one reading. The reading that will be produced is the one in which the universal has wider scope ($\forall x(soldier1(x) \supset \exists y(witch1(y) \wedge killed1(x,y)))$). This reading is the one in which the order of the quantifiers preserves the order of their corresponding determiners in the original sentence.

In what follows we consider a variety of ways of changing the grammar and/or the semantic translation process so that sentences that contain multiple quantified NPs *do* receive multiple translations. Ideal solutions satisfy at least three properties: they are sound, complete and compositional. A solution is **sound** if it does not produce readings it should not; a solution is **complete** if it produces all the readings that it should. A **compositional** solution will be capable not only of assigning multiple translations (and denotations) to complete sentences but also to constituents of the sentences, when appropriate. For example, just as the sentence *"Duncan gave every soldier a medal"* should have two translations (and denotations), its verb phrase, *"gave every soldier a medal"*, should also have two translations (and denotations).

Before describing our favoured solution, we briefly consider other solutions, whereby scope ambiguities are treated as special cases of structural ambiguities or of lexical ambiguities.

2.6.1 Scope ambiguities as a form of structural ambiguity

If scope ambiguity is to be treated as a form of *structural* ambiguity, then we need to give sentence (15) and sentences like it, multiple distinct phrase-structure trees. Then, given that semantic translation is driven by the phrase-structures, it will be possible for multiple readings for the sentence to be computed.

A large body of linguists, including Noam Chomsky, has adopted a particular version of this. Their approach is to use a different theory of grammar from the one we have used so far in this book. In particular, their theory is **polystratal**. What this means is that the theory uses more than one layer or stratum of syntactic representation. (By contrast, we have been presenting a **monostratal** theory, having only one layer of syntactic representation.) The first incarnation of their theory went under the name of Transformational Grammar. The modern re-incarnation of the theory is called Government-Binding Theory, or GB. Our presentation below will be based on Transformational Grammar, which is perhaps the more intuitive of the two. Many details have changed in the transition to GB but the way that a polystratal theory can account for scope ambiguity should become clear.

In Transformational Grammar, the syntactic representations (phrase-structure trees) we have presented so far in this book would be referred to as **surface structures**. But there would be at least one more layer of syntactic representation, the layer of **deep structure**. It is crucial to understand that deep structures are just as syntactic as surface structures. They still say nothing about meaning (although these are the structures on which some aspects of semantic interpretation is defined). There would also be special syntactic processes (i.e. processes that carry out structural manipulation without regard to the meaning of the structures they manipulate)

to convert surface structures into deep structures or *vice versa*.

An example of a syntactic construction that motivates a polystratal theory is the passive. *"Macbeth killed Duncan"* and *"Duncan was killed by Macbeth"* would be sentences receiving different surface structures but the same deep structure (in the early versions of the theory at least). The identity function maps the deep structure to the surface structure for *"Macbeth killed Duncan"*, but a more complex transformation maps the deep structure to the surface structure for *"Duncan was killed by Macbeth"*. Given that semantic interpretation is determined by the deep structure in these theories, we get an account of why an active sentence and its corresponding passive receive the same truth-conditions: their interpretation is computed from their common deep structure.

The polystratal theory then further provides an account of scope ambiguities. This account is the opposite of the account for the identical truth-conditions of active and passive sentences. Instead of one deep structure having multiple surface structures, the account of scope ambiguity posits multiple deep structures having a single surface structure. The theory says, for example, that although sentence (15), *"Every soldier killed some witch"*, has only one surface structure, this can be derived from *more than one deep structure*. The fact that there are multiple deep structures enables us to have more than one semantic interpretation.

In exercise 3 of chapter 1, you showed, for very simple sentences, that active sentences and their corresponding passive sentences can receive the same truth-conditions without the overhead of an extra layer of syntactic representation. It transpired during the 1970s and early 1980s that *all* of the original syntactic constructions for which deep structures were originally thought to be necessary could be handled in a monostratal grammar. Arguably, if several layers of syntactic representation are not necessary to account for passive sentences and other constructions, then it would be undesirable to use an extra layer of syntactic representation simply to account for scope ambiguities. Obviously, given their popularity among linguists (but not so much among computational linguists), we cannot dismiss polystratal theories out of hand. There is clearly an ideological issue at stake, and we can do no more in this book than state our ideological preference for monostratal theories and to include pointers, in the further reading section, to the literature on polystratal theories.

2.6.2 Scope ambiguities as a form of lexical ambiguity

In showing how scope ambiguities can be a form of *lexical* ambiguity, we use solution (4) from subsection 2.4.4 to the problem of quantified object NPs. It is, of course, unfortunate to switch from using solution (3), as we had been doing, to using a different solution, but the treatment of scope ambiguity as a form of lexical ambiguity is hugely simplified by doing so. It turns out that this is not a much favoured treatment, and we can revert to solution (3) in our favoured treatment in the next subsection.

In solution (4), the logic translation of a Vt such as *"killed"* is $\lambda\mathcal{P}[\lambda\mathcal{Q}[\mathcal{Q}(\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])])]]$. This translation ensures that the subject NP has the object NP in its scope, as can be seen from this demonstration for *"killed some witch"* (intermediate lambda conversions are not shown):

$$\begin{aligned}\mathsf{VP}' &= \mathsf{Vt}'(\mathsf{NP}')\\ &= \lambda\mathcal{P}[\lambda\mathcal{Q}[\mathcal{Q}(\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])])]](\lambda P[\exists z(witch1(z)\wedge P(z))])\\ &= \lambda\mathcal{Q}[\mathcal{Q}(\lambda y[\exists z(witch1(z)\wedge killed1(y,z))])]\end{aligned}$$

Consider a subject NP translation such as $\lambda P[\forall w(soldier1(w)\supset P(w))]$. It should be clear that when the VP translation is applied to the subject NP translation, the NP translation will replace $\mathcal{Q}$ in the above, and then, by a further lambda conversion, the expression with the existential quantifier will end up replacing P. It will therefore be within the scope of the universal introduced by the subject NP.

Suppose we take the Vt translation and swap the order of $\mathcal{P}$ and $\mathcal{Q}$ in the body of the lambda expression and of x and y in the argument to $killed1$. We get $\lambda\mathcal{P}[\lambda\mathcal{Q}[\mathcal{P}(\lambda y[\mathcal{Q}(\lambda x[killed1(x,y)])])]]$. Now we can obtain the other scoping where the object NP has the subject NP in its scope, as demonstrated for *"killed some witch"* (intermediate lambda conversions not shown again):

$$\begin{aligned}\mathsf{VP}' &= \mathsf{Vt}'(\mathsf{NP}')\\ &= \lambda\mathcal{P}[\lambda\mathcal{Q}[\mathcal{P}(\lambda y[\mathcal{Q}(\lambda x[killed1(x,y)])])]](\lambda P[\exists z(witch1(z)\wedge P(z))])\\ &= \lambda\mathcal{Q}[\exists z(witch1(z)\wedge\mathcal{Q}(\lambda x[killed1(x,z)]))]\end{aligned}$$

Now when this is applied to the translation of a subject NP such as $\lambda P[\forall w(soldier1(w)\supset P(w))]$, it is clear that the subject NP, in replacing $\mathcal{Q}$ will fall within the object NP's scope.

In summary then, we can get both readings by treating *"killed"* and all other transitive verbs as semantically lexically ambiguous; they would have two meanings, as follows:

	English syntax	Logic translation
•	*killed* : Vt;	$\mathsf{Vt}' = killed' = \lambda\mathcal{P}[\lambda\mathcal{Q}[\mathcal{Q}(\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])])]]$
		$\mathsf{Vt}' = killed' = \lambda\mathcal{P}[\lambda\mathcal{Q}[\mathcal{P}(\lambda y[\mathcal{Q}(\lambda x[killed1(x,y)])])]]$

There are a number of problems with this. Firstly, *"killed"* is intuitively unambiguous. Treating it as lexically ambiguous puts it on a footing with the kind of ambiguity found with *"draw"* (as in drawing the curtains versus drawing a picture) and *"throw"* (as in throwing a stone versus throwing a party). But, that is misleading: only a single predicate symbol, $killed1$, is involved. Secondly, the approach can find multiple derivations of logically equivalent formulae. For example, sentences that contain a transitive verb but only one or even no quantified NPs (e.g. *"Macbeth killed every witch"*) will receive two translations, though the translations will be identical ($\forall x(witch1(x)\supset killed1(m,x))$). From a theoretical point of view, there is nothing wrong with this; but from a *computational* point of view, it is undesirable. Finally, while *"killed"* now has two translations, ditransitive verbs such as *"gives"* will need *six* translations in the lexicon in order to give all possible permutations of their subject, direct object and indirect object NPs. This is not attractive.

2.6.3 Scope ambiguities as a separate type of ambiguity

The final solution we consider treats scope ambiguities as a type of ambiguity different from structural and lexical ambiguities. A popular strategy is to use some form of **storage**, whereby, whenever in the translation of a sentence an NP is encountered, the

NP translation is put into storage and a 'dummy' translation is used in its place. As the process of translation proceeds, there is an option at various points to retrieve the NP translation from store and apply it to the translation of the rest of the sentence. The NP translation which is pulled out of store last will be the one that has the widest scope.

The basic idea here comes from Richard Montague who termed it **quantifying in**. The storage implementation of the idea comes from Robin Cooper and is often termed **Cooper Storage**. We are going to present a simplified version of Cooper storage that draws ideas from a number of presentations of the topic.

Every constituent of the sentence being translated will now have *two* elements of semantic translation associated with it: we will call them the **matrix** and the **store**. The matrix will be a well-formed expression of logic as before. The matrix may contain free variables reflecting the fact that some or all of the NP translations will not have been 'cached out'. The store will be a set of NP translations. For each syntax rule there will now be two corresponding semantic translation rules. The first will say how the matrix of the LHS category is composed from the matrixes of its constituents. These rules will be just like the semantic translation rules that we have used up to now in this and the preceding chapter. The second rule will say how the store for the LHS category is composed from the stores of its constituents.

We will use some notation for this. If we want to say that the matrix of an S is the result of applying the matrix of an NP to the matrix of a VP, i.e. $\text{Matrix}(\mathsf{S}) = \text{Matrix}(\mathsf{NP})(\text{Matrix}(\mathsf{VP}))$, we will use the prime notation as before, i.e. $\mathsf{S}' = \mathsf{NP}'(\mathsf{VP}')$. If we want to say that the store of an S is the result of union-ing the store of an NP and the store of a VP, i.e. $\text{Store}(\mathsf{S}) = \text{Store}(\mathsf{NP}) \cup \text{Store}(\mathsf{VP})$, we will use a double prime notation, i.e. $\mathsf{S}'' = \mathsf{NP}'' \cup \mathsf{VP}''$.

We present, as grammar 2.8, the result of bringing this approach to bear on grammar 2.6. (We have reverted to solution (3) for the treatment of quantified object NPs, but we do not include any of the material needed to deal with PPs as this would clutter our presentation.)

The lexicon is not much changed. The matrix for each lexical entry is the same as the semantic translation we gave in grammar 2.6. The store is always the empty set. (Of course, this means that the store need not be shown at all; but showing it, emphasises that it is empty and also gives a more uniform treatment.)

Most of the grammar is not different either. With the exception of the rule $\mathsf{NP} \rightarrow \mathsf{Det\ Nbar}$, the matrix rules are the same as the semantic translation rules we gave in grammar 2.6. Each store rule simply forms the union of the stores of the constituents. For example, the matrix rule corresponding to the syntactic rule $\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$ is the same as the semantic translation rule we gave earlier (i.e. $\mathsf{VP}' = \mathsf{Vt}'(\mathsf{NP}')$) and the store rule forms the union of the stores of the Vt and the NP (i.e. $\mathsf{VP}'' = \mathsf{Vt}'' \cup \mathsf{NP}''$).

This is something of a simplification as it fails to show the option of retrieving one or more of the NP translations from the store. This option is certainly available at S and NP constituents and may well be available at other constituents too. Therefore, strictly, a constituent's store is the union of its immediate subconstituents' stores less any items retrieved from those stores. And, if items are retrieved, then they should be applied to the matrix. This should be kept in mind as we proceed with the simplified presentation.

Grammar	Matrix rule	Store rule
$\mathsf{S} \rightarrow \mathsf{NP\ VP}$	$\mathsf{S}' = \mathsf{NP}'(\mathsf{VP}')$	$\mathsf{S}'' = \mathsf{NP}'' \cup \mathsf{VP}''$
$\mathsf{VP} \rightarrow \mathsf{Vi}$	$\mathsf{VP}' = \mathsf{Vi}'$	$\mathsf{VP}'' = \mathsf{Vi}''$
$\mathsf{VP} \rightarrow \mathsf{Vt\ NP}$	$\mathsf{VP}' = \mathsf{Vt}'(\mathsf{NP}')$	$\mathsf{VP}'' = \mathsf{Vt}'' \cup \mathsf{NP}''$
$\mathsf{NP} \rightarrow \mathsf{Det\ Nbar}$	$\mathsf{NP}' = \lambda P[P(z)]$	$\mathsf{NP}'' = \mathsf{Det}'' \cup \mathsf{Nbar}'' \cup \{\langle \mathsf{Det}'(\mathsf{Nbar}'), z\rangle\}$
$\mathsf{NP} \rightarrow \mathsf{PName}$	$\mathsf{NP}' = \mathsf{PName}'$	$\mathsf{NP}'' = \mathsf{PName}''$
$\mathsf{Nbar} \rightarrow \mathsf{N}$	$\mathsf{Nbar}' = \mathsf{N}'$	$\mathsf{Nbar}'' = \mathsf{N}''$

Lexicon	Matrix	Store
every: Det	$\lambda Q[\lambda P[\forall x(Q(x) \supset P(x))]]$	$\emptyset$
soldier: N	$\lambda x[soldier1(x)]$	$\emptyset$
Duncan: PName	$\lambda P[P(d)]$	$\emptyset$
died: Vi	$\lambda x[died1(x)]$	$\emptyset$
killed: Vt	$\lambda \mathcal{P}[\lambda y[\mathcal{P}(\lambda x[killed1(y,x)])]]$	$\emptyset$

Grammar 2.8: Simplified storage grammar

The rule $\mathsf{NP} \rightarrow \mathsf{Det\ Nbar}$ is the one that deserves our further attention. This is the rule that creates a 'dummy' matrix and stores the real translation of the NP.

The 'dummy' matrix is $\lambda P[P(z)]$. Significantly, this matrix has the same semantic type as an ordinary NP translation (type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$). It is because the matrix has the same type as an ordinary NP translation that all the other matrix rules did not need changing: they are still fed with a translation of a suitable kind on which to operate. However, the 'dummy' matrix does have the peculiarity of introducing a free variable, z, into the logical translation. (Note that a new free variable is introduced for every NP, i.e. we do not always use z, we use a fresh free variable for each NP.)

The store will contain the union of the stores of the Det and the Nbar, and an additional item, $\langle \mathsf{Det}'(\mathsf{Nbar}'), z\rangle$.[26] Now you can see what is meant by putting the NP translation into store. The normal translation of the NP, $\mathsf{Det}'(\mathsf{Nbar}')$, is being placed into the store. It is accompanied by a variable. This is the same variable as was free in the 'dummy' matrix of the NP, i.e. in this case, z. By putting the same variable z into both the matrix and the store, we ensure that we can tie together the 'dummy' matrix and the stored item at a later point in the processing. (We say that the two are *co-indexed* by z.)

Figure 2.10 shows an annotated tree for the sentence *"Every soldier killed some witch"*; since labelling each node of the tree with both its matrix and its store would, lead to a cluttered tree, nodes are instead labelled with letters, and the corresponding matrixes and stores are tabulated in table 2.2; no intermediate (unreduced) lambda

[26] In the entries in grammar 2.8, this item appears as $\{\langle \mathsf{Det}'(\mathsf{Nbar}'), z\rangle\}$, i.e. enclosed in curly braces. This is to turn it into a (singleton) set so that it can be union-ed with Det'' and Nbar''.

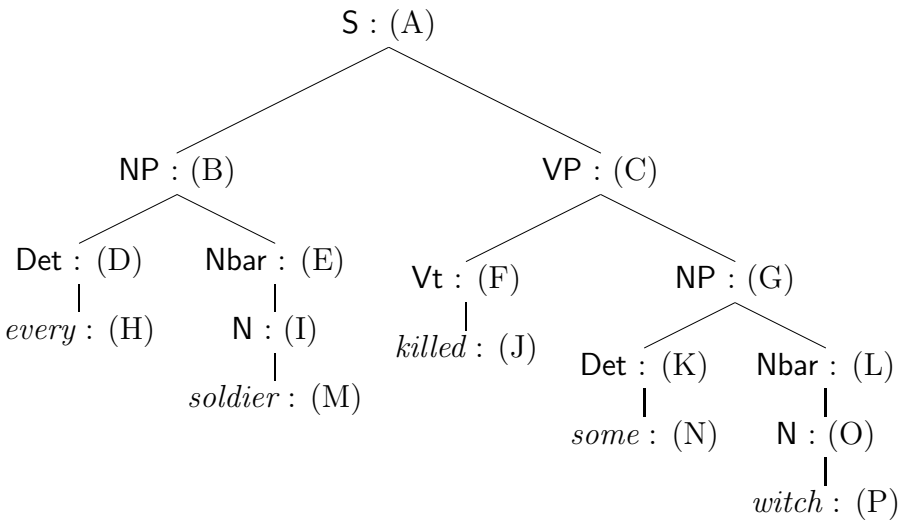

Figure 2.10: Annotated tree for *"Every soldier killed some witch"*

abstracts are shown.

At no point in this particular example do we avail ourselves of the option of retrieving items from the stores. Therefore, from the figure and table, we see that the S node (node A) has matrix $killed1(z_3, z_7)$ and store $\{\langle \lambda P_1[\forall x_1(soldier1(x_1) \supset P_1(x_1))], z_3\rangle, \langle \lambda P_5[\exists x_5(witch1(x_5) \wedge P_5(x_5))], z_7\rangle\}$.
Thus we have an unscoped matrix with free variables in it, plus a store with two NP translations 'on ice'.

We must now consider how to retrieve an NP translation from storage and use it to scope a matrix. We will show how this would work for retrieval at the root node in figure 2.10; we will not at this stage discuss the option of retrieval at deeper nodes in the tree.

Let S′ be the matrix of the S. An item in the store is of the form $\langle$NP′, $z\rangle$, i.e. NP′ is the stored NP translation and z is the stored variable (which is free in S′). To use a retrieved item to scope S′, we compute:

$$\mathsf{NP}'(\lambda z[\mathsf{S}'])$$

i.e. we wrap the S matrix in λz (giving something of type $\langle \mathbf{e}, \mathbf{t}\rangle$) and then use this as the argument to the NP translation (which is of type $\langle\langle \mathbf{e}, \mathbf{t}\rangle, \mathbf{t}\rangle$).

This is repeated for each item in the store. Different orders by which the stored NP translations are retrieved and applied to the S′ will give rise to different scopings of the S′. To show this, we will continue the example.

The matrix and store of the S are:

$$\begin{array}{lcl} \mathsf{S}' & = & killed1(z_3, z_7) \\ \mathsf{S}'' & = & \{\langle \lambda P_1[\forall x_1(soldier1(x_1) \supset P_1(x_1))], z_3\rangle, \langle \lambda P_5[\exists x_5(witch1(x_5) \wedge P_5(x_5))], z_7\rangle\} \end{array}$$

If we retrieve the first NP in the store, we compute a new S′ as follows:

$$\lambda P_1[\forall x_1(soldier1(x_1) \supset P_1(x_1))](\lambda z_3[killed1(z_3, z_7)])$$

A: $\text{Matrix} = killed1(z_3, z_7)$
$\text{Store} = \{\langle \lambda P_1[\forall x_1(soldier1(x_1) \supset P_1(x_1))], z_3\rangle, \langle \lambda P_5[\exists x_5(witch1(x_5) \wedge P_5(x_5))], z_7\rangle\}$

B: $\text{Matrix} = \lambda P_3[P_3(z_3)]$
$\text{Store} = \{\langle \lambda P_1[\forall x_1(soldier1(x_1) \supset P_1(x_1))], z_3\rangle\}$

C: $\text{Matrix} = \lambda y_4[killed1(y_4, z_7)]$
$\text{Store} = \{\langle \lambda P_5[\exists x_5(witch1(x_5) \wedge P_5(x_5))], z_7\rangle\}$

D: $\text{Matrix} = \lambda Q_1[\lambda P_1[\forall x_1(Q_1(x_1) \supset P_1(x_1))]]$
$\text{Store} = \emptyset$

E: $\text{Matrix} = \lambda x_2[soldier1(x_2)]$
$\text{Store} = \emptyset$

F: $\text{Matrix} = \lambda \mathcal{P}_4[\lambda y_4[\mathcal{P}_4(\lambda x_4[killed1(y_4, x_4)])]]$
$\text{Store} = \emptyset$

G: $\text{Matrix} = \lambda P_7[P_7(z_7)]$
$\text{Store} = \{\langle \lambda P_5[\exists x_5(witch1(x_5) \wedge P_5(x_5))], z_7\rangle\}$

H: $\text{Matrix} = \lambda Q_1[\lambda P_1[\forall x_1(Q_1(x_1) \supset P_1(x_1))]]$
$\text{Store} = \emptyset$

I: $\text{Matrix} = \lambda x_2[soldier1(x_2)]$
$\text{Store} = \emptyset$

J: $\text{Matrix} = \lambda \mathcal{P}_4[\lambda y_4[\mathcal{P}_4(\lambda x_4[killed1(y_4, x_4)])]]$
$\text{Store} = \emptyset$

K: $\text{Matrix} = \lambda Q_5[\lambda P_5[\exists x_5(Q_5(x_5) \wedge P_5(x_5))]]$
$\text{Store} = \emptyset$

L: $\text{Matrix} = \lambda x_6[witch1(x_6)]$
$\text{Store} = \emptyset$

M: $\text{Matrix} = \lambda x_2[soldier1(x_2)]$
$\text{Store} = \emptyset$

N: $\text{Matrix} = \lambda Q_5[\lambda P_5[\exists x_5(Q_5(x_5) \wedge P_5(x_5))]]$
$\text{Store} = \emptyset$

O: $\text{Matrix} = \lambda x_6[witch1(x_6)]$
$\text{Store} = \emptyset$

P: $\text{Matrix} = \lambda x_6[witch1(x_6)]$
$\text{Store} = \emptyset$

Table 2.2: Matrixes and stores for *"Every soldier killed some witch"*

$$\equiv \forall x_1(soldier1(x_1) \supset \lambda z_3[killed1(z_3, z_7)](x_1))$$
$$\equiv \forall x_1(soldier1(x_1) \supset killed1(x_1, z_7))$$

The matrix and store are now as follows:

$$\begin{array}{lcl} \mathsf{S}' & = & \forall x_1(soldier1(x_1) \supset killed1(x_1, z_7)) \\ \mathsf{S}'' & = & \{\langle \lambda P_5[\exists x_5(witch1(x_5) \wedge P_5(x_5))], z_7\rangle\} \end{array}$$

If we now retrieve the second NP translation, we compute the final translation for the S, as follows:

$$\begin{array}{l} \lambda P_5[\exists x_5(witch1(x_5) \wedge P_5(x_5))](\lambda z_7[\forall x_1(soldier1(x_1) \supset killed1(x_1, z_7))]) \\ \quad \equiv \exists x_5(witch1(x_5) \wedge \lambda z_7[\forall x_1(soldier1(x_1) \supset killed1(x_1, z_7))](x_5)) \\ \quad \equiv \exists x_5(witch1(x_5) \wedge \forall x_1(soldier1(x_1) \supset killed1(x_1, x_5))) \end{array}$$

We have obtained the reading in which the existential outscopes the universal (there is at least one particular witch whom every soldier killed).

Suppose, instead, we had retrieved the NPs in reverse order to the above. If we retrieve the second NP first, we compute a new S′ as follows:

$$\begin{array}{l} \lambda P_5[\exists x_5(witch1(x_5) \wedge P_5(x_5))](\lambda z_7[killed1(z_3, z_7)]) \\ \quad \equiv \exists x_5(witch1(x_5) \wedge \lambda z_7[killed1(z_3, z_7)](x_5)) \\ \quad \equiv \exists x_5(witch1(x_5) \wedge killed1(z_3, x_5)) \end{array}$$

We now retrieve the remaining NP, as follows:

$$\begin{array}{l} \lambda P_1[\forall x_1(soldier1(x_1) \supset P_1(x_1))](\lambda z_3[\exists x_5(witch1(x_5) \wedge killed1(z_3, x_5))]) \\ \quad \equiv \forall x_1(soldier1(x_1) \supset \lambda z_3[\exists x_5(witch1(x_5) \wedge killed1(z_3, x_5)))](u) \\ \quad \equiv \forall x_1(soldier1(x_1) \supset \exists x_5(witch1(x_5) \wedge killed1(x_1, x_5))) \end{array}$$

This time, we get the reading in which the universal outscopes the existential (each soldier killed a possibly different witch).

Obviously, in general, we would want to retrieve stored NP translations in all possible orders. In the above example, the store has two items, so there are two orders. If the store had contained three items, then there would have been six possible orderings. In general, if there are n items, then there are $n!$ (factorial of n) possible orderings (but see section 2.6.4 for further discussion of this).

As we see illustrated in the example above, the item that is retrieved *last* has *widest* scope. In general, if a translation NP'_i is retrieved after a translation NP'_j, then NP'_j will be within the scope of NP'_i.

2.6.4 Discussion

Treating scope ambiguities as a separate type of ambiguity, using Cooper storage, leads to an elegant solution that avoids the need for extra layers of syntactic representation or the positing of intuitively spurious lexical ambiguities. The solution is (arguably) compositional as, in all cases, the matrixes and stores of a constituent are determined by the matrixes and stores of the subconstituents of that constituent. In

the rest of this subsection, we discuss the following issues: the status of the logical translations; the elimination of redundant translations; the soundness and completeness of the solution; and the imposition of a preference order on the translations produced.

The status of the logical translations

The first point of concern is the status of the translations we are producing. In our discussion of semantic interpretation up to now, the semantic value of a natural language constituent was its model-theoretic denotation. Translating English into logic was not essential to interpreting the English phrases; it was only a convenient intermediate step in assigning them meanings (i.e. translating English expressions to expressions of logic seemed more perspicuous than persisting in the direct assignment of model-theoretic denotations to the English expressions). However, now that we have matrixes and stores, it is less clear whether the logic is just a convenience: can we assign model-theoretic denotations to English expressions and achieve the same effect as is achieved using matrixes and stores? In fact, the answer is that we can do this. We have no space to show this here, but we refer the reader to Cooper (1983), which is not only the original formulation of this approach but also directly assigns model-theoretic denotations without logic as a mediating level of representation. The main change (which, in fact, is a change that is needed to handle any kind of ambiguity, not just scope ambiguity) is to see semantic interpretation as a *relation* (a many-to-many mapping) between English expressions and denotations, rather than as a *function* (a many-to-one mapping). It is, therefore, the case that the logic continues to be only a convenient mediating representation. Note, however, that its status comes under renewed attack when we introduce modifiers, such as PPs, that contain quantified NPs. This is discussed in the context of soundness and completeness below.

Logically redundant translations

Since the translation of names, such as *"Macbeth"*, are not stored, the Cooper storage approach, unlike the lexical ambiguity solution, does not produce multiple (identical) translations for sentences containing transitive verbs but containing only one or no quantified NPs (such as *"Macbeth killed every witch"*).

However, there are other reasons why both approaches can produce logically redundant translations. By this we mean that they can produce translations of a sentence that are logically equivalent. For example, on the following sentence:

(16) *"Every soldier killed every witch."*

both approaches will produce two readings:

$$\forall x_1(soldier1(x_1) \supset \forall x_5(witch1(x_5) \supset killed1(x_1, x_5)))$$

$$\forall x_5(witch1(x_5) \supset \forall x_1(soldier1(x_1) \supset killed1(x_1, x_5)))$$

These two readings are truth-conditionally equivalent (whenever one is true so is the other); one is, therefore, redundant. From a theoretical point of view, no harm is being done; from a *computational* point of view, the extra work is undesirable.

Checking for equivalence, so that redundant translations can be thrown away, can be done cheaply enough for a handful of the most common cases, but carrying out such a check for the *general* case can be computationally very expensive (and may not, in general, terminate).

And, there is another source of redundancy in the Cooper storage approach: in its original formulation, due to the optionality of the retrieval operation, there can be multiple derivations of the same translation. In our extended example (figure 2.10 and table 2.2), we ignored this optionality: both the subject and object NP translations were stored, and then the two different orders in which this pair of stored translations could be retrieved at the root node gave the two readings. But this ignores at least three other possibilities: (i) we can retrieve the the object NP translation immediately (at node G) but choose to retrieve the subject NP translation at the root; when the subject NP translation is eventually retrieved, it will fall within the scope of the object NP's quantifier; (ii) we can choose to retrieve the subject NP translation immediately (at node B) but choose to retrieve the object NP translation at the root; when the object NP translation is eventually retrieved, it will fall within the scope of the subject NP's quantifier; and (iii) we can choose to retrieve both NP translations immediately (nodes B and G); the resulting reading for our grammar will have the object NP translation within the scope of the subject NP's quantifier. From these five derivations (the three just listed and the two exemplified earlier), only two logically different translations are produced. Again, from a theoretical point of view, no harm is being done. But, from a computational point of view, there will be a lot of undesirable effort. In addition to the effort of throwing away duplicate readings, the effort of finding each derivation is high: if semantic translation and storage are interleaved with parsing, then to compute an alternative derivation (using a different pattern of retrieval decisions) requires *re-parsing* the affected parts of the string.

The way to avoid finding multiple derivations of the same translation might appear obvious: always retrieve quantified NP translations at the root node, and nowhere else, computing the different translations by permuting this full store. Unfortunately, this simple solution is neither sound (some of the readings it produces are illegitimate) nor complete (some of the readings it should produce are not produced); we discuss these issues in the next subsection.

In practice, to overcome the problem, an alternative implementation of Cooper storage is usually used (although, for the same reasons that we discuss in the next subsection, this too requires some care if soundness and completeness are to be achieved). By this alternative, quantified NP translations are not ever placed in a separate store during the process of semantic translation. But neither are they used to produce scoped translations during semantic translation. Instead, NP translations are treated as special terms, often called 'qterms', that can form the arguments to predicates. The translation of sentence (15), *"Every soldier killed some witch"*, might then be, e.g.,

$$killed1(\quad qterm(\lambda P_1[\forall x_1(soldier1(x_1) \supset P_1(x_1))]),\\ qterm(\lambda P_5[\exists x_5(witch1(x_5) \wedge P_5(x_5))]) \quad)$$

The scoping algorithm works by recursively traversing such a translation, applying some qterms to the matrix and putting the others into an *ordered* store; decisions

about what to apply and what to store (and in what order) are made nondeterministically; subsequent traversals make different patterns of decisions, thus generating multiple readings. Scoping is thus done *after* parsing and semantic translation, with the immediate advantage that finding alternative readings only requires re-traversing the translation; it does not require any re-parsing. But, using ordered stores is the key to avoiding multiple derivations of the same translation: we can ensure that the only stores that get built are those that will give different readings from the readings obtained by immediate application of the qterm to the matrix. The further reading section points to full presentations of these algorithms.

Perhaps the 'ideal' solution is to abandon the requirement that all readings be found, and require instead that only one reading (the most preferred one) is computed. In this case, only one pattern of retrieval decisions need be made, this pattern being determined using pragmatic and non-linguistic knowledge. Unfortunately, this proposal asks too much of our current understanding of the factors that determine a contextually-preferred reading.

Soundness and completeness

The algorithm we have presented is not sound (it produces incorrect readings) and is not complete (it does not produce all readings). The introduction of optionality of retrieval (or its equivalent once translations have all been computed) remedies the incompleteness. The unsoundness arises when we introduce into the grammar rules that allow NP modifiers and complements that contain other NPs. The example of this that we have introduced in this chapter is modification of an Nbar by a PP.

As we have seen, if a sentence contains n quantified noun phrases, then our simple approach produces $n!$ readings. However, a sentence such as sentence (17)

(17) *"Every soldier in a battalion killed some witch."*

while it has three quantified NPs, has only five and not six (3!) readings. Disallowed, for example, is the following reading in which the translation of *"every soldier"* has widest scope, that of *"some witch"* has next widest and *"a battalion"* has narrowest scope (i.e. the reading in which there is a different battalion not only for each soldier but also for all the witches):

$$\forall x((soldier1(x) \wedge in1(x,y)) \supset \exists z(witch1(z) \wedge \exists y(battalion1(y) \wedge killed1(x,z))))$$

You will note that the first occurrence of variable y is free: that this variable is not getting bound properly renders this reading incorrect.

Similarly, sentence (18)

(18) *"Some soldier in every company in most battalions killed a few witches of each coven."*

while it has five quantified NPs, has forty-two, not 120 (5!), readings. Illegitimate translations will again be ones with free variables in them.

Sentences (17) and (18) contain NPs that are embedded within other NPs (in this case by virtue of being within a PP that is part of the NP, although NPs within relative clauses cause similar problems). For example, in *"every soldier in a battalion"*, the

NP *"a battalion"* is embedded within the larger NP. When an 'embedded' quantified NP translation is stored and not immediately retrieved again, i.e. it gets 'lifted' or 'raised' over its dominating NP, this 'embedded' NP's translation must receive wider scope than the NP in which it was embedded. Given what we know about how the order of application affects scope (the later a translation is retrieved, the wider its scope), it follows that the 'raised' NP translation must be retrieved after that of the NP that dominates it. As we have seen, if this does not happen, we are left with a translation in which binding of variables does not happen properly: an occurrence of a variable is not 'captured' by the quantifier and so remains free in the final scoped translation.

The obvious way of eliminating these spurious readings is to throw away those which have free variables in them. However, this makes the concern we raised earlier about the status of the logic translations more acute: the logic translations, which were once a mere convenience, will have become indispensable to a proper treatment, i.e. you can only get your final set of readings for the sentence by considering a syntactic property of the translations.

However, there is an alternative to checking this syntactic property. We can add more structure to the contents of the stores, this extra structuring being sufficient to prevent the illegitimate retrieval orders. Effectively, items in store will no longer comprise two components (the NP translation and a free variable), but will comprise three components: the NP translation, the free variable, and the store of the constituents of the NP (i.e. the union of the Det and nominal constituents). Hence, store contents created for subconstituents of NPs (particularly, e.g., for modifiers and complements that contain other NPs) will be nested within the item stored for the parent NP.

Specifically, for a rule such as $\mathsf{NP} \rightarrow \mathsf{Det\ Nbar}$, we get the following:

	English syntax	Matrix rule	Store rule
•	$\mathsf{NP} \rightarrow \mathsf{Det\ Nbar}$;	$\mathsf{NP}' = \lambda P[P(z)]$	$\mathsf{NP}'' = \{\langle \mathsf{Det}'(\mathsf{Nbar}'),\ z,\ \mathsf{Det}'' \cup \mathsf{Nbar}''\ \rangle\}$

When retrieving items, we must now remember to 'unpack' the nested stores. Specifically, if the store S'' contains an item $\langle \mathsf{NP}',\ z,\ \mathit{NestedStore} \rangle$, to retrieve this item, we compute $\mathsf{NP}'(\lambda z[\mathsf{S}'])$ as before, but the store that remains after doing so is now:

$$(\mathsf{S}'' - \{\langle \mathsf{NP}',\ z,\ \mathit{NestedStore} \rangle\}) \cup \mathit{NestedStore}$$

(where $-$ is the symbol for set difference), i.e. it is the original store less the retrieved item but with the nested store now augmenting the stored items; all these items are now available themselves for retrieval. You can see that this enforces the constraint that the 'raised' NP translation is retrieved after (and, therefore, receives wider scope than) that of the NP that dominates it.

In practical implementations, nested Cooper storage is rarely used. It is more common to ignore theoretical concerns and simply compute all the readings that result from all the orderings and then throw away those that still have free variables in them.

A remaining issue that affects soundness and completeness is that of the quantification of event variables. Consider the following sentence:

(19) *"Every soldier died."*

We must consider whether this has two readings, as follows:

$$\forall x(soldier(x) \supset \exists e\, died1(e, x))$$

$$\exists e(\forall x(soldier1(x) \supset died1(e, x)))$$

In the first reading, there is a possibly different dying event for every soldier; in the second reading, there is one dying event, in which every soldier died (some form of mass suicide perhaps). It is the first of these readings that would be produced by the semantic translation rule that we gave earlier: $\mathsf{S}' = \mathsf{NP}'(\lambda x[\exists e((\mathsf{VP}'(x))(e))])$. We can easily enough obtain the other reading by using an alternative rule $\mathsf{S}' = \exists e(\mathsf{NP}'(\lambda x[(\mathsf{VP}'(x))(e)]))$.

However, we need to consider how quantification of the event variable interacts with the quantification arising from the occurrence of multiple quantified NPs in the sentence. For example, given that the following sentence

(20) *"Every soldier killed many witches."*

has three sources of quantification (the two NPs and the existentially quantified event variable), we must ask: does this sentence have six (= 3!) readings? We leave this question unanswered and will not sketch the modification to Cooper storage that might compute these scopings.

Preferred quantifier scopes

Up to now, we have concentrated on the computation of all legitimate readings of scope ambiguous sentences. In this subsection, we very briefly discuss preferred readings of scope ambiguous sentences. The presentation of this material is heavily influenced by Moran (1988).

As with all disambiguation, a major factor is the use of pragmatic (i.e. contextual) and non-linguistic (i.e. domain and world) knowledge. For example, in the following sentence, structural preferences (see below) predict one reading, but pragmatic and non-linguistic knowledge overrides this structural preference (or, at least, bring it into question).

(21) *"Macbeth killed every soldier that defended a town."*

In (21), the structural preferences (below) probably predict a reading in which the universal outscopes the existential. However, it is common (world) knowledge that many soldiers defend some town or another during their careers. By the pragmatic principle that an utterance should be neither less nor more informative than is required by the purposes of the discourse, we can conclude that if *"a town"* receives narrow scope relative to *"every soldier"* and therefore tells us little that our world knowledge does not already let us conclude, then it would contribute too little to the discourse to justify it having been uttered. And so, we find ourselves entertaining the alternative reading, in which the existential outscopes the universal: Macbeth killed every soldier who defended a particular town.[27]

[27]Sometimes the structural preferences can be very strong. This presumably is what accounts for the punning nature of the sentence *"A woman in the U.S. gives birth every five minutes"*).

Of course, it is probably most normal for structural preferences and pragmatic and non-linguistic knowledge to make concurring predictions.

We will not consider how pragmatic and non-linguistic knowledge are brought to bear. This is a difficult question leading into wider issues in knowledge representation and reasoning; how to do it efficiently and integrate it with linguistic processing remain largely open questions. Instead, we present some 'structural' preferences that might operate by default in the absence of preferences from pragmatic and non-linguistic knowledge. It should be stressed that these are heuristics (rules of thumb) and not absolute constraints on quantifier scoping.

A simplification that makes presentation of this material more straightforward is to split the heuristics into two: those that give a preference ordering to a store of NP translations, and those that guide the decision about whether the translation of an NP that is embedded within some other NP should be 'raised' or not.[28]

Given a store of NP translations, we want to carry out a pairwise comparison of the translations to determine an ordering such that if NP translations are retrieved and applied in accordance with that ordering the readings will be produced in order of decreasing preference. We present three main criteria by which translation NP'_i might be preferred to outscope translation NP'_j:

- Determiner strength. English determiners have been assigned 'strengths', and it is claimed that there is a preference for the translation of the NP with the stronger determiner to outscope the translation of the determiner with the weaker determiner. Proposed strengths include:

$$\textit{"each"} \;>\; \begin{array}{c}\textit{"wh"}\text{-determiners}\\ \text{(e.g. } \textit{"who"},\\ \textit{"what"},\textit{which"})\end{array} \;>\; \textit{"every"} \;>\; \begin{array}{c}\textit{"all"},\\ \textit{"some"},\\ \textit{"some"},\ \textit{"a"}\end{array}$$

- Logical strength. There is a preference for the logically weakest interpretation. One translation is logically weaker than another if it is true in more models than the other. Thus, of the two translations:

$$\forall x(\mathit{soldier}1(x) \supset \exists y(\mathit{witch}1(y) \wedge \mathit{killed}1(x,y)))$$

$$\exists y(\mathit{witch}1(y) \wedge \forall x(\mathit{soldier}1(x) \supset \mathit{killed}1(x,y)))$$

  the $\forall\exists$ translations is preferred to the $\exists\forall$ translation. The $\forall\exists$ translation is true in more models (indeed, it is true in every model in which $\exists\forall$ is true and further models too).

- Surface order. Scopings that preserve the surface order of the corresponding determiners in the sentence are preferred. There is often the suggestion that this heuristic is stronger than the previous two. Consider, for example, sentences (22) and (23):

  (22) *"Every soldier killed a witch."*

  (23) *"A witch was killed by every soldier."*

[28] As per footnote 25, we continue to ignore other sources of scope ambiguity such as verbs that translate as modal or intensional operators. But, we note that heuristics have been adduced to account for the likelihood that an NP translation from the complement of such a verb will outscope the operator. See Moran (1988) for some of these.

> By determiner strength and logical strength, the preferred readings of both sentences are the same, with the universal outscoping the existential. But, our intuition suggests a preference for the other reading for sentence (23), and this is predicted by having the surface order preference win out.

Other, more idiosyncratic heuristics have been proposed; for example, the length of the NP might play a part: long indefinite NPs tend to take wide scope.

Implementing the above heuristics might require that additional elements be stored along with NP translations, e.g. the determiner or some numerical representation of is strength to allow for implementation of the determiner strength heuristic, and the surface position of the NP in the original input string to allow for implementation of the surface order heuristic.

As we have seen, an NP that is embedded within a complement or modifier of some other NP may be 'raised', in which case it must receive wider scope than the NP translation of the NP that dominates it. Heuristics have been proposed for determining when 'raising' is preferred.

Some argue that the preference is always for 'raising' the NP. However, it has also been argued that some constituents, referred to as **scope islands**, strongly prefer that translations of embedded NPs not be 'raised'. Relative clauses are putative examples of scope islands. Perhaps most appropriate is a ranking: NPs from PP modifiers are more likely to be 'raised' than those from reduced relative clauses and these, in turn, are more likely to be 'raised' than those from (full) relative clauses. For the following examples,

(24) *"A soldier near every witch laughed."*

(25) *"A soldier liked by every witch laughed."*

(26) *"A soldier who was liked by every witch laughed."*

according to the preferences, it is most likely that the universal outscopes the existential in (24) and least likely in (26), with (25) being more neutral.

Another very strong preference is not to 'raise' an NP across more than one major clause boundary. What this means is that a stored NP ought to have been retrieved at or below its closest dominating S node, and *must* have been retrieved at or below the second closest S node: there is a preference for discharge at S, but it if this is not done, it ought to be done at the next S up, and should not be left to Ss higher than this.

Finally, note that determiner strength may change when an NP is 'raised'. Specifically, less deeply embedded ones tend to outscope more deeply embedded ones, even where this contradicts the normal strength relation.

A discussion of preferred readings in particular and scope ambiguity in general would not be complete without mentioning an alternative position. Determining an intended quantifier scoping is extremely difficult. There is an opinion that in at least some cases even human discourse participants do not disambiguate scope ambiguous sentences. Consider:[29]

[29]This example is taken from Mitchell Marcus's commentary on Pulman (1987).

(27) *"A simple pharmacological test showed each of the drugs to be psycho-active."*

You understand this sentence well enough. But you would find it difficult if not impossible to answer the following question: was the same test used for every drug or was a possibly different test used for every drug? For this you would need to disambiguate the sentence's scope ambiguity. You would have no trouble, on the other hand, in answering the following question: do you want to feed the drugs to your dog? The answer depends on your attitude to your dog. Similarly, it would be possible to translate (27) into another natural language without resolving the ambiguity, provided that the target language exhibits similar scope ambiguities to those found in English, so that the ambiguity could be preserved in the translation. This suggests that full disambiguation is not needed for many language tasks (such as answering certain questions, performing certain translations): we can work with a single, ambiguous representation. This raises the challenge of developing meaning representation languages (logics) that can preserve certain natural language ambiguities, still have a suitable model-theoretic interpretation and allow some reasoning to be carried out (particularly subsequent disambiguation should more information come to light). These languages offer **underspecified representations**. There is a growing body of literature that addresses the challenge of defining such representations (see the further reading section). Their use will be mentioned several times in the next chapter.

2.7 Summary

- Quantified noun phrases comprise a determiner and a nominal expression, the determiner quantifying the number of individuals being referred to, and the nominal expression restricting the quantification to only those individuals which the nominal denotes.
- A logic, whose syntax and semantics includes quantifiers and connectives, can be defined as a suitable target in a translation from English that contains quantified NPs. An appropriate translation of these NPs and their constituents requires that the logic be a higher-order one.
- Quantified NPs denote sets of sets of individuals. This semantic type can also be given to NPs that comprise simple names.
- Generalised quantifier theory offers a treatment of a wide range of determiners and the statement of a number of linguistic universals.
- In type-raising, an argument expression is changed into a function-denoting expression which takes the former functor as argument. An amount of type-raising is necessary to incorporate quantified NPs into the semantic translation rules of the grammar. A particularly uniform approach is to type-raise all verbal categories so that they can continue to take NP denotations as their arguments.

- An event semantics, in which the ontology of the logic is extended by events, which are the denotations of individual constants and variables, gives a simple, first-order treatment, with appropriate entailment relationships between different sentence translations, for some cases of verb phrase modification. This semantics can also easily incorporate a treatment of thematic roles, which show the roles that different objects play in the events described by sentences.

- Where a sentence contains multiple quantified NPs, it is likely to be scope ambiguous. Scope ambiguities can be seen as forms of structural or semantic lexical ambiguity or as a separate type of ambiguity.

- Cooper storage offers an elegant solution to the computation of the multiple readings of scope ambiguous sentences: NP translations are put 'on ice' when originally formed and then applied at some later point to the translation of the S, the order of application determining the scoping of the resulting translation.

2.8 Further reading

The origins of the material covered by this chapter, as with those of the material covered by the previous chapter, are in the work of Richard Montague, collected in Montague (1974). Some of the best textbook treatments of the semantics of logics that include quantifiers (and therefore need to make use of value assignments) are the textbooks that also contain material on the semantics of quantified NPs, e.g. Dowty et al. (1981), Chierchia and McConnell-Ginet (1990), Partee et al. (1990), Gamut vols. I and II (1991) and Cann (1993).

Generalised quantifiers receive introductory treatments in Barwise and Cooper (1981), Cann (1993), Chierchia and McConnell-Ginet (1990), Partee et al. (1990) and van Benthem (1986).

The determiners *"the"* and *"a"* require special mention in this section. NPs in which the determiner is *"the"* are among the NPs referred to as **definite NPs**; those in which the determiner is *"a"* or *"an"* are among the NPs referred to as **indefinite NPs**. Both of these have long posed special problems. It is common to translate them using the existential quantifier:

English	Logic
"A king died"	$\exists x(king1(x) \wedge died1(x))$
"The king died"	$\exists x(king1(x) \wedge \forall y(king1(y) \supset x = y) \wedge died1(x))$

The extra conjunct, $\forall y(king1(y) \supset x = y)$, constrains the sentence to be true when there is only one king (it says that if anything else is a king, then it must be the same as x), which is commonly held to be a part of the meaning of definite (singular) NPs.[30]

[30]Note though that some contextual-dependence is needed: there need not be only one king in the whole universe, just one contextually salient king; and, note also that there are many researchers who would regard this uniqueness constraint as pragmatic in nature, and therefore would not include it in the semantic translation above.

The issues that surround the semantic and pragmatic treatment of definite and indefinite NPs and related constructions are manifold and complex. The two main approaches are associated with Russell (e.g. 1905) and Strawson (e.g. 1950). The discussion is part of a wider one on the notion of 'presupposition'. The following references cite computational treatments of presupposition (and hence of definite NPs); these will refer you to other sources in the linguistics and philosophy literature: Bridge (1991), Gunji (1981), Mercer (1987), and Weischedel (1979). Discourse Representation Theory (DRT) is a model-theoretic semantics that differs from the one presented in this book whose development has been partly driven by an attempt to give a satisfactory account of the meaning of definite and indefinite NPs, see, e.g., Kamp (1981).

We presented four solutions to the problems of incorporating quantified NPs into a grammar in which they appear both as subject and object NPs. Solution (1) ('rearranging' the translation of the transitive verb) is presented in Prolog by Pereira and Shieber (1987); solution (2) (type-raising the object NP) is described in van Benthem (1986) with reference to Geach (1972); solution (3) (type-raising transitive verbs) is Montague's solution and is presented in textbook form in Cann (1993); solution (4) is the one adopted in Generalised Phrase Structure Grammar (Gazdar et al. 1985).

Much more could have been said on the subject of type-raising and type-shifting in general. A discussion of a family of NP types (type $\mathbf{e}$ for 'referential' uses, type $\langle \mathbf{e}, \mathbf{t} \rangle$ for 'predicative' uses and type $\langle \langle \mathbf{e}, \mathbf{t} \rangle, \mathbf{t} \rangle$ for 'quantified' uses) is given by Partee (1986). Other references include Partee and Rooth (1983) and van Benthem (1986).

This still leaves many issues in the semantics of NPs that we have overlooked. For example, we have said nothing of bare nominals, i.e. nominal expressions without determiners. These include singular mass nouns (e.g. *"Tea is nice with lemon"*) and plural count nouns (e.g. *"Dogs attack when provoked"*); for these, see, e.g., Bunt (1985), Hoeksema (1983), Link (1983), Pelletier (1984), Scha (1981), Strzalkowski and Cercone (1989) and van Eijk (1983). Our treatment of modifiers (adjectives and PPs) is naïve and incomplete (e.g. we have excluded discussion of possessives, relative clauses, nominal compounds, etc.). Broader, slightly more sophisticated computational treatments can be found in Alshawi (1992).

The idea of event semantics is introduced by Davidson (1967) and is comprehensively covered in Parsons (1990). See also the papers, including one by Parsons, in LePore and McLaughlin (1985). A neat statement of the compositional construction of logical translations in event semantics, including some interesting treatments of adverbial modification, is given by Pulman (1989). The issues raised are situated in wider discussions of ontology and representation in, e.g., Moore (1981) and Hobbs (1985). Topics in the interpretation of PPs as VP modifiers can be found in Cresswell (1985). Thematic roles were proposed by Fillmore (1968); a catalogue of roles suitable for NLP purposes is given in Boguraev and Spärck Jones (1987).

The best of the early computational work on scope ambiguity is that of Woods (1978). Polystratal theories of grammar and the solution they offer to the problem of computing multiple readings for scope ambiguous sentences are described in Chierchia and McConnell-Ginet (1990) and May (1985). Cooper (1983) presents Cooper storage in full (though the presentation does not mediate assignment of denotations with logic). Hobbs and Shieber (1987) give sound and complete but efficient programs in Prolog and Lisp based on Cooper storage. A simpler version of the algorithm is given,

in Prolog, in Pereira and Shieber (1987), but this version does not prevent spurious readings from being produced for NPs embedded in the complements and modifiers of other NPs. The style of Cooper storage described in this chapter is akin to methods used by Pollack and Pereira (1988), and Pereira and Pollack (1991), and in the grammar theory known as Head-driven Phrase-Structure Grammar (HPSG), Pollard and Sag (1994). Nested storage is described in Keller (1988). Pereira (1990) proposes that spurious readings can be ruled out by constraints on function application and abstraction (see also König (1991)).

The best account of the preferences that operate in disambiguating scope ambiguous sentences is Moran (1988), a version of which appears as a chapter in Alshawi (1992). See also Hurum (1988).

As we mentioned in the body of the chapter, there is a growing amount of work on meaning representation languages that preserve ambiguities. A sample of this literature includes Hobbs (1983), Bunt (1984), Harper (1992), Alshawi (1990), Alshawi & Crouch (1992), and Reyle (1993), (1995).

Chapter 3

References

We use the following abbreviations in what follows:

AAAI: Annual Conference of the American Association for Artificial Intelligence;

ACL: Meeting of the Association for Computational Linguistics;

ACM: Association for Computing Machinery;

COLING: International Conference on Computational Linguistics; and

IJCAI: International Joint Conference on Artificial Intelligence.

Abramson, H., and Dahl, V. 1989: *Logic Grammars.* Springer-Verlag.

Aho, A.V. and Johnson, S. 1974: LR parsing. *Computing Surveys,* 6(2), 99-124.

Aho, A.V., Sethi, R., and Ullman, J.D. 1986: *Compilers: Principles, Techniques and Tools.* Addison-Wesley.

Aho, A.V. and Ullman, J.D. 1977: *Principles of Compiler Design.* Addison-Wesley.

Allen, J. 1982: Recognizing Intentions from Natural Language Utterances. In M.Brady (ed.), *Computational Models of Discourse,* MIT Press, 107-66.

Allen, J. 1994: *Natural Language Understanding (2nd edn.).* Benjamin/ Cummings.

Allen, J.F. and Perrault, C.R. 1980: Analyzing Intention in Utterances. *Artificial Intelligence,* 15, 143-78.

Allwood, J., Andersson, L.-G. and Dahl, Ö. 1977: *Logic in Linguistics.* Cambridge University Press.

Alshawi, H. 1990: Resolving Quasi Logical Form. *Computational Linguistics,* 16(3), 133-44.

Alshawi, H. (ed.) 1992: *The Core Language Engine.* The MIT Press.

Alshawi, H. and Crouch, R. 1994: Monotonic Semantic Interpretation. *Proceedings of Thirty-second ACL,* 32-9.

Altmann, G. 1985: The resolution of local syntactic ambiguity by the human sentence processing mechanism. *Proceedings of Second European ACL,* 123-7.

Altmann, G. 1987: Modularity and Interaction in Sentence Processing. In J.L.Garfield (ed.), *Modularity in Knowledge Representation and Natural-Language Understanding,* MIT Press, 249-57.

Androutsopoulos, I. Ritchie, G.D and Thanisch, P. 1994: *Natural Language Interfaces to Databases — An Introduction.* Research Paper No. 709, Department of Artificial Intelligence, University of Edinburgh.

Bach, E. 1974: *Syntactic Theory.* Holt, Rinehart and Winston.

Bach, E. 1976: An Extension of Classical Transformational Grammar. In R.Saenz (ed.), *Problems in Linguistic Metatheory: Proceedings of the 1976 Conference at Michigan State University,* Department of Linguistics, Michigan State University, 183-244.

Ballard, B.W., Lusth, J.C. and Tinkham, N.L. 1984: LFC-1: A Transportable Natural Language Processor for Office Environments. *ACM Transactions on Office Information Systems,* 2(1), 1-25.

Ballard, B.W. and Stumberger, D.E. 1986: Semantic Acquisition in TELI: A Transportable User-Customized Natural Language Processor. *Proceedings of Twentieth ACL,* 20-9.

Ballard, B.W. and Tinkham, N.L. 1984: A Phrase-Structured Grammatical Framework for Transportable Natural Language Processing. *Computational Linguistics,* 10(2), 81-96.

Barton, E.G. Jr., Berwick, R.C. and Ristad, E.S. 1987: *Computational Complexity and Natural Language.* MIT Press.

Barwise, J. and Cooper, R. 1981: Generalized Quantifiers and Natural Language. *Linguistics and Philosophy,* 4, 159-219.

Barwise, J. and Perry, J.R. 1983: *Situations and attitudes.* MIT Press.

Bates, J. and Lavie, A. 1991: *Recognising Substrings of LR(k) Languages in Linear Time.* Technical Report No. CMU-CS-91-188, Department of Computer Science, Carnegie-Mellon University.

Bates, M., Moser, M.G. and Stallard, D. 1986: The IRUS Transportable Natural Language Database Interface. In L.Kerschberg (ed.), *Expert Database Systems (Proceedings from the First International Workshop),* Benjamin/Cummings, 617-30.

Bear, J. and Hobbs, J.R. 1988: Localizing Expression of Ambiguity. *Proceedings of Second Conference on Applied NLP,* 235-41.

Benthem, J.F.A.K. van, and Meulen, A.G.B. ter 1996: *Handbook of Logic and Language.* Elsevier.

Bermudez, M.E. and Schimpf, K.M. 1986: A Practical Arbitrary Look-ahead LR Parsing Technique. *ACM SIGPLAN Notices,* 21(7), 136-44.

Berwick, R.C., Abney, S.P. and Tenny, C. (eds.) 1991: *Principle-Based Parsing: Computation and Psycholinguistics.* Kluwer Academic.

Berwick, R.C. and Weinberg, A.S. 1984: *The Grammatical Basis of Linguistic Performance: Language Use and Acquisition.* MIT Press.

Black, W.J. 1987: Acquisition of Conceptual Data Models from Natural Language Descriptions. *Proceedings of Third European ACL,* 241-8.

Bobrow, D.G., Kaplan, R.M., Kay, M., Norman, D.A., Thompson, H. and Winograd, T. 1977: GUS, A Frame-Driven Dialog System. *Artificial Intelligence,* 8, 155-73.

Bobrow, R.J. and Webber, B.-L. 1980: Knowledge Representation for Syntactic/Semantic Processing. *Proceedings of First AAAI,* 316-23.

Boguraev, B.K. and Spärck Jones, K. 1981: A General Semantic Analyser for Data Base Access. *Proceedings of Seventh IJCAI,* 443-5.

Boguraev, B.K. and Spärck Jones, K. 1983: How to Drive a Front-End using General Semantic Information. *Proceedings of First Applied ACL,* 81-8.

Boguraev, B.K. and Spärck Jones, K. 1985: *A Framework for Inference in Natural Language Front Ends to Databases.* Technical Report No. 64, University of Cambridge Computer Laboratory.

Boguraev, B.K. and Spärck Jones, K. 1987: *Material Concerning a Study of Cases.* Technical Report No. 118, University of Cambridge Computer Laboratory.

Bridge, D.G. 1991: *Computing Presuppositions in an Incremental Natural Language Processing System.* Ph.D Dissertation. Published as Technical Report No. 237, University of Cambridge Computer Laboratory.

Briscoe, E.J. 1983: Determinism and its implementation in Parsifal. In K. Spärck Jones and Y. Wilks (eds.), *Automatic Natural Language Parsing,* Ellis Horwood, 61-8.

Briscoe, E.J. 1987: *Modelling Human Speech Comprehension: A Computational Approach.* Ellis Horwood.

Briscoe, E.J. and Boguraev, B.K. 1984: Control strategies and theories of interaction in speech understanding systems. *Proceedings of Tenth COLING/Twenty-second ACL,* 259-66.

Briscoe, E. and Carroll, J. 1993: Generalised probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics,* 19(1), 25-59.

Brown, J.S. and Burton, R.R. 1975: Multiple Representations of Knowledge for Tutorial Reasoning. In D.G.Bobrow and A.Collins (eds.), *Representation and Understanding,* Academic Press, 311-49.

Brown, K. and Miller, J. 1991: *Syntax: A Linguistic Introduction to Sentence Structure (2nd edn.).* HarperCollins.

Bunt, H. 1984: The Resolution of Quantificational Ambiguity in the TENDUM System. *Proceedings of Tenth COLING/Twenty-second ACL,* 130-3.

Bunt, H.C. 1985: *Mass Terms and Model-Theoretic Semantics.* Cambridge University Press.

Burstall, R.M. and Darlington, J. 1977: A Transformation System for Developing Recursive Programs. *Journal of the ACM,* 24(1), 44-67.

Burton-Roberts, N. 1986: *Analysing Sentences: An Introduction to English Syntax.* Longman.

Cann, R. 1993: *Formal semantics: An introduction.* Cambridge University Press.

Capindale, R.A. and Crawford, R.G. 1990: Using a natural language interface with casual users. *International Journal of Man-Machine Studies,* 32, 341-62.

Carberry, S. 1986: Tracking User Goals in an Information-Seeking Environment. *Proceedings of Fourth AAAI,* 59-63.

Carberry, S. 1990: Incorporating Default Inference into Plan Recognition. *Proceedings of Eighth AAAI,* 471-8.

Carbonell, J.G. 1983: Discourse Pragmatics and Ellipsis Resolution in Task-Oriented Natural Language Interfaces. *Proceedings of Twenty-first ACL,* 164-8.

Carbonell, J.G., Boggs, W.M. and Mauldin, M.L. 1983: The XCALIBUR Project: A Natural Language Interface to Expert Systems. *Proceedings of Eighth IJCAI,* 653-6.

Carbonell, J.G and Hayes, P.J 1983: Recovery strategies for parsing extragrammatical language. *American Journal of Computational Linguistics,* 9, 123-46.

Carroll, J.A. 1993: *Practical Unification-Based Parsing of Natural Language.* Ph.D Dissertation. Published as Technical Report No. 314, University of Cambridge Computer Laboratory.

Carter, A.W. and Freiling, M.J. 1984: Simplifying Deterministic Parsing. *Proceedings of Tenth COLING/Twenty-second ACL,* 239-42.

Chang, C.-L. and Lee, R.C.-T. 1973: *Symbolic Logic and Mechanical Theorem-Proving.* Academic Press.

Chapman, N.P. 1987: *LR Parsing: Theory and Practice.* Cambridge University Press.

Charniak, E. and Goldman, R.P. 1993: A Bayesian Model of Plan Recognition. *Artificial Intelligence,* 64, 53-79.

Chierchia, G. and McConnell-Ginet, S. 1990: *Meaning and Grammar: An Introduction to Semantics.* MIT Press.

Chin, D.N. 1983: Knowledge Structures in UC, the UNIX Consultant. *Proceedings of Twenty-first ACL,* 159-63.

Chomsky, N. 1965: *Aspects of the Theory of Syntax.* MIT Press.

Chomsky, N. 1975: *Syntactic Structures.* Mouton.

Church, K. 1980: On parsing strategies and closure. *Proceedings of Eighteenth ACL,* 107-11.

Church, K. W. and Patil, R. 1982: Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table. *American Journal of Computational Linguistics,* 8, 139-49.

Clifford, J. 1988: Natural Language Querying of Historical Databases. *Computational Linguistics,* 14(4), 10-34.

Clocksin, W. and Mellish, C. 1984: *Programming in Prolog (2nd edn.).* Springer-Verlag.

Codd, E.F. 1979: Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems,* 4(4), 397-434.

Cohen, J. and Hickey, T.J. 1987: Parsing and Compiling using Prolog. *ACM Transactions on Programming Languages and Systems,* 9(2), 125-63.

Cohen, P.R. and Perrault, C.R. 1979: Elements of a Plan-Based Theory of Speech Acts. *Cognitive Science,* 3(3), 177-212.

Colmerauer, A. 1978: Metamorphosis Grammars. In L. Bolc (ed.), *Natural Language Communication with Computers,* Lecture Notes in Computer Science, vol. 63, Springer-Verlag, 133-89.

Cooper, R. 1983: *Quantification and syntactic theory.* D.Reidel.

Cooper, R., Crouch, R., van Eijck, J., Fox, C., van Genabith, J., Jaspers, J., Kamp, H., Pinkal, M., Poesio, M., Pulman, S. and Vestre, E. 1994a: *Describing the Approaches.* Deliverable D8, A Framework for Computational Semantics (FraCaS), LRE 62-051, Centre for Cognitive Science, University of Edinburgh.

Cooper, R., Crouch, R., van Eijck, J., Fox, C., van Genabith, J., Jaspers, J., Kamp, H., Pinkal, M., Poesio, M., Pulman, S. and Vestre, E. 1994a: *The State of the Art in Computational Semantics: Evaluating the Descriptive Capabilities of Semantic Theories.* Deliverable D9, A Framework for Computational Semantics (FraCaS), LRE 62-051, Centre for Cognitive Science, University of Edinburgh.

Copestake, A. and Spärck Jones, K. 1989: *Inference in a Natural Language Front End for Databases.* Technical Report No. 163, University of Cambridge Computer Laboratory.

Copestake, A. and Spärck Jones, K. 1990: Natural Language Interfaces to Databases. *Knowledge Engineering Review,* 5(4), 225-49.

Covington, M.A. 1994: *Natural Language Processing for Prolog Programmers.* Prentice Hall.

Cresswell, M.J. 1985: *Adverbial Modification: interval semantics and its rivals.* D.Reidel.

Crain, S. and Steedman, M. 1985: On not being led up the garden-path: the use of context by the psychological parser. In D.R. Dowty, L. Karttunen and A.M. Zwicky (eds.), *Natural Language Parsing,* Cambridge University Press, 320-58.

Crouch, R.S. and Pulman, S.G. 1993: Time and Modality in a Natural Language Interface to a Planning System. *Artificial Intelligence,* 63, 265-304.

Damerau, F.J. 1985: Problems and Some Solutions in Customization of Natural Language Database Front Ends. *ACM Transactions on Office Information Systems,* 3(2), 165-84.

Damerau, F.J., Joshi, A.K. and Kaplan, S.J. 1981: On the Utility of Computing Inferences in a Real Data Base Query Environment. *American Journal of Computational Linguistics,* 7(1), 43-5.

Davidson, D. 1967: The Logical Form of Action Sentences. In N. Rescher (ed.), *The Logic of Decision and Action,* University of Pittsburgh Press, 81-95.

Di Eugenio, B. and Webber, B. 1992: Plan Recognition in Understanding Instructions. *Proceedings of First Conference on A.I. Planning Systems,* 52-61.

Dowding, J., Moore, R., Andry, F. and Moran, D. 1994: Interleaving Syntax and Semantics in an Efficient Bottom-Up Parser. *Proceedings of Thirty-second ACL,* ???-???.

Dowty, D.R. D.Reidel: ,. Word Meaning and Montague Grammar. 1979

Dowty, D.R., Wall, R.E., and Peters, S. 1981: *Introduction to Montague Semantics.* D.Reidel.

Earley, J. 1970: An Efficient Context-Free Parsing Algorithm. *Communications of the ACM,* 13(2), 94-102.

Elmasri, R. and Navathe, S.B. 1994: *Fundamentals of Database Systems (2nd edn.).* Benjamin/Cummings.

Erbach, G. 1991: An Environment for Experimentation with Parsing Strategies. *Proceedings of Twelfth IJCAI,* 931-6.

Fass, D. 1991: met*: A Method for Discriminating Metonymy and Metaphor by Computer. *Computational Linguistics,* 17(1), 49-90.

Fillmore, C.J. 1968: The Case for Case. In E.Bach and R.Harms (eds.), *Universals in Linguistic Theory,* Holt, Rinehart and Winston, 1-90.

Flach, P.A. 1994: *Simply Logical: Intelligent Reasoning by Example.* John Wiley & Sons.

Fodor, J.D. and Inoue, A. 1994: The Diagnosis and Cure of Garden Paths. *Journal of Psycholinguistic Research,* 23(5), 407-34.

Ford, M., Bresnan, J. and Kaplan, R. 1981: A Competence-Based Theory of Syntactic Closure. In J. Bresnan (ed.), *The Mental Representation of Grammatical Relations,* MIT Press, 727-96.

Frazier, L. and Fodor, J.D. 1978: The sausage machine: a new two-stage parsing model. *Cognition,* 6(4), 291-325.

Frost, D.P. 1989: *The Design of a Natural Language Interface for Medical Expert Systems.* Ph.D. Thesis, University of London.

Gal, A. and Minker, J. 1988: Informative and Cooperative Answers in Databases Using Integrity Constraints. In V.Dahl and P.Saint-Dizier (eds.), *Natural Language Understanding and Logic Programming, II,* Elsevier, 277-300.

Gamut, L.T.F. 1991: *Logic, Language and Meaning, volume I: Introduction to Logic.* University of Chicago Press.

Gamut, L.T.F. 1991: *Logic, Language and Meaning, volume II: Intensional Logic and Logical Grammar.* University of Chicago Press.

Garey, M.R. and Johnson, D.S. 1979: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H.Freeman and Co.

Gazdar, G. 1979: *Pragmatics: Implicature, Presupposition, and Logical Form.* Academic Press.

Gazdar, G. 1983: NLs, CFLs and CF-PSGs. In K. Spärck Jones and Y. Wilks (eds.), *Automatic Natural Language Parsing,* Ellis Horwood, 81-93.

Gazdar, G. 1988: Applicability of Indexed Grammars to Natural Languages. In U. Reyle and C. Rohrer (eds.), *Natural Language Parsing and Linguistic Theories,* D.Reidel, 69-94.

Gazdar, Gerald, Klein, Ewan, Pullum, Geoffrey and Sag, Ivan 1985: *Generalized Phrase Structure Grammar.* Harvard University Press.

G., Gerald and Mellish, C.S. 1989: *Natural Language Processing In PROLOG: An Introduction to Computational Linguistics.* Addison-Wesley.

Gazdar, G. and Pullum, G.K. 1985: Computationally Relevant Properties of Natural Languages and their Grammars. *New Generation Computing,* 3, 273-306.

Geach, P. 1972: A Program for Syntax. In D.Davidson and G.Harman (eds.), *Semantics of Natural Language,* D.Reidel, 483-97.

Genesereth, M.R. and Nilsson, N.J. 1988: *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann.

Gerdemann, D. 1994: Parsing as Tree Traversal. *Proceedings of Nth??? COLING,* ???-???.

Green, G.M. 1989: *Pragmatics and natural language understanding.* Erlbaum.

Green, N. and Carberry, S. 1994: A Hybrid Reasoning Model for Indirect Answers. *Proceedings of Thirty-second ACL,* 58-65.

Grice, H.P. 1975: Logic and Conversation. In D.Davidson and G.Harman (eds.), *The Logic of Grammar,* Dickenson Publishing Co, 64-75.

Grice, H.P. 1978: Further Notes on Logic and Conversation. In P.Cole (ed.), *Syntax and Semantics, Volume 9: Pragmatics,* Academic Press, 113-27.

Grishman, R., Hirschmann, L., and Nhan, N.T. 1986: Discovery Procedures for Sublanguage Selectional Patterns: Initial Experiments. *Computational Linguistics,* 12(3), 205-15.

Grishman, R. and Chitrao, M. 1988: Evaluation of a Parallel Chart Parser. *Proceedings of Second Applied NLP,* 71-6.

Grosz, B.J. 1983: TEAM: A Transportable Natural-Language Interface System. *Proceedings of First Applied ACL,* 39-45.

Grosz, B.J., Appelt, D.E., Martin, P.A. and Pereira, F.C.N. 1987: TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces. *Artificial Intelligence,* 32, 173-243.

Grosz, B., Haas, N., Hendrix, G., Hobbs, J., Martin, P., Moore, R. and Robinson, J. 1982: Dialogic: A Core Natural-Language Processing System. *Proceedings of Ninth COLING,* 95-100.

Grosz, B.J., Spärck Jones, K., and Webber, B.-L. (eds.) 1986: *Readings in Natural Language Processing.* Morgan Kaufmann.

Grover, C., Briscoe, T., Carroll, J. and Boguraev, B. 1989: *The Alvey Natural Language Tools Grammar (Second Release).* Technical Report No. 162, University of Cambridge Computer Laboratory.

Gunji, T. 1981: *Towards a computational theory of pragmatics — discourse presupposition and implicature.* Ph.D Dissertation, Ohio State University.

Haddock, N.J. 1987: Incremental Interpretation and Combinatory Categorial Grammar. *Proceedings of Tenth IJCAI,* 661-3.

Haddock, N.J. 1989: Computational Models of Incremental Semantic Interpretation. *Language and Cognitive Processes,* 4(3/4), 337-68.

Harel, D. 1989: *The Science of Computing: Exploring the Nature and Power of Algorithms.* Addison-Wesley.

Harper, M.P. 1992: Ambiguous Noun Phrases in Logical Form. *Computational Linguistics,* 18(4), 419-61.

Haruno, M., Den, Y., Matsumoto, Y. and Nagao, M. 1993: Bidirectional Chart Generation of Natural Language Texts. *Proceedings of Eleventh AAAI,* 350-6.

Hayes, P.J. and Carbonell, J.G. 1981: Multi-Strategy Construction-Specific Parsing for Flexible Data Base Query and Update. *Proceedings of Seventh IJCAI,* 432-9.

Hendrix, G.G. 1977: Human Engineering for Applied Natural Language Processing. *Proceedings of Fifth IJCAI,* 183-91.

Hendrix, G.G. and Lewis, W.H. 1981: Transportable Natural-Language Interfaces to Databases. *Proceedings of Nineteenth ACL,* 159-65.

Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D. and Slocum, J. 1978: Developing a Natural Language Interface to Complex Data. *ACM Transactions on Database Systems,* 3(2), 105-47.

Hirschberg, J. 1984: Toward a Redefinition of Yes/No Questions. *Proceedings of Tenth COLING/Twenty-second ACL,* 48-51.

Hirschberg, J.B. 1985: *A Theory of Scalar Implicature.* Ph.D Thesis, University of Pennsylvania.

Hirschberg, J. 1986: Anticipating False Implicatures: Cooperative Response in Question-Answering Systems. In L.Kerschberg (ed.), *Expert Database Systems (Proceedings from the First International Workshop),* Benjamin/ Cummings, 631-8.

Hirst, G. 1987: *Semantic interpretation and the resolution of ambiguity.* Cambridge University Press.

Hobbs, J.R. 1983: An Improper Treatment of Quantification in Ordinary English. *Proceedings of Twenty-first ACL,* 57-63.

Hobbs, J.R. 1985: Ontological Promiscuity. *Proceedings of Twenty-third ACL,* 61-9.

Hobbs, J.R. and Martin, P. 1987: Local Pragmatics. *Proceedings of Tenth IJCAI,* 520-3.

Hobbs, J.R., Stickel, M., Martin, P. and Edwards, D. 1988: Interpretation as Abduction. *Proceedings of Twenty-sixth ACL,* 95-103.

Hobbs, J.R. and Shieber, S.M. 1987: An Algorithm for Generating Quantifier Scopings. *Computational Linguistics,* 13, 47-63.

Hobbs, J.R., Stickel, M.E., Appelt, D.E. and Martin, P. 1993: Interpretation as Abduction. *Artificial Intelligence,* 63, 69-142.

Hoeksema, J. 1983: Plurality and conjunction. In A.G.B. ter Meulen (ed.), *Studies in Model-theoretic Semantics,* Foris, 63-83.

Hoeppner, W., Christaller, T., Marburger, H., Morik, K., Nebel, B., O'Leary, M. and Wahlster, W. 1983: Beyond Domain-Independence: Experience with the Development of a German Language Access System to Highly Diverse Background Systems. *Proceedings of Eighth IJCAI,* 588-94.

Hopcroft, J.E. and Ullman, J.D. 1979: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

Horrocks, G. 1987: *Generative Grammar.* Longman.

Huck, G.J. and Ojeda, A.E. (eds.) 1987: *Syntax and Semantics, Volume 20: Discontinuous Constituency.* Academic Press.

Hurum, S. 1988: Handling Scope Ambiguities in English. *Proceedings of Second Applied NLP,* 58-65.

Huyck, C.R. and Lytinen, S.L. 1993: Efficient Heuristic Natural Language Parsing. *Proceedings of Eleventh AAAI,* 386-91.

Janssen, T.M.V. 1996: Compositionality. In J.F.A.K. van Benthem and A.G.G. ter Meulen (eds.), *Handbook of Logic and Language,* Elsevier, 417-73.

Johnson, M. 1989: The Computational Complexity of Tomita's Algorithm. *Proceedings of International Workshop on Parsing Technologies,* 203-8.

Joshi, A., Webber, B. and Weischedel, R.M. 1984: Preventing False Inferences. *Proceedings of Tenth COLING/Twenty-second ACL,* 134-8.

Kalita, J.K., Colbourn, M.J. and McCalla, G.I. 1984: A Response to the Need for Summary Responses. *Proceedings of Tenth COLING/Twenty-second ACL,* 432-6.

Kalita, J.K., Jones, M.L. and McCalla, G.I. 1986: Summarizing Natural Language Database Responses. *Computational Linguistics,* 12(2), 107-24.

Kamp, H. 1981: A theory of truth and semantic representation. In J.A.G. Groenendijk, T.M.V. Janssen and M.B.J. Stokhof (eds.), *Formal methods in the study of language,* Amsterdam: Mathematical Centre Tracts 135, 277-322.

Kamp, H. and Reyle, U. 1993: *From Discourse to Logic.* Kluwer.

Kaplan, R.M. 1973: A General Syntactic Processor. In R. Rustin (ed.), *Natural Language Processing,* Algorithmics Press, 193-241.

Kaplan, S.J. 1981: Appropriate Responses to Inappropriate Questions. In A.K.Joshi, B.L.Webber and I.A.Sag (eds.), *Elements of Discourse Understanding,* Cambridge University Press, 127-44.

Kaplan, S.J. 1982: Cooperative Responses from a Portable Natural Language Query System. *Artificial Intelligence,* 19, 165-87.

Kaplan, S.J. and Davidson, J. 1981: Interpreting Natural Language Database Updates. *Proceedings of Nineteenth ACL,* 139-41.

Kaplan, S.J., Mays, E. and Joshi, A.K. 1979: A Technique for Managing the Lexicon in a Natural Language Interface to a Changing Data Base. *Proceedings of Sixth IJCAI,* 463-5.

Kautz, H.A. 1991: A Formal Theory of Plan Recognition and its Implementation. In J.F.Allen, H.A.Kautz, R.N.Pelavin and J.D.Tenenberg, *Reasoning about Plans,* Morgan Kaufmann, 69-125.

Kautz, H.A. and Allen, J.F. 1986: Generalized Plan Recognition. *Proceedings of Fifth AAAI,* 32-7.

Kay, M. 1973: The MIND System. In R. Rustin (ed.), *Natural Language Processing,* Algorithmics Press, 155-88.

Kay, M. 1986: Algorithm Schemata and Data Structures in Syntactic Parsing. In B.J. Grosz, K. Spärck Jones, and B.L. Webber (eds.), *Readings in Natural Language Processing,* Morgan Kaufmann, 35-70.

Kay, M. 1989: Head-Driven Parsing. *Proceedings of International Workshop on Parsing Technologies,* 52-62.

Keller, W.R. 1988: Nested Cooper Storage: The Proper Treatment of Quantification in Ordinary Noun Phrases. In U. Reyle and C. Rohrer (eds.), *Natural Language Parsing and Linguistic Theories,* D.Reidel, 432-47.

Kent, W. 1978: *Data and Reality.* North-Holland.

Kimball, J. 1973: Seven principles of surface structure parsing in natural language. *Cognition,* 2(1), 15-47.

King, J. 1981: QUIST: A System for Semantic Query Optimization. *Proceedings of Seventh International Conference on Very Large Data Bases,* 510-7.

Kipps, J.R. 1989: Analysis of Tomita's Algorithm for General Context-Free Parsing. *Proceedings of International Workshop on Parsing Technologies,* 193-202.

Klein, E. and Sag, I. 1985: Type-driven translation. *Linguistics and Philosophy,* 8, 163-201.

Kobsa, A. and Wahlster, W. (eds.) 1988: *Special Issue on User Modelling.* Computational Linguistics, 14(3).

König, E. 1991: Incremental Syntactic and Semantic Processing. *Proceedings of Twelfth IJCAI,* 925-30.

Kukich, K. 1983: Design of a Knowledge-Based Report Generator. *Proceedings of Twenty-first ACL,* 145-50.

Kuno. S. 1965: The Predictive Analyzer and a Path Elimination Technique. *Communications of the ACM,* 8(7), 453-62.

Lakoff, G. and Johnson, M. 1980: *Metaphors We Live By.* University of Chicago Press.

LePore, E. and McLauglin, B. (eds.) 1985: *Actions and Events: Perspectives on the Philosophy of Donald Davidson.* Blackwell.

Levinson, S.C. 1983: *Pragmatics.* Cambridge University Press.

Lin, D. 1992: *Obvious Abduction.* Ph.D. Thesis, Department of Computer Science, University of Alberta.

Lin, D. 1993: Principle-based parsing without overgeneration. *Proceedings of Thirty-first ACL,* 112-20.

Lin, D. and Goebel, R. 1993: Context-Free Grammar Parsing by Message Passing. *Proceedings of PACLING-93,* 203-11.

Link, G. 1983: The logical analysis of plural and mass terms: a lattice-theoretic approach. In R. Bäuerle, C. Schwarze and A. von Stechow (eds.), *Meaning, Use, and Interpretation of Language,* de Gruyter, 302-23.

Litman, D.J. and Allen, J.F. 1984: A Plan Recognition Model for Clarification Subdialogues. *Proceedings of Tenth COLING/Twenty-second ACL,* 302-11.

Magerman, D. and Marcus, M. 1991: $\mathcal{P}$earl: A Probabilistic Chart Parser. *Proceedings of Second International Workshop on Parsing Technologies,* 193-9.

Magerman, D. and Weir, C. 1992: Efficiency, Robustness and Accuracy in $\mathcal{P}$icky Chart Parsing. *Proceedings of Thirtieth ACL,* 40-7.

Marcu, D. and Hirst, G. 1995: A Uniform Treatment of Pragmatic Inferences in Simple and Complex Utterances and Sequences of Utterances. *Proceedings of Thirty-third ACL,* 144-50.

Marcus, M. 1980: *A Theory of Syntactic Recognition for Natural Language.* MIT Press.

Marcus, M. 1981: A computational account of some constraints on language. In A. K. Joshi, B.L. Webber and I.A. Sag (eds.), *Elements of Discourse Understanding,* Cambridge University Press, 177-200.

Martin, P., Appelt, D. and Pereira, F. 1983: Transportability and Generality in a Natural-Language Interface System. *Proceedings of Eighth IJCAI,* 573-81.

Matsumoto, Y., Tanaka, H., Hirakawa, H., Miyoshi, H. and Yasukawa, H. 1983: BUP: A Bottom-Up Parser Embedded in Prolog. *New Generation Computing,* 1, 145-58.

May, R. 1985: *Logical form: its structure and derivation.* MIT Press.

Maybury, M.T. 1990: *Planning Multisentential English Text using Communicative Acts.* Technical Report RADC-TR-90-411, Rome Air Development Centre, Air Force Systems Command, Griffis Air Force Base, New York.

McCawley, J.D. 1981: *Everything That Linguists Have Always Wanted to Know about Logic But Were Ashamed to Ask.* Blackwell.

McCoy, K.F. 1984: Correcting Object-Related Misconceptions. *Proceedings of Tenth COLING/Twenty-second ACL,* 444-7.

McFadden, F.R. and Hoffer, J.A. 1994: *Modern Database Management (4th edn.).* Benjamin/Cummings.

McGee Wood, M. 1993: *Categorial Grammars.* Routledge.

McKeown, K.R. 1979: Paraphrasing using Given and New Information in a Question-Answer System. *Proceedings of Seventeenth ACL,* 67-72.

McKeown, K.R. 1985a: *Text Generation.* Cambridge University Press.

McKeown, K.R. 1985b: Discourse Strategies for Generating Natural-Language Text. *Artificial Intelligence,* 27, 1-41.

McRoy, S.W. and Hirst, G. 1995: The Repair of Speech Act Misunderstandings by Abductive Inference. *Computational Linguistics,* 21(4), 435-78.

Mellish, C.S. 1983: Incremental semantic interpretation in a modular parsing system. In K. Spärck Jones and Y. Wilks (eds.), *Automatic Natural Language Parsing,* Ellis Horwood, 148-55.

Mellish, C.S. 1985: *Computer Interpretation of Natural Language Descriptions.* Ellis Horwood.

Mellish, C.S. 1988: Implementing Systemic Classification by Unification. *Computational Linguistics,* 14(1), 40-51.

Mellish, C. 1989: Some Chart-Based Techniques for Parsing Ill-Formed Input. *Proceedings of Twenty-seventh ACL,* 102-9.

Mercer, R.E. 1987: *A Default Logic Approach to the Derivation of Natural Language Presuppositions.* Ph.D Dissertation. Published as Technical Report No. 87-35, University of Vancouver.

Mercer, R.E. and Rosenberg, R.S. 1984: Generating Corrective Answers by Computing Presuppositions of Answers, not of Questions or Mind *Your* P's, not Q's. *Proceedings of Canadian Society for Computational Studies of Intelligence,* 16-9.

Miller, P.H. 1999: *Strong generative capacity.* CSLI Publications.

Milne, R.W. 1982: Predicting garden-path sentences. *Cognitive Science,* 6, 349-73.

Milne, R.W. 1986: Resolving lexical ambiguity in a lexical parser. *Computational Linguistics,* 12(1), 1-12.

Milward, D. and Cooper, R. 1994: Incremental Interpretation: Applications, Theory, and Relationship to Dynamic Semantics. *Proceedings of Fifteenth COLING,* 748-54.

Mizoguchi, F. (ed.) 1991: *Prolog and its Applications: A Japanese Perspective.* Chapman and Hall.

Montague, R. 1974: *Formal philosophy.* Yale University Press.

Moore, R.C. 1981: Problems in Logical Form. *Proceedings of Nineteenth ACL,* 117-24.

Moore, R.C., Grosz, B.J., Petrick, S.R., Scha, R.J.H, Schwartz, S.P., Thompson, F.B. and Warren, D.H. 1982: Natural Language Access to Databases — Theoretical/Technical Issues (Panel Discussion). *Proceedings of Twentieth ACL,* 44-66.

Moran, D.B. 1988: Quantifier Scoping in the SRI Core Language Engine. *Proceedings of Twenty-sixth ACL,* 33-40.

Mueckstein, E.-M. M. 1983: Q-Trans: Query Translation into English. *Proceedings of Eighth IJCAI,* 660-2.

Nederhof, M.-J. 1994: *Linguistic Parsing and Program Transformations.* Ph.D. Dissertation, University of Nijmegen.

Nederhof, M.-J. and Satta, G. 1994: An Extended Theory of Head-Driven Parsing. *Proceedings of Thirty-second ACL,* ???-???.

Nilsson, U. 1986: AID: An Alternative Implementation of DCGs. *New Generation Computing,* 4, 383-99.

Nilsson, U. and Maluszyński, J. 1995: *Logic, Programming and Prolog (2nd edn.).* John Wiley & Sons.

Niv, M. 1994: A Psycholinguistically Motivated Parser for CCG. *Proceedings of Thirty-second ACL,* ???-???.

Nozohoor-Farshi, R. 1986: On formalizations of Marcus' parser. *Proceedings of Eleventh COLING,* 533-35.

Nozohoor-Farshi, R. 1987: Context-Freeness of the Language Accepted by Marcus' Parser. *Proceedings of Twenty-fifth ACL,* 117-22.

Nozohoor-Farshi, R. 1989: Handling of Ill-Designed Grammars in Tomita's Parsing Algorithm. *Proceedings of International Workshop on Parsing Technologies,* 182-92.

O'Keefe, R.A. 1990: *The Craft of Prolog.* MIT Press.

Osborne, M. 1990: *Corpus parsing.* Unpublished M.Phil. dissertation, Computer Laboratory University of Cambridge.

Parsons, T. 1990: *Events in the Semantics of English: A Study in Subatomic Semantics.* MIT Press.

Partee, B.H. 1986: Noun Phrase Interpretation and Type-Shifting Principles. In J. Groenendijk, D. de Jongh and M. Stokhof (eds.), *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers,* Foris Publications, 115-43.

Partee, B. and Rooth, M. 1983: Generalized Conjunction and Type Ambiguity. In R. Bäuerle, C. Schwarze and A. von Stechow (eds.), *Meaning, Use, and Interpretation of Language,* de Gruyter, 361-83.

Partee, B.H., ter Meulen, A., and Wall, R.E. 1990: *Mathematical methods in linguistics.* Kluwer.

Pelletier, F.J. 1984: Two Theories for Computing the Logical Form of Mass Expressions. *Proceedings of Tenth COLING/Twenty-second ACL,* 108-11.

Pereira, F. 1985: A new characterization of attachment preferences. In D.R. Dowty, L. Karttunen and A.M. Zwicky (eds.), *Natural Language Parsing,* Cambridge University Press, 307-19.

Pereira, F.C.N 1990: Categorial Semantics and Scoping. *Computational Linguistics,* 16(1), 1-10.

Pereira, F.C.N and Pollack, M.E. 1991: Incremental interpretation. *Artificial Intelligence,* 50, 37-82.

Pereira, F.C.N. and Shieber, S.M. 1987: *Prolog and Natural Language Analysis.* CSLI Lecture Notes, 10, Chicago University Press.

Pereira, F.C.N and Warren, D.H.D. 1980: Definite Clause Grammars for Language Analysis — A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence,* 13, 231-78.

Perrault, C. R. 1984: On The Mathematical Properties of Linguistic Theories. *Computational Linguistics,* 10(3-4), 165-76.

Pollack, M.E. and Pereira, F.C.N. 1988: An Integrated Framework for Semantic and Pragmatic Interpretation. *Proceedings of Twenty-sixth ACL,* 75-86.

Pollard, C. and Sag, I.A. 1994: *Head-Driven Phrase Structure Grammar.* Chicago University Press.

Pratt, V.R. 1975: LINGOL — A Progress Report. *Proceedings of Fourth IJCAI,* 422-8.

Pullum, G.K. 1982: Syncategorematicity and English Infinitival *To. Glossa,* 16, 181-215.

Pullum, G.K. 1983: Context-Freeness and the Computer Processing of Human Language. *Proceedings of 21st Annual Meeting of the ACL,* 1-6.

Pullum, G.K. 1984a: On Two Recent Attempts to Show that English is not a CFL. *Computational Linguistics,* 10(3-4), 182-6.

Pullum, G.K. 1984b: Syntactic and Semantic Parsability. *Proceedings of Tenth COLING/Twenty-second ACL,* 112-22.

Pullum, G.K. 1985: *Such That* Clauses and the Context-Freeness of English. *Linguistic Inquiry,* 16(2), 291-8.

Pullum, G.K. 1991: *The Great Eskimo Vocabulary Hoax.* Chicago University Press.

Pullum, G.K. and Gazdar, G. 1982: Natural Languages and Context-Free Languages. *Linguistics and Philosophy,* 4, 471-504.

Pulman, S.G. 1986: Grammars, Parsers and Memory Limitations. *Language and Cognitive Processes,* 1(3), 197-225.

Pulman, S. 1987: The Syntax-Semantics Interface. In P. Whitelock, M.M. Wood, H.L. Somers, R.Johnson, and P. Bennet (eds.), *Linguistic Theory and Computer Applications,* Academic Press, 189-224.

Pulman, S.G. 1989: *Events and VP Modifiers.* Technical Report No. 156, University of Cambridge Computer Laboratory.

Radford, A. 1988: *Transformational Grammar: a first course.* Cambridge University Press.

Reyle, U. 1993: Dealing with Ambiguities by Underspecification: Construction, Representation and Deduction. *Journal of Semantics,* 10(2), 123-79.

Reyle, U. 1995: On Reasoning with Ambiguities. *Proceedings of Seventh European ACL,* ???-???.

Rounds, W.C. 1996: Feature Logics. In J.F.A.K. van Benthem and A.G.G. ter Meulen (eds.), *Handbook of Logic and Language,* Elsevier, 475-533.

Ross, P. and Lewis, J. 1987: *Plan Recognition and Chart Planning.* Department of Artificial Intelligence Research Paper No. 309, University of Edinburgh.

Quirk, R., Greenbaum, S., Leech, G. and Svartvik, J. 1985: *A Comprehensive Grammar of the English Language.* Longman.

Rieger, C. 1976: An Organization of Knowledge for Problem Solving and Language Comprehension. *Artificial Intelligence,* 7(2), 89-127.

Ritchie, G. 1980: *Computational Grammar.* Barnes and Noble.

Ritchie, G. 1983: Semantics in Parsing. In M. King (ed.), *Parsing Natural Language,* Academic Press, 199-217.

Rosenschein, S.J. and Shieber, S.M. 1982: Translating English into logical form. *Proceedings of Twentieth ACL,* 1-8.

Russell, B. 1905: On Denoting. *Mind,* 14, 479-93.

Sag, I.A. 1997: English Relative Clause Constructions. *Journal of Linguistics,* 33(2), 431-483.

Sag, I.A. and Wasow, T. 1999: *Syntactic Theory: a Formal Introduction.* CSLI Publications.

Sampson, G.R. 1983: Deterministic Parsing. In M. King (ed.), *Parsing Natural Language,* Academic Press, 91-116.

Salveter, S. 1982: Natural Language Updates. *Proceedings of Ninth COLING,* 345-50.

Salveter, S. 1986: Supporting Natural Language Database Updates by Modelling Real World Actions. In L.Kerschberg (ed.), *Expert Database Systems (Proceedings from the First International Workshop),* Benjamin/Cummings, 639-57.

Satta, G. and Stock, O. 1989: Head-Driven Bidirectional Parsing: A Tabular Method. *Proceedings of International Workshop on Parsing Technologies,* 43-51.

Satta, G. and Stock, O. 1991: A Tabular Method for Island-Driven Context-Free Grammar Parsing. *Proceedings of Ninth AAAI,* 143-8.

Scha, R. 1981: Distributive, Collective and Cumulative Quantification. In J.A.G. Groenendijk, T.M.V. Janssen and M.B.J. Stokhof (eds.), *Formal Methods in the Study of Language,* Amsterdam: Mathematical Centre Tracts 135, 131-58.

Schank, R.C. 1975: *Conceptual Information Processing.* North-Holland.

Schank, R.C. 1980: Language and Memory. *Cognitive Science,* 4(3), 243-84.

Schank, R.C. and Abelson, R.P. 1977: *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum.

Schank, R.C. and Rieger, C.J. 1974: Inference and the Computer Understanding of Natural Language. *Artificial Intelligence,* 5, 373-412.

Schmidt, C.F., Sridharan, N.S. and Goodson, J.L. 1978: The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence. *Artificial Intelligence,* 11, 45-83.

Schöter, A. 1994: *Evidential Bilattice Logic and Lexical Inference.* Centre for Cognitive Science, University of Edinburgh.

Schubert, L.K. 1984: On Parsing Preferences. *Proceedings of Tenth COLING/Twenty-second ACL,* 247-50.

Schubert, L.K. 1986: Are There Preference Trade-Offs in Attachment Decisions?. *Proceedings of Fifth AAAI,* 601-5.

Schubert, L.K. and Pelletier, F.J. 1982: From English to Logic: Context-Free Computation of 'Conventional' Logical Translation. *American Journal of Computational Linguistics,* 8(1), 26-44.

Seo, J. and Simmons, R.F. 1989: Syntactic Graphs: A Representation for the Union of All Ambiguous Parse Trees. *Computational Linguistics,* 15(1), 19-32.

Shieber, S. 1983a: Sentence Disambiguation by a Shift-Reduce Parsing Technique. *Proceedings of Twenty-first ACL,* 113-8.

Shieber, S. 1983b: Sentence Disambiguation by a Shift-Reduce Parsing Technique. *Proceedings of Eighth IJCAI,* 699-703.

Shieber, S.M. 1985: Evidence Against the Context-Freeness of Natural Languages. *Linguistics and Philosophy,* 8, 333-43.

Shieber, S.M. and Johnson, M. 1993: Variations on Incremental Interpretation. *Journal of Psycholinguistic Research,* 22(2), 287-318.

Simpkins, N.K. and Hancox, P. 1990: Chart Parsing in Prolog. *New Generation Computing,* 8, 113-38.

Slocum, J. 1981: A Practical Comparison of Parsing Strategies. *Proceedings of Nineteenth ACL,* 1-6.

Slocum, J. 1988: *Machine Translation Systems.* Cambridge University Press.

Small, S. 1979: Word Expert Parsing. *Proceedings of Seventeenth ACL,* 9-13.

Small, S. 1983: Parsing as Cooperative Distributed Inference: Understanding through Memory Interactions. In M. King (ed.), *Parsing Natural Language,* Academic Press, 247-76.

Smith, P.D. and Barnes, G.M. 1987: *Files and Databases.* Addison-Wesley.

Sondheimer, N.K., Crout, J.N., Mann, W.C., Petrick, S.R., Sacerdoti, E.R., Tennant, H. and Thompson, B.H. 1981: Evaluation of Natural Language Interfaces to Database Systems (Panel Discussion). *Proceedings of Nineteenth ACL,* 29-42.

Spärck Jones, K. 1982: So what about parsing compound nouns?. In B.K. Boguraev, K. Spärck Jones and J.I. Tait (eds.), *Three Papers on Parsing,* Technical Report No. 17, University of Cambridge Computer Laboratory, 1-6.

Spärck Jones, K., Bates, M., Carbonell, J.G., Flickinger, D. and McKeown, K.R. 1984: Is there still gold in the database mine (Panel discussion). *Proceedings of Tenth COLING/Twenty-second ACL,* 182-93.

Spiegel, F. (ed.) 1965: *What the Papers Didn't Mean to Say.* Scouse Press.

Stabler, Jr., E.P. 1983: Deterministic and bottom-up parsing in Prolog. *Proceedings of Fourth AAAI,* 383-6.

Stabler, Jr., Edward P. 1992: *The Logical Approach to Syntax: Foundations, Specifications, and Implementations of Theories of Government and Binding.* The MIT Press.

Sterling, L. and Shapiro, E. 1994: *The Art of Prolog (2nd edn.).* The MIT Press.

Strawson, P.F. 1950: On Referring. *Mind,* 59, 320-44.

Strzalkowski, T. and Cercone, N. 1989: Non-Singular Concepts in Natural Language Discourse. *Computational Linguistics,* 15(3), 171-86.

Tanaka, H. and Numazaki, H. 1989: Parallel Generalized LR Parsing based on Logic Programming. *Proceedings of International Workshop on Parsing Technologies,* 329-38.

Tennant, H. 1979: Experience with the Evaluation of Natural Language Question Answerers. *Proceedings of Sixth IJCAI,* 874-6.

Tennant, H.R., Ross, K.M., Saenz, R.M., Thompson, C.W. and Miller, J.R. 1983: Menu-Based Natural Language Understanding. *Proceedings of Twenty-first ACL,* 151-8.

Thomason, R.H. 1996: Non-monotonicity in Linguistics. In J.F.A.K. van Benthem and A.G.G. ter Meulen (eds.), *Handbook of Logic and Language,* Elsevier, 777-831.

Thompson, F.B. and Thompson, B.H. 1975: Practical Natural Language Processing: The REL System as Prototype. In M.Rubinoff and M. Yovitz (eds.), *Advances in Computing, vol.13,* Academic Press, 109-68.

Thompson, B.H. and Thompson, F.B. 1983: Introducing ASK: A Simple Knowledgeable System. *Proceedings of First Applied ACL,* 17-24.

Thompson, B.H. and Thompson, F.B 1985: ASK is Transportable in Half a Dozen Ways. *ACM Transactions on Office Information Systems,* 3(2), 185-203.

Thompson, H.S. 1981: Chart Parsing and Rule Schemata in PSG. *Proceedings of Nineteenth ACL,* 167-72.

Thompson, H. 1983: MCHART: A Flexible, Modular Chart Parsing System. *Proceedings of Fourth AAAI,* 408-10.

Thompson, H.S. 1989: Chart Parsing for Loosely Coupled Parallel Systems. *Proceedings of International Workshop on Parsing Technologies,* 320-8.

Thompson, H.S. and Ritchie, G.D. 1984: Implementing natural language parsers. In T. O'Shea and M. Eisenstadt (eds.), *Artificial Intelligence: Tools, Techniques, and Applications,* Harper and Row, 245-300.

Tomita. M. 1984a: LR parsers for natural languages. *Proceedings of Tenth COLING/Twenty-second ACL,* 354-7.

Tomita. M. 1984b: Disambiguating Grammatically Ambiguous Sentences by Asking. *Proceedings of Tenth COLING/Twenty-second ACL,* 476-80.

Tomita. M. 1985: An efficient context-free parsing algorithm for natural languages. *Proceedings of Ninth IJCAI,* 756-64.

Tomita, M. 1986: *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers.

Tomita, M. 1987: An Efficient Augmented Context-Free Parsing Algorithm. *Computational Linguistics,* 13(1), 31-46.

van Beek, P. and Cohen, R. 1991: Resolving Plan Ambiguity for Cooperative Response Generation. *Proceedings of Twelfth IJCAI,* 938-44.

van Benthem, J. 1986: *Essays in Logical Semantics.* D.Reidel.

van Eijk, J. 1983: Discourse representation theory and plurality. In A.G.B. ter Meulen (ed.), *Studies in Model-theoretic Semantics,* Foris, 85-106.

van Noord, G. 19??: An Efficient Implementation of the Head-Corner Parser. *Computational Linguistics???,* ??, ???-???.

Vilain, M. 1990: Getting Serious about Parsing Plans: a Grammatical Analysis of Plan Recognition. *Proceedings of Eighth AAAI,* 190-7.

Wahlster, W., Carbonell, J.G., Hendrix, G.G., and Tennant, H. 1986: Natural Language Interfaces — Ready for Commercial Success (Panel Discussion). *Proceedings of Eleventh COLING,* 161-7.

Walker, A., McCord, M.C., Sowa, J.F. and Wilson, W.G. 1987: *Knowledge Systems in Prolog.* Addison-Wesley.

Waltz, D. 1975: Natural Language Access to a Large Data Base: An Engineering Approach. *Proceedings of Fourth IJCAI,* 868-72.

Waltz, D.L. 1978: An English Language Question Answering System for a Large Relational Database. *Communications of the ACM,* 21(7), 526-39.

Waltz, D.L. and Goodman, B.A. 1977: Writing a Natural Language Data Base System. *Proceedings of Fifth IJCAI,* 144-50.

Warren, D.H.D and Pereira, F.C.N. 1982: An Efficient Easily Adaptable System for Interpreting Natural Language Queries. *American Journal of Computational Linguistics,* 8(3-4), 110-22.

Warren, D.S. 1983: Using λ-calculus to represent meanings in logic grammars. *Proceedings of Twenty-first ACL,* 51-6.

Webber, B.-L. 1986: Question, Answer and Responses: Interacting with Knowledge Base Systems. In M.L.Brodie and J.Mylopoulos (eds.), *On Knowledge Base Management Systems,* Springer Verlag, 365-402.

Webber, B. and Joshi, A. 1982: Taking the Initiative in Natural Language Data Base Interactions: Justifying Why. *Proceedings of Ninth COLING,* 413-8.

Weischedel, R.M. 1979: A New Semantic Computation While Parsing: Presupposition and Entailment. In C.-K. Oh and D. Dinneen (eds.), *Syntax and Semantics, Volume 11: Presupposition,* Academic Press, 155-82.

Wilensky, R. 1983: *Planning and Understanding.* Addison-Wesley.

Wilensky, R., Chin, D.N., Luria, M., Martin, J., Mayfield, J. and Wu, D. 1988: The Berkeley UNIX Consultant Project. *Computational Linguistics,* 14(4), 35-84.

Wilks, Y. 1975: A Preferential Pattern-Seeking Semantics for Natural Language Inference. *Artificial Intelligence,* 6, 53-74.

Wilks, Y. 1978: Making Preferences More Active. *Artificial Intelligence,* 11, 197-223.

Wilks, Y. 1983: Deep and Superficial Parsing. In M. King (ed.), *Natural Language Parsing,* Academic Press, 219-46.

Wilks, Y. 1985: Right attachment and preference semantics. *Proceedings of Second European ACL,* 89-92.

Wilks, Y., Huang, X., and Fass, D. 1985: Syntax, preference, and right attachment. *Proceedings of Ninth IJCAI,* 779-84.

Winograd, T. 1972: *Understanding Natural Language.* Edinburgh University Press.

Winograd, T. 1983: *Language as a Cognitive Process, Vol.1, Syntax.* Addison-Wesley.

Winston, P.H. 1984: *Artificial Intelligence (2nd edn.).* Addison-Wesley.

Wirén, M. 1987: A Comparison of Rule-Invocation Strategies in Context-Free Chart Parsing. *Proceedings of Third European ACL,* 226-33.

Woods, W.A. 1978: Semantics and quantification in natural language question answering. In M. Yovitz (ed.), *Advances in Computing, vol.17,* Academic Press, 2-64.

Woods, W.A. 1980: Cascaded ATN Grammars. *American Journal of Computational Linguistics,* 6(1), 1-12.

Woods, W.A. 1981: Procedural Semantics as a Theory of Meaning. In A.K.Joshi, B.L.Webber and I.A.Sag (eds.), *Elements of Discourse Understanding,* Cambridge University Press, 300-34.

Wright, J.H. and Wrigley, E.N. 1989: Probabilistic LR Parsing for Speech Recognition. *Proceedings of International Workshop on Parsing Technologies,* 105-14.

Zaniolo, C., Ait-Kaci, H., Beech, D., Camarata, S., Kerschberg, L. and Maier, D. 1986: Object-Oriented Database Systems and Knowledge Systems. In L.Kerschberg (ed.), *Expert Database Systems (Proceedings from the First International Workshop),* Benjamin/Cummings, 49-65.

Zeevat, H. 1989: A Compositional Approach to Discourse Representation Theory. *Linguistics and Philosophy,* 12, 95-131.