CHAPTER

14 The Representation of Sentence Meaning

ISHMAEL: *Surely all this is not without meaning.*
Herman Melville, *Moby Dick*

meaning representations

meaning representation languages

The approach to semantics introduced here, and elaborated on in the next two chapters, is based on the idea that the meaning of linguistic expressions can be captured in formal structures called **meaning representations**. Correspondingly, the frameworks that specify the syntax and semantics of these representations are called **meaning representation languages**. These meaning representations play a role analogous to that of the syntactic representations introduced in earlier chapters—they abstract away from surface forms and facilitate downstream processing.

The need for meaning representations arises when neither the raw linguistic inputs nor any of the syntactic structures derivable from them facilitate the kind of semantic processing that is required. We need representations that bridge the gap from linguistic inputs to the knowledge of the world needed to perform tasks. Consider the following ordinary language tasks that require some form of semantic processing of natural language:

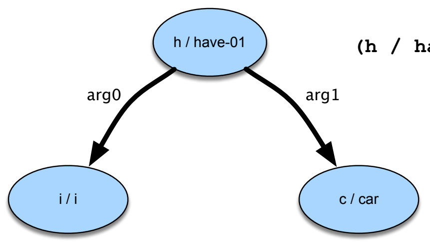
- Deciding what to order at a restaurant by reading a menu
- Learning to use a new piece of software by reading the manual
- Answering essay questions on an exam
- Realizing that you've been insulted
- Following recipes

Grammatical representations aren't sufficient to accomplish these tasks. What is required are representations that link the linguistic elements to the non-linguistic *knowledge of the world* needed to successfully accomplish the tasks:

- Reading a menu and deciding what to order, giving advice about where to go to dinner, following a recipe, and generating new recipes all require knowledge about food and its preparation, what people like to eat, and what restaurants are like.
- Answering and grading essay questions requires background knowledge about the topic of the question, the desired knowledge level of the students, and how such questions are *normally* answered.
- Learning to use a piece of software by reading a manual, or giving advice about how to use the software, requires knowledge about current computers, the specific software in question, similar software applications, and knowledge about users in general.

In this chapter, we assume that linguistic expressions have meaning representations that are made up of the *same kind of stuff* that is used to represent this kind

$$\exists e, y\ Having(e) \wedge Haver(e, Speaker) \wedge HadThing(e, y) \wedge Car(y)$$




```
(h / have-01
     arg0: (i / i)
     arg1: (c / car))
```

```
Having:
    Haver:  Speaker
    HadThing:  Car
```

Figure 14.1 A list of symbols, two directed graphs, and a record structure: a sampler of meaning representations for *I have a car.*

of everyday common-sense knowledge of the world. The process whereby such representations are created and assigned to linguistic inputs is called **semantic analysis**, and the entire enterprise of designing meaning representations and associated semantic analyzers is referred to as **computational semantics**.

computational semantics

To make these notions a bit more concrete, consider Fig. 14.1, which shows example meaning representations for the sentence *I have a car* using four commonly used meaning representation languages. The top row illustrates a sentence in **First-Order Logic**, covered in detail in Section 14.3; the directed graph and its corresponding textual form is an example of an **Abstract Meaning Representation (AMR)** form, to be discussed in Chapter 18, and finally a **Frame-Based** or **Slot-Filler** representation, discussed in Section 14.5 and again in Chapter 17.

While there are non-trivial differences among these approaches, they all share the notion that a meaning representation consists of structures composed from a set of symbols, or representational vocabulary. When appropriately arranged, these symbol structures are taken to *correspond* to objects, properties of objects, and relations among objects in some state of affairs being represented or reasoned about. In this case, all four representations make use of symbols corresponding to the speaker, a car, and a relation denoting the possession of one by the other.

Importantly, these representations can be viewed from at least two distinct perspectives in all of these approaches: as representations of the meaning of the particular linguistic input *I have a car*, and as representations of the state of affairs in some world. It is this dual perspective that allows these representations to be used to link linguistic inputs to the world and to our knowledge of it.

literal meaning

This chapter introduces the basics of what is needed in a meaning representation. A number of extremely important issues are therefore deferred to later chapters. The focus of this chapter is on representing the **literal meaning** of individual sentences. By this, we mean representations that are derived from the core conventional meanings of words and that do not reflect much of the context in which they occur. Chapter 15 and Chapter 16 introduce techniques for generating these formal meaning representations for linguistic inputs. Chapter 17 focuses on the extraction of entities, relations events and times, Chapter 18 and Chapter 19 on semantic structure of verbs and their arguments, while the task of producing representations for larger stretches of discourse is deferred to Chapter 20 and Chapter 21.

There are four major parts to this chapter. Section 14.1 explores some of the key computational requirements for what we need in a meaning representation language. Section 14.2 discusses how we can provide some guarantees that these representations will actually do what we need them to do—provide a correspondence to the state of affairs being represented. Section 14.3 then introduces First-Order Logic, which has historically been the primary technique for investigating issues in natural language semantics. Section 14.4 then describes how FOL can be used to capture the semantics of events and states in English.

14.1 Computational Desiderata for Representations

We begin by considering the issue of why meaning representations are needed and what they should do for us. To focus this discussion, we use the task of giving advice about restaurants to tourists. Assume that we have a computer system that accepts spoken language inputs from tourists and constructs appropriate responses using a knowledge base of relevant domain knowledge. A series of examples will serve to introduce some of the basic requirements that a meaning representation must fulfill and some of the complications that inevitably arise in the process of designing such meaning representations.

14.1.1 Verifiability

Consider the following simple question:

(14.1) Does Maharani serve vegetarian food?

This example illustrates the most basic requirement for a meaning representation: it must be possible to use the representation to determine the relationship between the meaning of a sentence and the state of the world as we know it. In other words, we need to be able to determine the truth of our representations. Section 14.2 explores the standard approach to this topic in some detail. For now, let's assume that computational semantic systems require the ability to compare, or *match*, meaning representations associated with linguistic expressions with formal representations in

knowledge base

a **knowledge base**, its store of information about its world.

In this example, assume that the meaning of this question involves the proposition *Maharani serves vegetarian food*. For now, we will simply gloss this representation as

$$Serves(Maharani, VegetarianFood) \tag{14.2}$$

This representation of the input can be matched against our knowledge base of facts about a set of restaurants. If the system finds a representation matching this proposition in its knowledge base, it can return an affirmative answer. Otherwise, it must either say *No* if its knowledge of local restaurants is complete, or say that it doesn't know if there is reason to believe that its knowledge is incomplete.

verifiability

This notion, known as **verifiability**, describes a system's ability to compare the state of affairs described by a representation to the state of affairs in some world as modeled in a knowledge base.

14.1.2 Unambiguous Representations

Semantics, like all the other domains we have studied, is subject to ambiguity. Specifically, individual linguistic expressions can have different meaning representations assigned to them based on the circumstances in which they occur. Consider the following example from the BERP corpus:

(14.3) I wanna eat someplace that's close to ICSI.

Given the allowable argument structures for the verb *eat*, this sentence can either mean that the speaker wants to eat *at* some nearby location, or under a Godzilla-as-speaker interpretation, the speaker may want to devour some nearby location. The answer generated by the system for this request will depend on which interpretation is chosen as the correct one.

Since ambiguities such as this abound in all genres of all languages, some means of determining that certain interpretations are preferable (or alternatively, not as preferable) to others is needed. The various linguistic phenomena that give rise to such ambiguities and the techniques that can be employed to deal with them are discussed in detail in the next four chapters.

Our concern in this chapter, however, is with the status of our meaning representations with respect to ambiguity, and not with the means by which we might arrive at correct interpretations. Since we reason about, and act upon, the semantic content of linguistic inputs, the final representation of an input's meaning should be free from any ambiguity.[1]

vagueness

A concept closely related to ambiguity is **vagueness**. Like ambiguity, vagueness can make it difficult to determine what to do with a particular input on the basis of its meaning representation. Vagueness, however, does not give rise to multiple representations. Consider the following request as an example:

(14.4) I want to eat Italian food.

While the use of the phrase *Italian food* may provide enough information for a restaurant advisor to provide reasonable recommendations, it is nevertheless quite *vague* as to what the user really wants to eat. Therefore, a vague representation of the meaning of this phrase may be appropriate for some purposes, while a more specific representation may be needed for other purposes. It will, therefore, be advantageous for a meaning representation language to support representations that maintain a certain level of vagueness. Note that it is not always easy to distinguish ambiguity from vagueness. Zwicky and Sadock (1975) provide a useful set of tests that can be used as diagnostics.

14.1.3 Canonical Form

The notion that single sentences can be assigned multiple meanings leads to the related phenomenon of distinct inputs that should be assigned the same meaning representation. Consider the following alternative ways of expressing (14.1):

(14.5) Does Maharani have vegetarian dishes?

(14.6) Do they have vegetarian food at Maharani?

(14.7) Are vegetarian dishes served at Maharani?

(14.8) Does Maharani serve vegetarian fare?

---

[1] This does not preclude the use of intermediate semantic representations that maintain some level of ambiguity on the way to a single unambiguous form. Examples of such representations are discussed in Chapter 15.

Given that these alternatives use different words and have widely varying syntactic analyses, it would not be unreasonable to expect them to have quite different meaning representations. Such a situation would, however, have undesirable consequences for how we determine the truth of our representations. If the system's knowledge base contains only a single representation of the fact in question, then the representations underlying all but one of our alternatives will fail to produce a match. We could, of course, store all possible alternative representations of the same fact in the knowledge base, but doing so would lead to an enormous number of problems related to keeping such a knowledge base consistent.

The way out of this dilemma is motivated by the fact that since the answers given for each of these alternatives should be the same in all situations, we might say that they all mean the same thing, at least for the purposes of giving restaurant recommendations. In other words, at least in this domain, we can legitimately consider assigning the same meaning representation to the propositions underlying each of these requests. Taking such an approach would guarantee that our simple scheme for answering yes-no questions will still work.

canonical form

The notion that distinct inputs that mean the same thing should have the same meaning representation is known as the doctrine of **canonical form**. This approach greatly simplifies various reasoning tasks since systems need only deal with a single meaning representation for a potentially wide range of expressions.

Canonical form does complicate the task of semantic analysis. To see this, note that the alternatives given above use completely different words and syntax to refer to vegetarian fare and to what restaurants do with it. To assign the same representation to all of these requests, our system would have to conclude that *vegetarian fare*, *vegetarian dishes*, and *vegetarian food* refer to the same thing in this context, that the use here of *having* and *serving* are similarly equivalent, and that the different syntactic parses underlying these requests are all compatible with the same meaning representation.

Being able to assign the same representation to such diverse inputs is a tall order. Fortunately, systematic meaning relationships among word senses and among grammatical constructions can be exploited to make this task tractable. Consider the issue of the meanings of the words *food*, *dish*, and *fare* in these examples. A little introspection or a glance at a dictionary reveals that these words have a fair number of distinct uses. However, it also reveals that at least one sense is shared among them all. If a system has the ability to choose that shared sense, then an identical meaning representation can be assigned to the phrases containing these words.

Just as there are systematic relationships among the meanings of different words, there are similar relationships related to the role that syntactic analyses play in assigning meanings to sentences. Specifically, alternative syntactic analyses often have meanings that are, if not identical, at least systematically related to one another. Consider the following pair of examples:

(14.9) Maharani serves vegetarian dishes.

(14.10) Vegetarian dishes are served by Maharani.

Despite the different placement of the arguments to *serve* in these examples, we can still assign *Maharani* and *vegetarian dishes* to the same roles in both of these examples because of our knowledge of the relationship between active and passive sentence constructions. In particular, we can use knowledge of where grammatical subjects and direct objects appear in these constructions to assign *Maharani* to the role of the server, and *vegetarian dishes* to the role of thing being served in both of these examples, despite the fact that they appear in different surface locations.

The precise role of the grammar in the construction of meaning representations is covered in Chapter 15.

14.1.4 Inference and Variables

Continuing with the topic of the computational purposes that meaning representations should serve, we should consider more complex requests such as the following:

(14.11) Can vegetarians eat at Maharani?

Here, it would be a mistake to invoke canonical form to force our system to assign the same representation to this request as for the previous examples. That this request results in the same answer as the others arises, not because they mean the same thing, but because there is a common-sense connection between what vegetarians eat and what vegetarian restaurants serve. This is a fact about the world and not a fact about any particular kind of linguistic regularity. This implies that no approach based on canonical form and simple matching will give us an appropriate answer to this request. What is needed is a systematic way to connect the meaning representation of this request with the facts about the world as they are represented in a knowledge base.

inference

We use the term **inference** to refer generically to a system's ability to draw valid conclusions based on the meaning representation of inputs and its store of background knowledge. It must be possible for the system to draw conclusions about the truth of propositions that are not explicitly represented in the knowledge base but that are nevertheless logically derivable from the propositions that are present.

Now consider the following somewhat more complex request:

(14.12) I'd like to find a restaurant where I can get vegetarian food.

Unlike our previous examples, this request does not make reference to any particular restaurant. The user is expressing a desire for information about an unknown and unnamed entity that is a restaurant that serves vegetarian food. Since this request does not mention any particular restaurant, the kind of simple matching-based approach we have been advocating is not going to work. Rather, answering this request requires a more complex kind of matching that involves the use of variables. We can gloss a representation containing such variables as follows:

$$Serves(x, VegetarianFood) \tag{14.13}$$

Matching such a proposition succeeds only if the variable x can be replaced by some known object in the knowledge base in such a way that the entire proposition will then match. The concept that is substituted for the variable can then be used to fulfill the user's request. Of course, this simple example only hints at the issues involved in the use of such variables. Suffice it to say that linguistic inputs contain many instances of all kinds of indefinite references, and it is, therefore, critical for any meaning representation language to be able to handle this kind of expression.

14.1.5 Expressiveness

Finally, to be useful, a meaning representation scheme must be expressive enough to handle a wide range of subject matter. The ideal situation would be to have a single meaning representation language that could adequately represent the meaning of any sensible natural language utterance. Although this is probably too much to expect from any single representational system, First-Order Logic, as described in

Section 14.3, is expressive enough to handle quite a lot of what needs to be represented.

14.2 Model-Theoretic Semantics

The last two sections focused on various desiderata for meaning representations and on some of the ways in which natural languages convey meaning. We haven't said much formally about what it is about meaning representation languages that allows them to do all the things we want them to. In particular, we might like to have some kind of guarantee that these representations can do the work that we require of them: bridge the gap from merely formal representations to representations that tell us something about some state of affairs in the world.

To see how we might provide such a guarantee, let's start with the basic notions shared by most meaning representation schemes. What they all have in common is the ability to represent objects, properties of objects, and relations among objects. This ability can be formalized by the notion of a **model**. A model is a formal construct that stands for the particular state of affairs in the world. Expressions in a meaning representation language can be mapped in a systematic way to the elements of the model. If the model accurately captures the facts we're interested in concerning some state of affairs, then a consistent mapping between the meaning representation and the model provides the bridge between the meaning representation and world being considered. As we show, models provide a surprisingly simple and powerful way to ground the expressions in meaning representation languages.

model

First, some terminology. The vocabulary of a meaning representation consists of two parts: the non-logical vocabulary and the logical vocabulary. The **non-logical vocabulary** consists of the open-ended set of names for the objects, properties, and relations that make up the world we're trying to represent. These appear in various schemes as predicates, nodes, labels on links, or labels in slots in frames, The **logical vocabulary** consists of the closed set of symbols, operators, quantifiers, links, etc., that provide the formal means for composing expressions in a given meaning representation language.

non-logical vocabulary

logical vocabulary

We'll start by requiring that each element of the non-logical vocabulary have a **denotation** in the model. By denotation, we simply mean that every element of the non-logical vocabulary corresponds to a fixed, well-defined part of the model. Let's start with objects, the most basic notion in most representational schemes. The **domain** of a model is simply the set of objects that are part of the application, or state of affairs, being represented. Each distinct concept, category, or individual in an application denotes a unique element in the domain. A domain is therefore formally a set. Note that it isn't mandatory that every element of the domain have a corresponding concept in our meaning representation; it's perfectly acceptable to have domain elements that aren't mentioned or conceived of in the meaning representation. Nor do we require that elements of the domain have a single denoting concept in the meaning representation; a given element in the domain might have several distinct representations denoting it, such as *Mary*, *WifeOf(Abe)*, or *MotherOf(Robert)*.

denotation

domain

We can capture properties of objects in a model by denoting those domain elements that have the property in question; that is, properties denote sets. Similarly, relations among objects denote sets of ordered lists, or tuples, of domain elements that take part in the corresponding relations. This approach to properties and relations is thus an **extensional** one: the denotation of properties like *red* is the set of

extensional

things we think are red, the denotation of a relation like *Married* is simply set of pairs of domain elements that are married. To summarize:

- Objects denote *elements* of the domain
- Properties denote *sets of elements* of the domain
- Relations denote *sets of tuples of elements* of the domain

There is one additional element that we need to make this scheme work. We need a mapping that systematically gets us from our meaning representation to the corresponding denotations. More formally, we need a function that maps from the non-logical vocabulary of our meaning representation to the proper denotations in the model. We'll call such a mapping an **interpretation**.

interpretation

To make these notions more concrete, let's return to our restaurant advice application. Assume that our application domain consists of sets of restaurants, patrons, and various facts about the likes and dislikes of the patrons, and facts about the restaurants such as their cuisine, typical cost, and noise level.

To begin populating our domain, $\mathscr{D}$, let's assume that we're dealing with four patrons designated by the non-logical symbols *Matthew, Franco, Katie*, and *Caroline*. These four symbols will denote four unique domain elements. We'll use the constants a, b, c and, d to stand for these domain elements. Note that we're deliberately using meaningless, non-mnemonic names for our domain elements to emphasize the fact that whatever it is that we know about these entities has to come from the formal properties of the model and not from the names of the symbols. Continuing, let's assume that our application includes three restaurants, designated as *Frasca*, *Med*, and *Rio* in our meaning representation, that denote the domain elements e, f, and g. Finally, let's assume that we're dealing with the three cuisines *Italian*, *Mexican*, and *Eclectic*, denoted by h, i, and j in our model.

Having populated the domain, let's move on to the properties and relations we believe to be true in this particular state of affairs. For our application, we need to represent various properties of restaurants such as the fact that some are noisy or expensive. Properties like *Noisy* denote the subset of restaurants from our domain that are known to be noisy. Two-place relational notions, such as which restaurants individual patrons *Like*, denote ordered pairs, or tuples, of the objects from the domain. And, since we decided to represent cuisines as objects in our model, we can capture which restaurants *Serve* which cuisines as a set of tuples. One possible state of affairs using this scheme is given in Fig. 14.2.

Given this simple scheme, we can ground our meaning representations by consulting the appropriate denotations in the corresponding model. For example, we can evaluate a representation claiming that *Matthew likes the Rio*, or that *The Med serves Italian* by mapping the objects in the meaning representations to their corresponding domain elements and mapping any links, predicates, or slots in the meaning representation to the appropriate relations in the model. More concretely, we can verify a representation asserting that *Matthew likes Frasca* by first using our interpretation function to map the symbol *Matthew* to its denotation a, *Frasca* to e, and the *Likes* relation to the appropriate set of tuples. We then check that set of tuples for the presence of the tuple $\langle a, e \rangle$. If, as it is in this case, the tuple is present in the model, then we can conclude that *Matthew likes Frasca* is true; if it isn't then we can't.

This is all pretty straightforward—we're using sets and operations on sets to ground the expressions in our meaning representations. Of course, the more interesting part comes when we consider more complex examples such as the following:

(14.14) Katie likes the Rio and Matthew likes the Med.

Domain	$\mathscr{D} = \{a,b,c,d,e,f,g,h,i,j\}$
Matthew, Franco, Katie and Caroline	a,b,c,d
Frasca, Med, Rio	e,f,g
Italian, Mexican, Eclectic	h,i,j
Properties	
Noisy	$Noisy = \{e,f,g\}$
Frasca, Med, and Rio are noisy	
Relations	
Likes	$Likes = \{\langle a,f\rangle, \langle c,f\rangle, \langle c,g\rangle, \langle b,e\rangle, \langle d,f\rangle, \langle d,g\rangle\}$
Matthew likes the Med	
Katie likes the Med and Rio	
Franco likes Frasca	
Caroline likes the Med and Rio	
Serves	$Serves = \{\langle f,j\rangle, \langle g,i\rangle, \langle e,h\rangle\}$
Med serves eclectic	
Rio serves Mexican	
Frasca serves Italian	

Figure 14.2 A model of the restaurant world.

(14.15) Katie and Caroline like the same restaurants.

(14.16) Franco likes noisy, expensive restaurants.

(14.17) Not everybody likes Frasca.

Our simple scheme for grounding the meaning of representations is not adequate for examples such as these. Plausible meaning representations for these examples will not map directly to individual entities, properties, or relations. Instead, they involve complications such as conjunctions, equality, quantified variables, and negations. To assess whether these statements are consistent with our model, we'll have to tear them apart, assess the parts, and then determine the meaning of the whole from the meaning of the parts according to the details of how the whole is assembled.

Consider the first example given above. A meaning representation for an example like this will include two distinct propositions expressing the individual patron's preferences, conjoined with some kind of implicit or explicit conjunction operator. Our model doesn't have a relation that encodes pairwise preferences for all of the patrons and restaurants in our model, nor does it need to. We know from our model that *Matthew likes the Med* and separately that *Katie likes the Rio* (that is, the tuples $\langle a,f\rangle$ and $\langle c,g\rangle$ are members of the set denoted by the *Likes* relation). All we really need to know is how to deal with the semantics of the conjunction operator. If we assume the simplest possible semantics for the English word *and*, the whole statement is true if it is the case that each of the components is true in our model. In this case, both components are true since the appropriate tuples are present and therefore the sentence as a whole is true.

truth-conditional semantics

What we've done with this example is provide a **truth-conditional semantics** for the assumed conjunction operator in some meaning representation. That is, we've provided a method for determining the truth of a complex expression from the meanings of the parts (by consulting a model) and the meaning of an operator by consulting a truth table. Meaning representation languages are truth-conditional to the extent that they give a formal specification as to how we can determine the mean-

$$
\begin{array}{rcl}
\textit{Formula} & \rightarrow & \textit{AtomicFormula} \\
& | & \textit{Formula Connective Formula} \\
& | & \textit{Quantifier Variable}, \ldots \textit{ Formula} \\
& | & \neg \textit{ Formula} \\
& | & (\textit{Formula}) \\
\textit{AtomicFormula} & \rightarrow & \textit{Predicate}(\textit{Term}, \ldots) \\
\textit{Term} & \rightarrow & \textit{Function}(\textit{Term}, \ldots) \\
& | & \textit{Constant} \\
& | & \textit{Variable} \\
\textit{Connective} & \rightarrow & \wedge \mid \vee \mid \Longrightarrow \\
\textit{Quantifier} & \rightarrow & \forall \mid \exists \\
\textit{Constant} & \rightarrow & A \mid \textit{VegetarianFood} \mid \textit{Maharani} \cdots \\
\textit{Variable} & \rightarrow & x \mid y \mid \cdots \\
\textit{Predicate} & \rightarrow & \textit{Serves} \mid \textit{Near} \mid \cdots \\
\textit{Function} & \rightarrow & \textit{LocationOf} \mid \textit{CuisineOf} \mid \cdots
\end{array}
$$

Figure 14.3 A context-free grammar specification of the syntax of First-Order Logic representations. Adapted from Russell and Norvig (2002)

ing of complex sentences from the meaning of their parts. In particular, we need to know the semantics of the entire logical vocabulary of the meaning representation scheme being used.

Note that although the details of how this happens depends on details of the particular meaning representation being used, it should be clear that assessing the truth conditions of examples like these involves nothing beyond the simple set operations we've been discussing. We return to these issues in the next section, where we discuss them in the context of the semantics of First-Order Logic.

14.3 First-Order Logic

First-Order Logic (FOL) is a flexible, well-understood, and computationally tractable meaning representation language that satisfies many of the desiderata given in Section 14.1. It provides a sound computational basis for the verifiability, inference, and expressiveness requirements, as well as a sound model-theoretic semantics.

An additional attractive feature of FOL is that it makes very few specific commitments as to how things ought to be represented. And, the specific commitments it does make are ones that are fairly easy to live with and that are shared by many of the schemes mentioned earlier; the represented world consists of objects, properties of objects, and relations among objects.

The remainder of this section introduces the basic syntax and semantics of FOL and then describes the application of FOL to the representation of events.

14.3.1 Basic Elements of First-Order Logic

Let's explore FOL by first examining its various atomic elements and then showing how they can be composed to create larger meaning representations. Figure 14.3, which provides a complete context-free grammar for the particular syntax of FOL that we will use, is our roadmap for this section.

term Let's begin by examining the notion of a **term**, the FOL device for representing objects. As can be seen from Fig. 14.3, FOL provides three ways to represent these basic building blocks: constants, functions, and variables. Each of these devices can be thought of as designating an object in the world under consideration.

Constants

Constants in FOL refer to specific objects in the world being described. Such constants are conventionally depicted as either single capitalized letters such as A and B or single capitalized words that are often reminiscent of proper nouns such as *Maharani* and *Harry*. Like programming language constants, FOL constants refer to exactly one object. Objects can, however, have multiple constants that refer to them.

Functions

Functions in FOL correspond to concepts that are often expressed in English as genitives such as *Frasca's location*. A FOL translation of such an expression might look like the following.

$$LocationOf(Frasca) \tag{14.18}$$

FOL functions are syntactically the same as single argument predicates. It is important to remember, however, that while they have the appearance of predicates, they are in fact *terms* in that they refer to unique objects. Functions provide a convenient way to refer to specific objects without having to associate a named constant with them. This is particularly convenient in cases in which many named objects, like restaurants, have a unique concept such as a location associated with them.

variable

Variables are **variable** our final FOL mechanism for referring to objects. Variables, depicted as single lower-case letters, let us make assertions and draw inferences about objects without having to make reference to any particular named object. This ability to make statements about anonymous objects comes in two flavors: making statements about a particular unknown object and making statements about all the objects in some arbitrary world of objects. We return to the topic of variables after we have presented quantifiers, the elements of FOL that make variables useful.

Now that we have the means to refer to objects, we can move on to the FOL mechanisms that are used to state relations that hold among objects. Predicates are symbols that refer to, or name, the relations that hold among some fixed number of objects in a given domain. Returning to the example introduced informally in Section 14.1, a reasonable FOL representation for *Maharani serves vegetarian food* might look like the following formula:

$$Serves(Maharani, VegetarianFood) \tag{14.19}$$

This FOL sentence asserts that *Serves*, a two-place predicate, holds between the objects denoted by the constants *Maharani* and *VegetarianFood*.

A somewhat different use of predicates is illustrated by the following fairly typical representation for a sentence like *Maharani is a restaurant*:

$$Restaurant(Maharani) \tag{14.20}$$

This is an example of a one-place predicate that is used, not to relate multiple objects, but rather to assert a property of a single object. In this case, it encodes the category membership of *Maharani*.

With the ability to refer to objects, to assert facts about objects, and to relate objects to one another, we can create rudimentary composite representations. These representations correspond to the atomic formula level in Fig. 14.3. This ability to compose complex representations is, however, not limited to the use of single predicates. Larger composite representations can also be put together through the

logical connectives

use of **logical connectives**. As can be seen from Fig. 14.3, logical connectives let us create larger representations by conjoining logical formulas using one of three operators. Consider, for example, the following BERP sentence and one possible representation for it:

(14.21) I only have five dollars and I don't have a lot of time.

$$Have(Speaker, FiveDollars) \wedge \neg Have(Speaker, LotOfTime) \tag{14.22}$$

The semantic representation for this example is built up in a straightforward way from semantics of the individual clauses through the use of the $\wedge$ and $\neg$ operators. Note that the recursive nature of the grammar in Fig. 14.3 allows an infinite number of logical formulas to be created through the use of these connectives. Thus, as with syntax, we can use a finite device to create an infinite number of representations.

14.3.2 Variables and Quantifiers

We now have all the machinery necessary to return to our earlier discussion of variables. As noted above, variables are used in two ways in FOL: to refer to particular anonymous objects and to refer generically to all objects in a collection. These two uses are made possible through the use of operators known as **quantifiers**. The two operators that are basic to FOL are the existential quantifier, which is denoted $\exists$ and is pronounced as "there exists", and the universal quantifier, which is denoted $\forall$ and is pronounced as "for all".

quantifiers

The need for an existentially quantified variable is often signaled by the presence of an indefinite noun phrase in English. Consider the following example:

(14.23) a restaurant that serves Mexican food near ICSI.

Here, reference is being made to an anonymous object of a specified category with particular properties. The following would be a reasonable representation of the meaning of such a phrase:

$$\begin{aligned} \exists x Restaurant(x) \ \wedge\ & Serves(x, MexicanFood) \\ \wedge\ & Near((LocationOf(x), LocationOf(ICSI)) \end{aligned} \tag{14.24}$$

The existential quantifier at the head of this sentence instructs us on how to interpret the variable x in the context of this sentence. Informally, it says that for this sentence to be true there must be at least one object such that if we were to substitute it for the variable x, the resulting sentence would be true. For example, if *AyCaramba* is a Mexican restaurant near ICSI, then substituting *AyCaramba* for x results in the following logical formula:

$$\begin{aligned} Restaurant(AyCaramba) \wedge Serves(AyCaramba, MexicanFood) \\ \wedge Near((LocationOf(AyCaramba), LocationOf(ICSI)) \end{aligned} \tag{14.25}$$

Based on the semantics of the $\wedge$ operator, this sentence will be true if all of its three component atomic formulas are true. These in turn will be true if they are either present in the system's knowledge base or can be inferred from other facts in the knowledge base.

The use of the universal quantifier also has an interpretation based on substitution of known objects for variables. The substitution semantics for the universal quantifier takes the expression *for all* quite literally; the $\forall$ operator states that for the logical formula in question to be true, the substitution of *any* object in the knowledge base for the universally quantified variable should result in a true formula. This is in marked contrast to the $\exists$ operator, which only insists on a single valid substitution for the sentence to be true.

Consider the following example:

(14.26) All vegetarian restaurants serve vegetarian food.

A reasonable representation for this sentence would be something like the following:

$$\forall x VegetarianRestaurant(x) \implies Serves(x, VegetarianFood) \tag{14.27}$$

For this sentence to be true, it must be the case that every substitution of a known object for x must result in a sentence that is true. We can divide the set of all possible substitutions into the set of objects consisting of vegetarian restaurants and the set consisting of everything else. Let us first consider the case in which the substituted object actually is a vegetarian restaurant; one such substitution would result in the following sentence:

$$VegetarianRestaurant(Maharani) \implies Serves(Maharani, VegetarianFood) \tag{14.28}$$

If we assume that we know that the consequent clause

$$Serves(Maharani, VegetarianFood) \tag{14.29}$$

is true, then this sentence as a whole must be true. Both the antecedent and the consequent have the value *True* and, therefore, according to the first two rows of Fig. 14.4 on page 15 the sentence itself can have the value *True*. This result will be the same for all possible substitutions of *Terms* representing vegetarian restaurants for x.

Remember, however, that for this sentence to be true, it must be true for all possible substitutions. What happens when we consider a substitution from the set of objects that are not vegetarian restaurants? Consider the substitution of a non-vegetarian restaurant such as *Ay Caramba's* for the variable x:

$$VegetarianRestaurant(AyCaramba) \implies Serves(AyCaramba, VegetarianFood)$$

Since the antecedent of the implication is *False*, we can determine from Fig. 14.4 that the sentence is always *True*, again satisfying the $\forall$ constraint.

Note that it may still be the case that *Ay Caramba* serves vegetarian food without actually being a vegetarian restaurant. Note also, that despite our choice of examples, there are no implied categorical restrictions on the objects that can be substituted for x by this kind of reasoning. In other words, there is no restriction of x to restaurants or concepts related to them. Consider the following substitution:

$$VegetarianRestaurant(Carburetor) \implies Serves(Carburetor, VegetarianFood)$$

Here the antecedent is still false, and hence, the rule remains true under this kind of irrelevant substitution.

To review, variables in logical formulas must be either existentially ($\exists$) or universally ($\forall$) quantified. To satisfy an existentially quantified variable, at least one substitution must result in a true sentence. Sentences with universally quantified variables must be true under all possible substitutions.

14.3.3 Lambda Notation

lambda notation

The final element we need to complete our discussion of FOL is called the **lambda notation** (Church, 1940). This notation provides a way to abstract from fully specified FOL formula in a way that will be particularly useful for semantic analysis. The lambda notation extends the syntax of FOL to include expressions of the following form:

$$\lambda x.P(x) \tag{14.30}$$

Such expressions consist of the Greek symbol λ, followed by one or more variables, followed by a FOL formula that makes use of those variables.

λ-reduction

The usefulness of these λ-expressions is based on the ability to apply them to logical terms to yield new FOL expressions where the formal parameter variables are bound to the specified terms. This process is known as **λ-reduction** and consists of a simple textual replacement of the λ variables with the specified FOL terms, accompanied by the subsequent removal of the λ. The following expressions illustrate the application of a λ-expression to the constant A, followed by the result of performing a λ-reduction on this expression:

$$\begin{array}{c} \lambda x.P(x)(A) \\ P(A) \end{array} \tag{14.31}$$

An important and useful variation of this technique is the use of one λ-expression as the body of another as in the following expression:

$$\lambda x.\lambda y.Near(x,y) \tag{14.32}$$

This fairly abstract expression can be glossed as the state of something being near something else. The following expressions illustrate a single λ-application and subsequent reduction with this kind of embedded λ-expression:

$$\begin{array}{c} \lambda x.\lambda y.Near(x,y)(Bacaro) \\ \lambda y.Near(Bacaro,y) \end{array} \tag{14.33}$$

The important point here is that the resulting expression is still a λ-expression; the first reduction bound the variable x and removed the outer λ, thus revealing the inner expression. As might be expected, this resulting λ-expression can, in turn, be applied to another term to arrive at a fully specified logical formula, as in the following:

$$\begin{array}{c} \lambda y.Near(Bacaro,y)(Centro) \\ Near(Bacaro,Centro) \end{array} \tag{14.34}$$

currying

This general technique, called **currying**[2] (Schönkfinkel, 1924) is a way of converting a predicate with multiple arguments into a sequence of single-argument predicates.

As we show in Chapter 15, the λ-notation provides a way to incrementally gather arguments to a predicate when they do not all appear together as daughters of the predicate in a parse tree.

[2] *Currying* is the standard term, although Heim and Kratzer (1998) present an interesting argument for the term *Schönkfinkelization* over currying, since Curry *later* built on Schönfinkel's work.

14.3.4 The Semantics of First-Order Logic

The various objects, properties, and relations represented in a FOL knowledge base acquire their meanings by virtue of their correspondence to objects, properties, and relations out in the external world being modeled. We can accomplish this by employing the model-theoretic approach introduced in Section 14.2. Recall that this approach employs simple set-theoretic notions to provide a truth-conditional mapping from the expressions in a meaning representation to the state of affairs being modeled. We can apply this approach to FOL by going through all the elements in Fig. 14.3 on page 10 and specifying how each should be accounted for.

We can start by asserting that the objects in our world, FOL terms, denote elements in a domain, and asserting that atomic formulas are captured either as sets of domain elements for properties, or as sets of tuples of elements for relations. As an example, consider the following:

(14.35) Centro is near Bacaro.

Capturing the meaning of this example in FOL involves identifying the *Terms* and *Predicates* that correspond to the various grammatical elements in the sentence and creating logical formulas that capture the relations implied by the words and syntax of the sentence. For this example, such an effort might yield something like the following:

$$Near(Centro, Bacaro) \tag{14.36}$$

The meaning of this logical formula is based on whether the domain elements denoted by the terms *Centro* and *Bacaro* are contained among the tuples denoted by the relation denoted by the predicate *Near* in the current model.

The interpretations of formulas involving logical connectives is based on the meaning of the components in the formulas combined with the meanings of the connectives they contain. Figure 14.4 gives interpretations for each of the logical operators shown in Fig. 14.3.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \implies Q$
False	*False*	*True*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*	*True*
True	*False*	*False*	*False*	*True*	*False*
True	*True*	*False*	*True*	*True*	*True*

Figure 14.4 Truth table giving the semantics of the various logical connectives.

The semantics of the $\wedge$ (and) and $\neg$ (not) operators are fairly straightforward, and are correlated with at least some of the senses of the corresponding English terms. However, it is worth pointing out that the $\vee$ (or) operator is not disjunctive in the same way that the corresponding English word is, and that the $\implies$ (implies) operator is only loosely based on any common-sense notions of implication or causation.

The final bit we need to address involves variables and quantifiers. Recall that there are no variables in our set-based models, only elements of the domain and relations that hold among them. We can provide a model-based account for formulas with variables by employing the notion of a substitution introduced earlier on page 12. Formulas involving $\exists$ are true if a substitution of terms for variables results in a formula that is true in the model. Formulas involving $\forall$ must be true under all possible substitutions.

14.3.5 Inference

One of the most important desiderata given in Section 14.1 for a meaning representation language is that it should support inference, or deduction. That is, the ability to add valid new propositions to a knowledge base or to determine the truth of propositions not explicitly contained within a knowledge base. This section briefly discusses **modus ponens**, the most widely implemented inference method provided by FOL. Applications of modus ponens to inference in discourse is discussed in Chapter 21.

Modus ponens

Modus ponens is a familiar form of inference that corresponds to what is informally known as *if-then* reasoning. We can abstractly define modus ponens as follows, where α and β should be taken as FOL formulas:

$$\frac{\begin{array}{l}\alpha \\ \alpha \implies \beta\end{array}}{\beta} \tag{14.37}$$

A schema like this indicates that the formula below the line can be inferred from the formulas above the line by some form of inference. Modus ponens simply states that if the left-hand side of an implication rule is true, then the right-hand side of the rule can be inferred. In the following discussions, we will refer to the left-hand side of an implication as the antecedent and the right-hand side as the consequent.

For a typical use of modus ponens, consider the following example, which uses a rule from the last section:

$$\frac{\begin{array}{l}VegetarianRestaurant(Leaf) \\ \forall x VegetarianRestaurant(x) \implies Serves(x, VegetarianFood)\end{array}}{Serves(Leaf, VegetarianFood)} \tag{14.38}$$

Here, the formula $VegetarianRestaurant(Leaf)$ matches the antecedent of the rule, thus allowing us to use modus ponens to conclude $Serves(Leaf, VegetarianFood)$.

forward chaining

Modus ponens can be put to practical use in one of two ways: forward chaining and backward chaining. In **forward chaining** systems, modus ponens is used in precisely the manner just described. As individual facts are added to the knowledge base, modus ponens is used to fire all applicable implication rules. In this kind of arrangement, as soon as a new fact is added to the knowledge base, all applicable implication rules are found and applied, each resulting in the addition of new facts to the knowledge base. These new propositions in turn can be used to fire implication rules applicable to them. The process continues until no further facts can be deduced.

The forward chaining approach has the advantage that facts will be present in the knowledge base when needed, because, in a sense all inference is performed in advance. This can substantially reduce the time needed to answer subsequent queries since they should all amount to simple lookups. The disadvantage of this approach is that facts that will never be needed may be inferred and stored.

backward chaining

In **backward chaining**, modus ponens is run in reverse to prove specific propositions called queries. The first step is to see if the query formula is true by determining if it is present in the knowledge base. If it is not, then the next step is to search for applicable implication rules present in the knowledge base. An applicable rule is one whereby the consequent of the rule matches the query formula. If there are any such rules, then the query can be proved if the antecedent of any one them can be shown to be true. Not surprisingly, this can be performed recursively by backward chaining on the antecedent as a new query. The Prolog programming language is a backward chaining system that implements this strategy.

To see how this works, let's assume that we have been asked to verify the truth of the proposition $Serves(Leaf, VegetarianFood)$, assuming the facts given above the line in (14.38). Since this proposition is not present in the knowledge base, a search for an applicable rule is initiated resulting in the rule given above. After substituting the constant *Leaf* for the variable x, our next task is to prove the antecedent of the rule, $VegetarianRestaurant(Leaf)$, which, of course, is one of the facts we are given.

Note that it is critical to distinguish between reasoning by backward chaining from queries to known facts and reasoning backwards from known consequents to unknown antecedents. To be specific, by reasoning backwards we mean that if the consequent of a rule is known to be true, we assume that the antecedent will be as well. For example, let's assume that we know that $Serves(Leaf, VegetarianFood)$ is true. Since this fact matches the consequent of our rule, we might reason backwards to the conclusion that $VegetarianRestaurant(Leaf)$.

While backward chaining is a sound method of reasoning, reasoning backwards is an invalid, though frequently useful, form of *plausible reasoning*. Plausible reasoning from consequents to antecedents is known as **abduction**, and as we show in Chapter 21, is often useful in accounting for many of the inferences people make while analyzing extended discourses.

abduction

complete

While forward and backward reasoning are sound, neither is **complete**. This means that there are valid inferences that cannot be found by systems using these methods alone. Fortunately, there is an alternative inference technique called **resolution** that is sound and complete. Unfortunately, inference systems based on resolution are far more computationally expensive than forward or backward chaining systems. In practice, therefore, most systems use some form of chaining and place a burden on knowledge-base developers to encode the knowledge in a fashion that permits the necessary inferences to be drawn.

resolution

14.4 Event and State Representations

Much of the semantics that we wish to capture consists of representations of states and events. States are conditions, or properties, that remain unchanged over an extended period of time, and events denote changes in some state of affairs. The representation of both states and events may involve a host of participants, props, times and locations.

The representations for events and states that we have used thus far have consisted of single predicates with as many arguments as are needed to incorporate all the roles associated with a given example. For example, the representation for *Leaf serves vegetarian fare* consists of a single predicate with arguments for the entity doing the serving and the thing served.

$$Serves(Leaf, VegetarianFare) \tag{14.39}$$

This approach assumes that the predicate used to represent an event verb has the same number of arguments as are present in the verb's syntactic subcategorization frame. Unfortunately, this is clearly not always the case. Consider the following examples of the verb *eat*:

(14.40) I ate.

(14.41) I ate a turkey sandwich.
(14.42) I ate a turkey sandwich at my desk.
(14.43) I ate at my desk.
(14.44) I ate lunch.
(14.45) I ate a turkey sandwich for lunch.
(14.46) I ate a turkey sandwich for lunch at my desk.

Clearly, choosing the correct number of arguments for the predicate representing the meaning of *eat* is a tricky problem. These examples introduce five distinct arguments, or roles, in an array of different syntactic forms, locations, and combinations. Unfortunately, predicates in FOL have fixed **arity** – they take a fixed number of arguments.

arity

event variable

To address this problem, we introduce the notion of an **event variable** to allow us to make assertions about particular events. To do this, we can refactor our event predicates to have an existentially quantified variable as their first, *and only*, argument. Using this event variable, we can introduce additional predicates to represent the other information we have about the event. These predicates take an event variable as their first argument and related FOL terms as their second argument. The following formula illustrates this scheme with the meaning representation of 14.41 from our earlier discussion.

$$\exists e\, Eating(e) \;\wedge\; Eater(e, Speaker) \wedge Eaten(e, TurkeySandwich)$$

Here, the quantified variable e stands for the eating event and is used to bind the event predicate with the core information provided via the named roles *Eater* and *Eaten*. To handle the more complex examples, we simply add additional relations to capture the provided information, as in the following for 14.46.

$$\begin{aligned} \exists e\, Eating(e) \;&\wedge\; Eater(e, Speaker) \wedge Eaten(e, TurkeySandwich) \\ &\wedge\; Meal(e, Lunch) \wedge Location(e, Desk) \end{aligned} \tag{14.47}$$

neo-Davidsonian

Event representations of this sort are referred to as **neo-Davidsonian** event representations (Davidson, 1967; Parsons, 1990) after the philosopher Donald Davidson who introduced the notion of an event variable (Davidson, 1967). To summarize, in the neo-Davidsonian approach to event representations:

- Events are captured with predicates that take a single event variable as an argument.
- There is no need to specify a fixed number of arguments for a given FOL predicate; rather, as many roles and fillers can be glued on as are provided in the input.
- No more roles are postulated than are mentioned in the input.
- The logical connections among closely related inputs that share the same predicate are satisfied without the need for additional inference.

This approach still leaves us with the problem of determining the set of predicates needed to represent roles associated with specific events like *Eater* and *Eaten*, as well as more general concepts like *Location* and *Time*. We'll return to this problem in more detail in Chapter 18.

14.4.1 Representing Time

In our discussion of events, we did not seriously address the issue of capturing the time when the represented events are supposed to have occurred. The representation temporal logic

tense logic

of such information in a useful form is the domain of **temporal logic**. This discussion introduces the most basic concerns of temporal logic and briefly discusses the means by which human languages convey temporal information, which, among other things, includes **tense logic**, the ways that verb tenses convey temporal information. A more detailed discussion of robust approaches to the representation and analysis of temporal expressions is presented in Chapter 17.

The most straightforward theory of time holds that it flows inexorably forward and that events are associated with either points or intervals in time, as on a timeline. Given these notions, we can order distinct events by situating them on the timeline. More specifically, we can say that one event *precedes* another if the flow of time leads from the first event to the second. Accompanying these notions in most theories is the idea of the current moment in time. Combining this notion with the idea of a temporal ordering relationship yields the familiar notions of past, present, and future.

Not surprisingly, a large number of schemes can represent this kind of temporal information. The one presented here is a fairly simple one that stays within the FOL framework of reified events that we have been pursuing. Consider the following examples:

(14.48) I arrived in New York.

(14.49) I am arriving in New York.

(14.50) I will arrive in New York.

These sentences all refer to the same kind of event and differ solely in the tense of the verb. In our current scheme for representing events, all three would share the following kind of representation, which lacks any temporal information:

$$\exists e Arriving(e) \wedge Arriver(e, Speaker) \wedge Destination(e, NewYork) \tag{14.51}$$

The temporal information provided by the tense of the verbs can be exploited by predicating additional information about the event variable e. Specifically, we can add temporal variables representing the interval corresponding to the event, the end point of the event, and temporal predicates relating this end point to the current time as indicated by the tense of the verb. Such an approach yields the following representations for our *arriving* examples:

$$\begin{aligned}
\exists e,i,n\, Arriving(e) &\wedge Arriver(e,Speaker) \wedge Destination(e,NewYork)\\
&\wedge IntervalOf(e,i) \wedge EndPoint(i,n) \wedge Precedes(n,Now)\\
\exists e,i,n\, Arriving(e) &\wedge Arriver(e,Speaker) \wedge Destination(e,NewYork)\\
&\wedge IntervalOf(e,i) \wedge MemberOf(i,Now)\\
\exists e,i,n\, Arriving(e) &\wedge Arriver(e,Speaker) \wedge Destination(e,NewYork)\\
&\wedge IntervalOf(e,i) \wedge EndPoint(i,n) \wedge Precedes(Now,n)
\end{aligned}$$

This representation introduces a variable to stand for the interval of time associated with the event and a variable that stands for the end of that interval. The two-place predicate *Precedes* represents the notion that the first time-point argument precedes the second in time; the constant *Now* refers to the current time. For past events, the end point of the interval must precede the current time. Similarly, for future events the current time must precede the end of the event. For events happening in the present, the current time is contained within the event interval.

Unfortunately, the relation between simple verb tenses and points in time is by no means straightforward. Consider the following examples:

(14.52) Ok, we fly from San Francisco to Boston at 10.

(14.53) Flight 1390 will be at the gate an hour now.

In the first example, the present tense of the verb *fly* is used to refer to a future event, while in the second the future tense is used to refer to a past event.

More complications occur when we consider some of the other verb tenses. Consider the following examples:

(14.54) Flight 1902 arrived late.

(14.55) Flight 1902 had arrived late.

Although both refer to events in the past, representing them in the same way seems wrong. The second example seems to have another unnamed event lurking in the background (e.g., Flight 1902 had already arrived late *when* something else happened). To account for this phenomena, Reichenbach (1947) introduced the notion of a **reference point**. In our simple temporal scheme, the current moment in time is equated with the time of the utterance and is used as a reference point for when the event occurred (before, at, or after). In Reichenbach's approach, the notion of the reference point is separated from the utterance time and the event time. The following examples illustrate the basics of this approach:

reference point

(14.56) When Mary's flight departed, I ate lunch.

(14.57) When Mary's flight departed, I had eaten lunch.

In both of these examples, the eating event has happened in the past, that is, prior to the utterance. However, the verb tense in the first example indicates that the eating event began when the flight departed, while the second example indicates that the eating was accomplished prior to the flight's departure. Therefore, in Reichenbach's terms the *departure* event specifies the reference point. These facts can be accommodated by additional constraints relating the *eating* and *departure* events. In the first example, the reference point precedes the *eating* event, and in the second example, the eating precedes the reference point. Figure 14.5 illustrates Reichenbach's approach with the primary English tenses. Exercise 14.6 asks you to represent these examples in FOL.

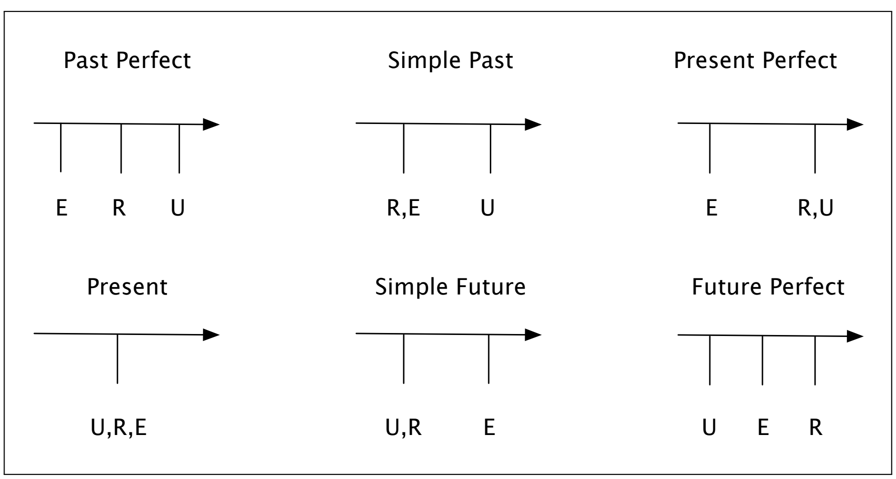


Figure 14.5 Reichenbach's approach applied to various English tenses. In these diagrams, time flows from left to right, an **E** denotes the time of the event, an **R** denotes the reference time, and an **U** denotes the time of the utterance.

This discussion has focused narrowly on the broad notions of past, present, and future and how they are signaled by various English verb tenses. Of course, languages also have many other more direct and more specific ways to convey temporal information, including the use of a wide variety of temporal expressions, as in the following ATIS examples:

(14.58) I'd like to go at 6:45, in the morning.

(14.59) Somewhere around noon, please.

As we show in Chapter 17, grammars for such temporal expressions are of considerable practical importance to information extraction and question-answering applications.

Finally, we should note that a systematic conceptual organization is reflected in examples like these. In particular, temporal expressions in English are frequently expressed in spatial terms, as is illustrated by the various uses of *at*, *in*, *somewhere*, and *near* in these examples (Lakoff and Johnson, 1980; Jackendoff, 1983). Metaphorical organizations such as these, in which one domain is systematically expressed in terms of another, are very common in languages of the world.

14.4.2 Aspect

In the last section, we discussed ways to represent the time of an event with respect to the time of an utterance describing it. In this section, we address the notion of **aspect**, which concerns a cluster of related topics, including whether an event has ended or is ongoing, whether it is conceptualized as happening at a point in time or over some interval, and whether any particular state in the world comes about because of it. Based on these and related notions, event expressions have traditionally been divided into four general classes illustrated in the following examples:

aspect

Stative: I know my departure gate.

Activity: John is flying.

Accomplishment: Sally booked her flight.

Achievement: She found her gate.

Although the earliest versions of this classification were discussed by Aristotle, the one presented here is due to Vendler (1967).

Stative expressions

Stative expressions represent the notion of an event participant having a particular property, or being in a state, at a given point in time. As such, these expressions can be thought of as capturing an aspect of a world at a single point in time. Consider the following ATIS examples.

(14.60) I like Flight 840 arriving at 10:06.

(14.61) I need the cheapest fare.

(14.62) I want to go first class.

In examples like these, the event participant denoted by the subject can be seen as experiencing something at a specific point in time. Whether or not the experiencer was in the same state earlier or will be in the future is left unspecified.

Activity expressions

Activity expressions describe events undertaken by a participant and have no particular end point. Unlike statives, activities are seen as occurring over some span of time and are therefore not associated with single points in time. Consider the following examples:

(14.63) She drove a Mazda.

(14.64) I live in Brooklyn.

These examples both specify that the subject is engaged in, or has engaged in, the activity specified by the verb for some period of time.

achievement expressions

The final aspectual class, **achievement expressions**, is similar to accomplishments in that these expressions result in a state. Consider the following:

(14.65) She found her gate.

(14.66) I reached New York.

Unlike accomplishments, achievement events are thought of as happening in an instant and are not equated with any particular activity leading up to the state. To be more specific, the events in these examples may have been preceded by extended *searching* or *traveling* events, but the events corresponding directly to *found* and *reach* are conceived of as points, not intervals.

Note that since both accomplishments and achievements are events that result in a state, they are sometimes characterized as subtypes of a single aspectual class.

telic eventualities

Members of this combined class are known as **telic eventualities**.

14.5 Description Logics

As noted at the beginning of this chapter, a fair number of representational schemes have been invented to capture the meaning of linguistic utterances. It is now widely accepted that meanings represented in these various approaches can, in principle, be translated into equivalent statements in FOL with relative ease. The difficulty is that in many of these approaches the semantics of a statement are defined procedurally. That is, the meaning arises from whatever the system that interprets it does with it.

Description logics are an effort to better specify the semantics of these earlier structured network representations and to provide a conceptual framework that is especially well suited to certain kinds of domain modeling. Formally, the term Description Logics refers to a family of logical approaches that correspond to varying subsets of FOL. The restrictions placed on the expressiveness of Description Logics serve to guarantee the tractability of various critical kinds of inference. Our focus here, however, will be on the modeling aspects of DLs rather than on computational complexity issues.

When using Description Logics to model an application domain, the emphasis is on the representation of knowledge about categories, individuals that belong to those categories, and the relationships that can hold among these individuals. The set of categories, or concepts, that make up a particular application domain is called its **terminology**. The portion of a knowledge base that contains the terminology is traditionally called the **TBox**; this is in contrast to the **ABox** that contains facts about individuals. The terminology is typically arranged into a hierarchical organization called an **ontology** that captures the subset/superset relations among the categories.

terminology

TBox

ABox

ontology

Returning to our earlier culinary domain, we represented domain concepts like using unary predicates such as $Restaurant(x)$; the DL equivalent simply omits the variable, so the restaurant category is simply written as $\mathsf{Restaurant}$.[3] To capture the fact that a particular domain element, such as *Frasca*, is a restaurant, we assert $\mathsf{Restaurant(Frasca)}$ in much the same way we would in FOL. The semantics of

[3] DL statements are conventionally typeset with a sans serif font. We'll follow that convention here, reverting to our standard mathematical notation when giving FOL equivalents of DL statements.

these categories are specified in precisely the same way that was introduced earlier in Section 14.2: a category like Restaurant simply denotes the set of domain elements that are restaurants.

Once we've specified the categories of interest in a particular domain, the next step is to arrange them into a hierarchical structure. There are two ways to capture the hierarchical relationships present in a terminology: we can directly assert relations between categories that are related hierarchically, or we can provide complete definitions for our concepts and then rely on inference to provide hierarchical relationships. The choice between these methods hinges on the use to which the resulting categories will be put and the feasibility of formulating precise definitions for many naturally occurring categories. We'll discuss the first option here and return to the notion of definitions later in this section.

subsumption

To directly specify a hierarchical structure, we can assert **subsumption** relations between the appropriate concepts in a terminology. The subsumption relation is conventionally written as $C \sqsubseteq D$ and is read as C is subsumed by D; that is, all members of the category C are also members of the category D. Not surprisingly, the formal semantics of this relation are provided by a simple set relation; any domain element that is in the set denoted by C is also in the set denoted by D.

Adding the following statements to the TBox asserts that all restaurants are commercial establishments and, moreover, that there are various subtypes of restaurants.

$$\mathsf{Restaurant} \sqsubseteq \mathsf{CommercialEstablishment} \tag{14.67}$$
$$\mathsf{ItalianRestaurant} \sqsubseteq \mathsf{Restaurant} \tag{14.68}$$
$$\mathsf{ChineseRestaurant} \sqsubseteq \mathsf{Restaurant} \tag{14.69}$$
$$\mathsf{MexicanRestaurant} \sqsubseteq \mathsf{Restaurant} \tag{14.70}$$

Ontologies such as this are conventionally illustrated with diagrams such as the one shown in Fig. 14.6, where subsumption relations are denoted by links between the nodes representing the categories.

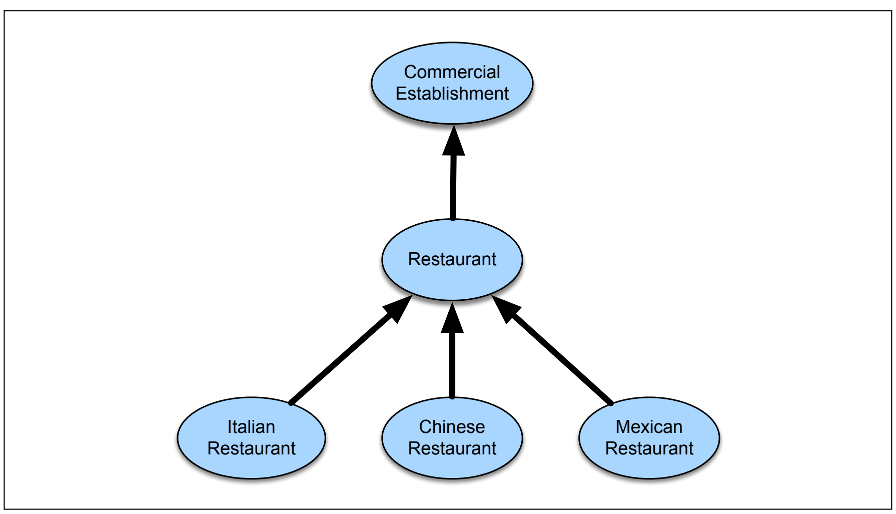


Figure 14.6 A graphical network representation of a set of subsumption relations in the restaurant domain.

Note, that it was precisely the vague nature of semantic network diagrams like this that motivated the development of Description Logics. For example, from this diagram we can't tell whether the given set of categories is exhaustive or disjoint. That is, we can't tell if these are all the kinds of restaurants that we'll be dealing with in our domain or whether there might be others. We also can't tell if an individual restaurant must fall into only *one* of these categories, or if it is possible, for example, for a restaurant to be *both* Italian and Chinese. The DL statements given above are more transparent in their meaning; they simply assert a set of subsumption relations between categories and make no claims about coverage or mutual exclusion.

If an application requires coverage and disjointness information, then such information must be made explicitly. The simplest ways to capture this kind of information is through the use of negation and disjunction operators. For example, the following assertion would tell us that Chinese restaurants can't also be Italian restaurants.

$$\mathsf{ChineseRestaurant} \sqsubseteq \textbf{not}\ \mathsf{ItalianRestaurant} \tag{14.71}$$

Specifying that a set of subconcepts covers a category can be achieved with disjunction, as in the following:

$$\begin{aligned}&\mathsf{Restaurant} \sqsubseteq \\ &\quad (\textbf{or}\ \mathsf{ItalianRestaurant\ ChineseRestaurant\ MexicanRestaurant})\end{aligned} \tag{14.72}$$

Having a hierarchy such as the one given in Fig. 14.6 tells us next to nothing about the concepts in it. We certainly don't know anything about what makes a restaurant a restaurant, much less Italian, Chinese, or expensive. What is needed are additional assertions about what it means to be a member of any of these categories. In Description Logics such statements come in the form of relations between the concepts being described and other concepts in the domain. In keeping with its origins in structured network representations, relations in Description Logics are typically binary and are often referred to as roles, or role-relations.

To see how such relations work, let's consider some of the facts about restaurants discussed earlier in the chapter. We'll use the hasCuisine relation to capture information as to what kinds of food restaurants serve and the hasPriceRange relation to capture how pricey particular restaurants tend to be. We can use these relations to say something more concrete about our various classes of restaurants. Let's start with our ItalianRestaurant concept. As a first approximation, we might say something uncontroversial like Italian restaurants serve Italian cuisine. To capture these notions, let's first add some new concepts to our terminology to represent various kinds of cuisine.

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{MexicanCuisine} & \sqsubseteq & \mathsf{Cuisine} & \mathsf{ExpensiveRestaurant} & \sqsubseteq & \mathsf{Restaurant}\\
\mathsf{ItalianCuisine} & \sqsubseteq & \mathsf{Cuisine} & \mathsf{ModerateRestaurant} & \sqsubseteq & \mathsf{Restaurant}\\
\mathsf{ChineseCuisine} & \sqsubseteq & \mathsf{Cuisine} & \mathsf{CheapRestaurant} & \sqsubseteq & \mathsf{Restaurant}\\
\mathsf{VegetarianCuisine} & \sqsubseteq & \mathsf{Cuisine} & & &
\end{array}$$

Next, let's revise our earlier version of ItalianRestaurant to capture cuisine information.

$$\mathsf{ItalianRestaurant} \ \sqsubseteq\ \mathsf{Restaurant} \sqcap \exists \mathsf{hasCuisine.ItalianCuisine} \tag{14.73}$$

The correct way to read this expression is that individuals in the category ItalianRestaurant are subsumed both by the category Restaurant and by an unnamed class defined by the existential clause—the set of entities that serve Italian cuisine. An equivalent statement in FOL would be

$$\forall x ItalianRestaurant(x) \rightarrow Restaurant(x) \wedge (\exists y Serves(x,y) \wedge ItalianCuisine(y)) \tag{14.74}$$

This FOL translation should make it clear what the DL assertions given above do and do not entail. In particular, they don't say that domain entities classified as Italian restaurants can't engage in other relations like being expensive or even serving Chinese cuisine. And critically, they don't say much about domain entities that we know do serve Italian cuisine. In fact, inspection of the FOL translation makes it clear that we cannot *infer* that any new entities belong to this category based on their characteristics. The best we can do is infer new facts about restaurants that we're explicitly told are members of this category.

Of course, inferring the category membership of individuals given certain characteristics is a common and critical reasoning task that we need to support. This brings us back to the alternative approach to creating hierarchical structures in a terminology: actually providing a definition of the categories we're creating in the form of necessary and sufficient conditions for category membership. In this case, we might explicitly provide a definition for ItalianRestaurant as being those restaurants that serve Italian cuisine, and ModerateRestaurant as being those whose price range is moderate.

$$\mathsf{ItalianRestaurant} \equiv \mathsf{Restaurant} \sqcap \exists \mathsf{hasCuisine.ItalianCuisine} \tag{14.75}$$

$$\mathsf{ModerateRestaurant} \equiv \mathsf{Restaurant} \sqcap \mathsf{hasPriceRange.ModeratePrices} \tag{14.76}$$

While our earlier statements provided necessary conditions for membership in these categories, these statements provide both necessary and sufficient conditions.

Finally, let's now consider the superficially similar case of vegetarian restaurants. Clearly, vegetarian restaurants are those that serve vegetarian cuisine. But they don't merely serve vegetarian fare, that's all they serve. We can accommodate this kind of constraint by adding an additional restriction in the form of a universal quantifier to our earlier description of VegetarianRestaurants, as follows:

$$\mathsf{VegetarianRestaurant} \equiv \mathsf{Restaurant} \sqcap \exists \mathsf{hasCuisine.VegetarianCuisine} \sqcap \forall \mathsf{hasCuisine.VegetarianCuisine} \tag{14.77}$$

Inference

Paralleling the focus of Description Logics on categories, relations, and individuals is a processing focus on a restricted subset of logical inference. Rather than employing the full range of reasoning permitted by FOL, DL reasoning systems emphasize the closely coupled problems of subsumption and instance checking.

Subsumption

Subsumption, as a form of inference, is the task of determining, based on the facts asserted in a terminology, whether a superset/subset relationship exists between two concepts. Correspondingly, **instance checking** asks if an individual can be a member of a particular category given the facts we know about both the individual and the terminology. The inference mechanisms underlying subsumption and instance checking go beyond simply checking for explicitly stated subsumption relations in a terminology. They must explicitly reason using the relational information

instance checking

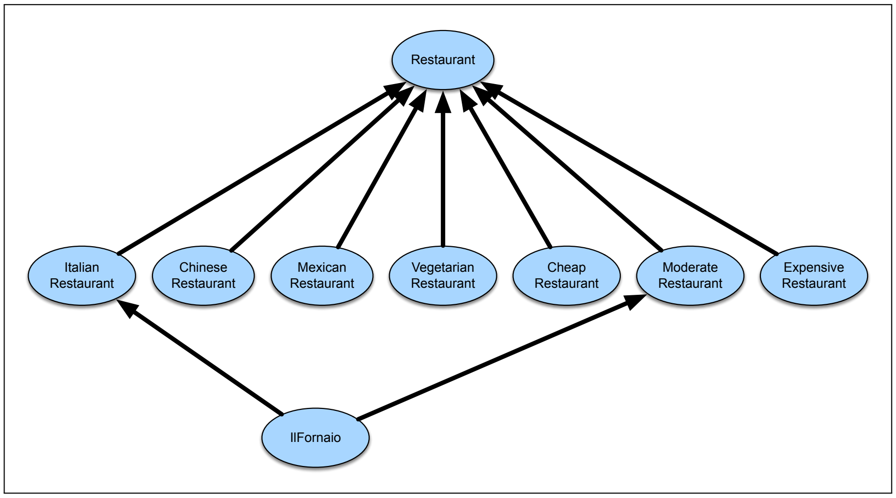


Figure 14.7 A graphical network representation of the complete set of subsumption relations in the restaurant domain given the current set of assertions in the TBox.

asserted about the terminology to infer appropriate subsumption and membership relations.

Returning to our restaurant domain, let's add a new kind of restaurant using the following statement:

$$\mathsf{IlFornaio} \sqsubseteq \mathsf{ModerateRestaurant} \sqcap \exists \mathsf{hasCuisine.ItalianCuisine} \tag{14.78}$$

Given this assertion, we might ask whether the IlFornaio chain of restaurants might be classified as an Italian restaurant or a vegetarian restaurant. More precisely, we can pose the following questions to our reasoning system:

$$\mathsf{IlFornaio} \sqsubseteq \mathsf{ItalianRestaurant} \tag{14.79}$$

$$\mathsf{IlFornaio} \sqsubseteq \mathsf{VegetarianRestaurant} \tag{14.80}$$

The answer to the first question is positive since IlFornaio meets the criteria we specified for the category ItalianRestaurant: it's a Restaurant since we explicitly classified it as a ModerateRestaurant, which is a subtype of Restaurant, and it meets the has.Cuisine class restriction since we've asserted that directly.

The answer to the second question is negative. Recall, that our criteria for vegetarian restaurants contains two requirements: it has to serve vegetarian fare, and that's all it can serve. Our current definition for IlFornaio fails on both counts since we have not asserted any relations that state that IlFornaio serves vegetarian fare, and the relation we have asserted, hasCuisine.ItalianCuisine, contradicts the second criteria.

implied hierarchy

A related reasoning task, based on the basic subsumption inference, is to derive the **implied hierarchy** for a terminology given facts about the categories in the terminology. This task roughly corresponds to a repeated application of the subsumption operator to pairs of concepts in the terminology. Given our current collection of statements, the expanded hierarchy shown in Fig. 14.7 can be inferred. You should convince yourself that this diagram contains all and only the subsumption links that should be present given our current knowledge.

Instance checking is the task of determining whether a particular individual can be classified as a member of a particular category. This process takes what is known about a given individual, in the form of relations and explicit categorical statements, and then compares that information with what is known about the current terminology. It then returns a list of *the most specific* categories to which the individual can belong.

As an example of a categorization problem, consider an establishment that we're told is a restaurant and serves Italian cuisine.

$$\mathsf{Restaurant}(\mathsf{Gondolier})$$
$$\mathsf{hasCuisine}(\mathsf{Gondolier}, \mathsf{ItalianCuisine})$$

Here, we're being told that the entity denoted by the term Gondolier is a restaurant and serves Italian food. Given this new information and the contents of our current TBox, we might reasonably like to ask if this is an Italian restaurant, if it is a vegetarian restaurant, or if it has moderate prices.

Assuming the definitional statements given earlier, we can indeed categorize the Gondolier as an Italian restaurant. That is, the information we've been given about it meets the necessary and sufficient conditions required for membership in this category. And as with the IlFornaio category, this individual fails to match the stated criteria for the VegetarianRestaurant. Finally, the Gondolier might also turn out to be a moderately priced restaurant, but we can't tell at this point since we don't know anything about its prices. What this means is that given our current knowledge the answer to the query $\mathsf{ModerateRestaurant}(\mathsf{Gondolier})$ would be false since it lacks the required hasPriceRange relation.

The implementation of subsumption, instance checking, as well as other kinds of inferences needed for practical applications, varies according to the expressivity of the Description Logic being used. However, for a Description Logic of even modest power, the primary implementation techniques are based on satisfiability methods that in turn rely on the underlying model-based semantics introduced earlier in this chapter.

OWL and the Semantic Web

The highest-profile role for Description Logics, to date, has been as a part of the development of the Semantic Web. The Semantic Web is an ongoing effort to provide a way to formally specify the semantics of the contents of the Web (Fensel et al., 2003). A key component of this effort involves the creation and deployment of ontologies for various application areas of interest. The meaning representation language used to represent this knowledge is the **Web Ontology Language** (OWL) (McGuiness and van Harmelen, 2004). OWL embodies a Description Logic that corresponds roughly to the one we've been describing here.

Web Ontology Language

14.6 Summary

This chapter has introduced the representational approach to meaning. The following are some of the highlights of this chapter:

- A major approach to meaning in computational linguistics involves the creation of **formal meaning representations** that capture the meaning-related content of linguistic inputs. These representations are intended to bridge the gap from language to common-sense knowledge of the world.

- The frameworks that specify the syntax and semantics of these representations are called **meaning representation languages**. A wide variety of such languages are used in natural language processing and artificial intelligence.
- Such representations need to be able to support the practical computational requirements of semantic processing. Among these are the need to determine **the truth of propositions**, to support **unambiguous representations**, to represent **variables**, to support **inference**, and to be sufficiently **expressive**.
- Human languages have a wide variety of features that are used to convey meaning. Among the most important of these is the ability to convey a **predicate-argument structure**.
- **First-Order Logic** is a well-understood, computationally tractable meaning representation language that offers much of what is needed in a meaning representation language.
- Important elements of semantic representation including **states** and **events** can be captured in FOL.
- **Semantic networks** and **frames** can be captured within the FOL framework.
- Modern **Description Logics** consist of useful and computationally tractable subsets of full First-Order Logic. The most prominent use of a description logic is the **Web Ontology Language** (OWL), used in the specification of the Semantic Web.

Bibliographical and Historical Notes

The earliest computational use of declarative meaning representations in natural language processing was in the context of question-answering systems (Green et al., 1961; Raphael, 1968; Lindsey, 1963). These systems employed ad hoc representations for the facts needed to answer questions. Questions were then translated into a form that could be matched against facts in the knowledge base. Simmons (1965) provides an overview of these early efforts.

Woods (1967) investigated the use of FOL-like representations in question answering as a replacement for the ad hoc representations in use at the time. Woods (1973) further developed and extended these ideas in the landmark Lunar system. Interestingly, the representations used in Lunar had both truth-conditional and procedural semantics. Winograd (1972) employed a similar representation based on the Micro-Planner language in his SHRDLU system.

During this same period, researchers interested in the cognitive modeling of language and memory had been working with various forms of associative network representations. Masterman (1957) was the first to make computational use of a semantic network-like knowledge representation, although semantic networks are generally credited to Quillian (1968). A considerable amount of work in the semantic network framework was carried out during this era (Norman and Rumelhart, 1975; Schank, 1972; Wilks, 1975b, 1975a; Kintsch, 1974). It was during this period that a number of researchers began to incorporate Fillmore's notion of case roles (Fillmore, 1968) into their representations. Simmons (1973) was the earliest adopter of case roles as part of representations for natural language processing.

Detailed analyses by Woods (1975) and Brachman (1979) aimed at figuring out what semantic networks actually mean led to the development of a number of more sophisticated network-like languages including KRL (Bobrow and Winograd, 1977) and KL-ONE (Brachman and Schmolze, 1985). As these frameworks became more sophisticated and well defined, it became clear that they were restricted variants of FOL coupled with specialized indexing inference procedures. A useful collection of papers covering much of this work can be found in Brachman and Levesque (1985). Russell and Norvig (2002) describe a modern perspective on these representational efforts.

Linguistic efforts to assign semantic structures to natural language sentences in the generative era began with the work of Katz and Fodor (1963). The limitations of their simple feature-based representations and the natural fit of logic to many of the linguistic problems of the day quickly led to the adoption of a variety of predicate-argument structures as preferred semantic representations (Lakoff, 1972; McCawley, 1968). The subsequent introduction by Montague (1973) of the truth-conditional model-theoretic framework into linguistic theory led to a much tighter integration between theories of formal syntax and a wide range of formal semantic frameworks. Good introductions to Montague semantics and its role in linguistic theory can be found in Dowty et al. (1981) and Partee (1976).

The representation of events as reified objects is due to Davidson (1967). The approach presented here, which explicitly reifies event participants, is due to Parsons (1990).

Most current computational approaches to temporal reasoning are based on Allen's notion of temporal intervals (Allen, 1984); see Chapter 17. ter Meulen (1995) provides a modern treatment of tense and aspect. Davis (1990) describes the use of FOL to represent knowledge across a wide range of common-sense domains including quantities, space, time, and beliefs.

A recent comprehensive treatment of logic and language can be found in van Benthem and ter Meulen (1997). A classic semantics text is Lyons (1977). McCawley (1993) is an indispensable textbook covering a wide range of topics concerning logic and language. Chierchia and McConnell-Ginet (1991) also broadly covers semantic issues from a linguistic perspective. Heim and Kratzer (1998) is a more recent text written from the perspective of current generative theory.

Exercises

14.1 Peruse your daily newspaper for three examples of ambiguous sentences or headlines. Describe the various sources of the ambiguities.

14.2 Consider a domain in which the word *coffee* can refer to the following concepts in a knowledge-based system: a caffeinated or decaffeinated beverage, ground coffee used to make either kind of beverage, and the beans themselves. Give arguments as to which of the following uses of coffee are ambiguous and which are vague.

1. I've had my coffee for today.
2. Buy some coffee on your way home.
3. Please grind some more coffee.

14.3 The following rule, which we gave as a translation for Example 14.26, is not a reasonable definition of what it means to be a vegetarian restaurant.

$$\forall x VegetarianRestaurant(x) \implies Serves(x, VegetarianFood)$$

Give a FOL rule that better defines vegetarian restaurants in terms of what they serve.

14.4 Give FOL translations for the following sentences:

1. Vegetarians do not eat meat.
2. Not all vegetarians eat eggs.

14.5 Give a set of facts and inferences necessary to prove the following assertions:

1. McDonald's is not a vegetarian restaurant.
2. Some vegetarians can eat at McDonald's.

Don't just place these facts in your knowledge base. Show that they can be inferred from some more general facts about vegetarians and McDonald's.

14.6 For the following sentences, give FOL translations that capture the temporal relationships between the events.

1. When Mary's flight departed, I ate lunch.
2. When Mary's flight departed, I had eaten lunch.

14.7 On page 15, we gave the representation $Near(Centro, Bacaro)$ as a translation for the sentence *Centro is near Bacaro*. In a truth-conditional semantics, this formula is either true or false given some model. Critique this truth-conditional approach with respect to the meaning of words like *near*.

Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, *23*(2), 123–154.

Bobrow, D. G. and Winograd, T. (1977). An overview of KRL, a knowledge representation language. *Cognitive Science*, *1*(1), 3–46.

Brachman, R. J. (1979). On the epistemogical status of semantic networks. In Findler, N. V. (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, pp. 3–50. Academic Press.

Brachman, R. J. and Levesque, H. J. (Eds.). (1985). *Readings in Knowledge Representation*. Morgan Kaufmann.

Brachman, R. J. and Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, *9*(2), 171–216.

Chierchia, G. and McConnell-Ginet, S. (1991). *Meaning and Grammar*. MIT Press.

Church, A. (1940). A formulation of a simple theory of types. *Journal of Symbolic Logic*, *5*, 56–68.

Davidson, D. (1967). The logical form of action sentences. In Rescher, N. (Ed.), *The Logic of Decision and Action*. University of Pittsburgh Press.

Davis, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann.

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

Fensel, D., Hendler, J. A., Lieberman, H., and Wahlster, W. (Eds.). (2003). *Spinning the Semantic Web: Bring the World Wide Web to its Full Potential*. MIT Press, Cambridge, MA.

Fillmore, C. J. (1968). The case for case. In Bach, E. W. and Harms, R. T. (Eds.), *Universals in Linguistic Theory*, pp. 1–88. Holt, Rinehart & Winston.

Green, B. F., Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference 19*, pp. 219–224. Reprinted in Grosz et al. (1986).

Heim, I. and Kratzer, A. (1998). *Semantics in a Generative Grammar*. Blackwell Publishers, Malden, MA.

Jackendoff, R. (1983). *Semantics and Cognition*. MIT Press.

Katz, J. J. and Fodor, J. A. (1963). The structure of a semantic theory. *Language*, *39*, 170–210.

Kintsch, W. (1974). *The Representation of Meaning in Memory*. Wiley, New York.

Lakoff, G. (1972). Linguistics and natural logic. In Davidson, D. and Harman, G. (Eds.), *Semantics for Natural Language*, pp. 545–665. D. Reidel.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press, Chicago, IL.

Lindsey, R. (1963). Inferential memory as the basis of machines which understand natural language. In Feigenbaum, E. and Feldman, J. (Eds.), *Computers and Thought*, pp. 217–233. McGraw Hill.

Lyons, J. (1977). *Semantics*. Cambridge University Press.

Masterman, M. (1957). The thesaurus in syntax and semantics. *Mechanical Translation*, *4*(1), 1–2.

McCawley, J. D. (1968). The role of semantics in a grammar. In Bach, E. W. and Harms, R. T. (Eds.), *Universals in Linguistic Theory*, pp. 124–169. Holt, Rinehart & Winston.

McCawley, J. D. (1993). *Everything that Linguists Have Always Wanted to Know about Logic* (2nd Ed.). University of Chicago Press, Chicago, IL.

McGuiness, D. L. and van Harmelen, F. (2004). OWL web ontology overview. Tech. rep. 20040210, World Wide Web Consortium.

Montague, R. (1973). The proper treatment of quantification in ordinary English. In Thomason, R. (Ed.), *Formal Philosophy: Selected Papers of Richard Montague*, pp. 247–270. Yale University Press, New Haven, CT.

Norman, D. A. and Rumelhart, D. E. (1975). *Explorations in Cognition*. Freeman.

Parsons, T. (1990). *Events in the Semantics of English*. MIT Press.

Partee, B. H. (Ed.). (1976). *Montague Grammar*. Academic Press.

Quillian, M. R. (1968). Semantic memory. In Minsky, M. (Ed.), *Semantic Information Processing*, pp. 227–270. MIT Press.

Raphael, B. (1968). SIR: A computer program for semantic information retrieval. In Minsky, M. (Ed.), *Semantic Information Processing*, pp. 33–145. MIT Press.

Reichenbach, H. (1947). *Elements of Symbolic Logic*. Macmillan, New York.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach* (2nd Ed.). Prentice Hall.

Schank, R. C. (1972). Conceptual dependency: A theory of natural language processing. *Cognitive Psychology*, *3*, 552–631.

Schönkfinkel, M. (1924). Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, *92*, 305–316. English translation appears in *From Frege to Gödel: A Source Book in Mathematical Logic*, Harvard University Press, 1967.

Simmons, R. F. (1965). Answering English questions by computer: A survey. *Communications of the ACM*, *8*(1), 53–70.

Simmons, R. F. (1973). Semantic networks: Their computation and use for understanding English sentences. In Schank, R. C. and Colby, K. M. (Eds.), *Computer Models of Thought and Language*, pp. 61–113. W.H. Freeman and Co.

ter Meulen, A. (1995). *Representing Time in Natural Language*. MIT Press.

van Benthem, J. and ter Meulen, A. (Eds.). (1997). *Handbook of Logic and Language*. MIT Press.

Vendler, Z. (1967). *Linguistics in Philosophy*. Cornell University Press, Ithaca, NY.

Wilks, Y. (1975a). Preference semantics. In Keenan, E. L. (Ed.), *The Formal Semantics of Natural Language*, pp. 329–350. Cambridge Univ. Press.

Wilks, Y. (1975b). A preferential, pattern-seeking, semantics for natural language inference. *Artificial Intelligence*, *6*(1), 53–74.

Winograd, T. (1972). *Understanding Natural Language*. Academic Press.

Woods, W. A. (1967). *Semantics for a Question-Answering System*. Ph.D. thesis, Harvard University.

Woods, W. A. (1973). Progress in natural language understanding. In *Proceedings of AFIPS National Conference*, pp. 441–450.

Woods, W. A. (1975). What's in a link: Foundations for semantic networks. In Bobrow, D. G. and Collins, A. M. (Eds.), *Representation and Understanding: Studies in Cognitive Science*, pp. 35–82. Academic Press.

Zwicky, A. and Sadock, J. M. (1975). Ambiguity tests and how to fail them. In Kimball, J. (Ed.), *Syntax and Semantics 4*, pp. 1–36. Academic Press.