CHAPTER

19 Logical Representations of Sentence Meaning

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

meaning representations

In this chapter we introduce the idea that the meaning of linguistic expressions can be captured in formal structures called **meaning representations**. Consider tasks that require some form of semantic processing, like learning to use a new piece of software by reading the manual, deciding what to order at a restaurant by reading a menu, or following a recipe. Accomplishing these tasks requires representations that link the linguistic elements to the necessary non-linguistic *knowledge of the world*. 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. Learning to use a piece of software by reading a manual, or giving advice on using software, requires knowledge about the software and similar apps, computers, and 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 of everyday common-sense knowledge of the world. The process whereby such representations are created and assigned to linguistic inputs is called **semantic parsing** or **semantic analysis**, and the entire enterprise of designing meaning representations and associated semantic parsers is referred to as **computational semantics**.

semantic parsing

computational semantics

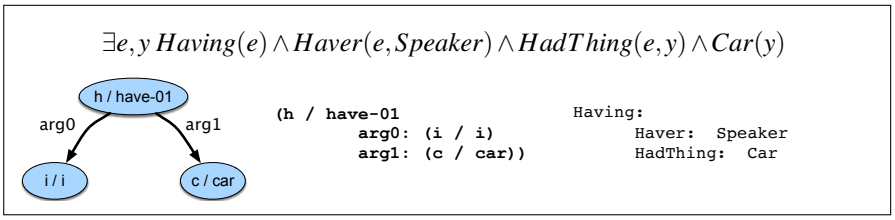

Figure 19.1 A list of symbols, two directed graphs, and a record structure: a sampler of meaning representations for *I have a car.*

Consider Fig. 19.1, 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 19.3; the directed graph and its corresponding textual form is an example of an **Abstract Meaning Representation (AMR)** form (Banarescu et al., 2013), and on the right is a **frame-based** or **slot-filler** representation, discussed in Section 19.5 and again in Chapter 21.

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.

In the next sections we give some background: our desiderata for a meaning representation language and some guarantees that these representations will actually do what we need them to do—provide a correspondence to the state of affairs being represented. In Section 19.3 we introduce First-Order Logic, historically the primary technique for investigating natural language semantics, and see in Section 19.4 how it can be used to capture the semantics of events and states in English. Chapter 20 then introduces techniques for **semantic parsing**: generating these formal meaning representations given linguistic inputs.

19.1 Computational Desiderata for Representations

Let's consider why meaning representations are needed and what they should do for us. To focus this discussion, let's consider a system that gives restaurant advice to tourists based on a knowledge base.

Verifiability

Consider the following simple question:

(19.1) Does Maharani serve vegetarian food?

verifiability

To answer this question, we have to know what it's asking, and know whether what it's asking is true of Maharini or not. **Verifiability** is 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. For example, we'll need some sort of representation like $Serves(Maharani, VegetarianFood)$, which a system can match against its knowledge base of facts about particular restaurants, and if it finds a representation matching this proposition, it can answer yes. Otherwise, it must either say *No* if its knowledge of local restaurants is complete, or say that it doesn't know if it knows its knowledge is incomplete.

Unambiguous Representations

Semantics, like all the other domains we have studied, is subject to ambiguity. Words and sentences have different meaning representations in different contexts. Consider the following example:

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

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 sentence is ambiguous; a single linguistic expression can have one of two meanings. But our *meaning representations* itself cannot be ambiguous. The representation of an input's meaning should be free from any ambiguity, so that the system can reason over a representation that means either one thing or the other in order to decide how to answer.

vagueness

A concept closely related to ambiguity is **vagueness**: in which a meaning representation leaves some parts of the meaning underspecified. Vagueness does not give rise to multiple representations. Consider the following request:

(19.3) I want to eat Italian food.

While *Italian food* may provide enough information to provide recommendations, it is nevertheless *vague* as to what the user really wants to eat. 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.

Canonical Form

canonical form

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

Consider the following alternative ways of expressing (19.1):

(19.4) Does Maharani have vegetarian dishes?

(19.5) Do they have vegetarian food at Maharani?

(19.6) Are vegetarian dishes served at Maharani?

(19.7) Does Maharani serve vegetarian fare?

Despite the fact these alternatives use different words and syntax, we want them to map to a single canonical meaning representations. If they were all different, assuming the system's knowledge base contains only a single representation of this fact, most of the representations wouldn't match. We could, of course, store all possible alternative representations of the same fact in the knowledge base, but doing so would lead to enormous difficulty in keeping the knowledge base consistent.

Canonical form does complicate the task of semantic parsing. Our system must conclude that *vegetarian fare*, *vegetarian dishes*, and *vegetarian food* refer to the same thing, that *having* and *serving* are equivalent here, and that all these parse structures still lead to the same meaning representation. Or consider this pair of examples:

(19.8) Maharani serves vegetarian dishes.

(19.9) Vegetarian dishes are served by Maharani.

Despite the different placement of the arguments to *serve*, a system must still assign *Maharani* and *vegetarian dishes* to the same roles in the two examples by drawing on grammatical knowledge, such as the relationship between active and passive sentence constructions.

Inference and Variables

What about more complex requests such as:

(19.10) Can vegetarians eat at Maharani?

This request results in the same answer as the others 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. We'll need to connect the meaning representation of this request with this fact about the world in a knowledge base. A system must be able to use **inference**—to draw valid conclusions based on the meaning representation of inputs and its 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.

inference

Now consider the following somewhat more complex request:

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

This request does not make reference to any particular restaurant; the user wants information about an unknown restaurant that serves vegetarian food. Since no restaurants are named, simple matching is not going to work. Answering this request requires the use of **variables**, using some representation like the following:

variables

$$Serves(x, VegetarianFood) \tag{19.12}$$

Matching succeeds only if the variable x can be replaced by some 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. It is critical for any meaning representation language to be able to handle these kinds of indefinite references.

Expressiveness

Finally, a meaning representation scheme must be expressive enough to handle a wide range of subject matter, ideally 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 19.3, is expressive enough to handle quite a lot of what needs to be represented.

19.2 Model-Theoretic Semantics

What is it about meaning representation languages that allows them to fulfill these desiderata, bridging the gap from formal representations to representations that tell us something about some state of affairs in the world?

model

The answer is 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 to elements of the model, like objects, properties of objects, and relations among objects. If the model accurately captures the facts we're interested in, then a consistent mapping between the meaning representation and the model provides the bridge between meaning representation and world. Models provide a surprisingly simple and powerful way to ground the expressions in meaning representation languages.

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,

non-logical vocabulary

logical vocabulary

etc., that provide the formal means for composing expressions in a given meaning representation language.

denotation

Each element of the non-logical vocabulary must have a **denotation** in the model, meaning that every element corresponds to a fixed, well-defined part of the model.

domain

Let's start with objects. The **domain** of a model is the set of objects that are being represented. Each distinct concept, category, or individual denotes a unique element in the domain.

We represent *properties* of objects in a model by denoting the domain elements that have the property; that is, properties denote sets. The denotation of the property *red* is the set of things we think are red. Similarly, a relation among object denotes a set of ordered lists, or tuples, of domain elements that take part in the relation: the denotation of the relation *Married* is set of pairs of domain objects that are married.

extensional

This approach to properties and relations is called **extensional**, because we define concepts by their extension, their denotations. 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

We now need a mapping that gets us from our meaning representation to the corresponding denotations: 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

Let's return to our restaurant advice application, and let its domain consist of sets of restaurants, patrons, 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*. denoting four unique domain elements. We'll use the constants a, b, c and, d to stand for these domain elements. 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.

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. 19.2.

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

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 19.2 A model of the restaurant world.

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:

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

(19.14) Katie and Caroline like the same restaurants.

(19.15) Franco likes noisy, expensive restaurants.

(19.16) 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.

Consider the first example above. A meaning representation for this example 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**

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

Figure 19.3 A context-free grammar specification of the syntax of First-Order Logic representations. Adapted from Russell and Norvig 2002.

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 meaning 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 depend 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 in the context of the semantics of First-Order Logic.

19.3 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 19.1. 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 few specific commitments as to how things ought to be represented, and those it does 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.

19.3.1 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 19.3,

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. 19.3, 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.

constant **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.

function **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.

$$LocationOf(Frasca) \tag{19.17}$$

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 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 19.1, a reasonable FOL representation for *Maharani serves vegetarian food* might look like the following formula:

$$Serves(Maharani, VegetarianFood) \tag{19.18}$$

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*:

$$Restaurant(Maharani) \tag{19.19}$$

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. 19.3. This ability

logical connectives

to compose complex representations is, however, not limited to the use of single predicates. Larger composite representations can also be put together through the use of **logical connectives**. As can be seen from Fig. 19.3, 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:

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

$$Have(Speaker, FiveDollars) \wedge \neg Have(Speaker, LotOfTime) \tag{19.21}$$

The semantic representation for this example is built up in a straightforward way from the semantics of the individual clauses through the use of the $\wedge$ and $\neg$ operators. Note that the recursive nature of the grammar in Fig. 19.3 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.

19.3.2 Variables and Quantifiers

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".

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

(19.22) 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:

$$\begin{aligned} \exists x Restaurant(x) \ &\wedge \ Serves(x, MexicanFood) \\ &\wedge \ Near((LocationOf(x), LocationOf(ICSI)) \end{aligned} \tag{19.23}$$

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:

$$\begin{aligned} Restaurant(AyCaramba) \wedge Serves(AyCaramba, MexicanFood) \\ \wedge Near((LocationOf(AyCaramba), LocationOf(ICSI)) \end{aligned} \tag{19.24}$$

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:

(19.25) All vegetarian restaurants serve vegetarian food.

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

$$\forall x VegetarianRestaurant(x) \implies Serves(x, VegetarianFood) \tag{19.26}$$

For this sentence to be true, 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:

$$VegetarianRestaurant(Maharani) \implies Serves(Maharani, VegetarianFood) \tag{19.27}$$

If we assume that we know that the consequent clause

$$Serves(Maharani, VegetarianFood) \tag{19.28}$$

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. 19.4 on page 12 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 *AyCaramba* for the variable x:

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

Since the antecedent of the implication is *False*, we can determine from Fig. 19.4 that the sentence is always *True*, again satisfying the $\forall$ constraint.

Note that it may still be the case that *AyCaramba* 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 so 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. To satisfy a universally quantified variable, all substitutions must result in true sentences.

19.3.3 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 formulas 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:

$$\lambda x.P(x) \tag{19.29}$$

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 and the 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:

$$\begin{gathered}\lambda x.P(x)(A) \\ P(A)\end{gathered} \tag{19.30}$$

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:

$$\lambda x.\lambda y.Near(x,y) \tag{19.31}$$

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:

$$\begin{gathered}\lambda x.\lambda y.Near(x,y)(Bacaro) \\ \lambda y.Near(Bacaro,y)\end{gathered} \tag{19.32}$$

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:

$$\begin{gathered}\lambda y.Near(Bacaro,y)(Centro) \\ Near(Bacaro,Centro)\end{gathered} \tag{19.33}$$

currying

This general technique, called **currying**[1] (Schönfinkel, 1924) is a way of converting a predicate with multiple arguments into a sequence of single-argument predicates.

As we show in Chapter 20, 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.

19.3.4 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

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

relations out in the external world being modeled. We can accomplish this by employing the model-theoretic approach introduced in Section 19.2. 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. 19.3 on page 7 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:

(19.34) 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:

$$Near(Centro, Bacaro) \tag{19.35}$$

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 interpretation of formulas involving logical connectives is based on the meanings of the components in the formulas combined with the meanings of the connectives they contain. Figure 19.4 gives interpretations for each of the logical operators shown in Fig. 19.3.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Longrightarrow Q$
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 19.4 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 $\Longrightarrow$ (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 9. 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.

19.3.5 Inference

A meaning representation language must support inference to add valid new propositions to a knowledge base or to determine the truth of propositions not explicitly

contained within a knowledge base (Section 19.1). This section briefly discusses **modus ponens**, the most widely implemented inference method provided by FOL.

Modus ponens

Modus ponens is a 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:

$$\frac{\alpha \quad \alpha \implies \beta}{\beta} \tag{19.36}$$

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 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:

$$\frac{VegetarianRestaurant(Leaf) \quad \forall x VegetarianRestaurant(x) \implies Serves(x, VegetarianFood)}{Serves(Leaf, VegetarianFood)} \tag{19.37}$$

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. 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 (19.37). 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 27, 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

19.4 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.

$$Serves(Leaf, VegetarianFare) \tag{19.38}$$

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*:

(19.39) I ate.

(19.40) I ate a turkey sandwich.

(19.41) I ate a turkey sandwich at my desk.

(19.42) I ate at my desk.

(19.43) I ate lunch.

(19.44) I ate a turkey sandwich for lunch.

(19.45) 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 19.40 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 19.45.

$$\begin{aligned}\exists e\, Eating(e) \;&\wedge\; Eater(e, Speaker) \wedge Eaten(e, TurkeySandwich) \\ &\wedge\; Meal(e, Lunch) \wedge Location(e, Desk)\end{aligned} \tag{19.46}$$

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 22 and Chapter 24.

19.5 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 using unary predicates such as $Restaurant(x)$; the DL equivalent omits the variable, so the restaurant category is simply written as Restaurant.[2] To capture the fact that a particular domain element, such as *Frasca*, is a restaurant, we assert Restaurant(Frasca) in much the same way we would in FOL. The semantics of these categories are specified in precisely the same way that was introduced earlier in Section 19.2: 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.

$$\mathsf{Restaurant} \sqsubseteq \mathsf{CommercialEstablishment} \tag{19.47}$$
$$\mathsf{ItalianRestaurant} \sqsubseteq \mathsf{Restaurant} \tag{19.48}$$
$$\mathsf{ChineseRestaurant} \sqsubseteq \mathsf{Restaurant} \tag{19.49}$$
$$\mathsf{MexicanRestaurant} \sqsubseteq \mathsf{Restaurant} \tag{19.50}$$

Ontologies such as this are conventionally illustrated with diagrams such as the one

[2] 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.

shown in Fig. 19.5, where subsumption relations are denoted by links between the nodes representing the categories.

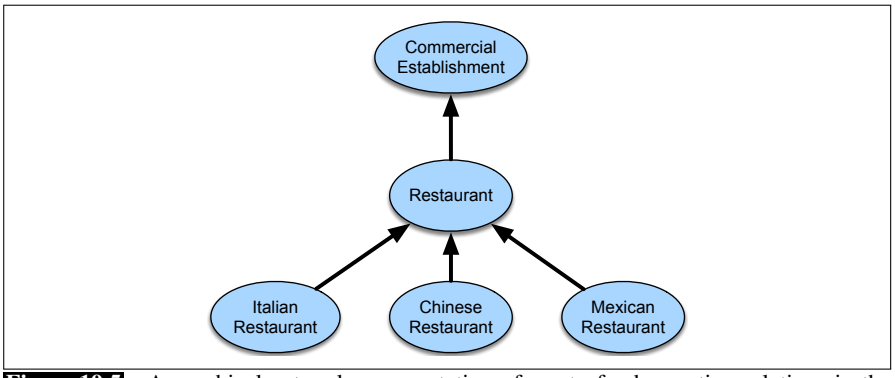

Figure 19.5 A graphical network representation of a set of subsumption relations in the restaurant domain.

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.

$$\mathsf{ChineseRestaurant} \sqsubseteq \textbf{not}\ \mathsf{ItalianRestaurant} \tag{19.51}$$

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

$$\mathsf{Restaurant} \sqsubseteq (\textbf{or}\ \mathsf{ItalianRestaurant}\ \mathsf{ChineseRestaurant}\ \mathsf{MexicanRestaurant}) \tag{19.52}$$

Having a hierarchy such as the one given in Fig. 19.5 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.

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

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

$$\mathsf{ItalianRestaurant} \sqsubseteq \mathsf{Restaurant} \sqcap \exists \mathsf{hasCuisine}.\mathsf{ItalianCuisine} \tag{19.53}$$

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

$$\begin{aligned} \forall x \mathit{ItalianRestaurant}(x) \rightarrow\ & \mathit{Restaurant}(x) \\ & \wedge (\exists y \mathit{Serves}(x,y) \wedge \mathit{ItalianCuisine}(y)) \end{aligned} \tag{19.54}$$

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.

$$\mathsf{ItalianRestaurant} \equiv \mathsf{Restaurant} \sqcap \exists \mathsf{hasCuisine}.\mathsf{ItalianCuisine} \tag{19.55}$$

$$\mathsf{ModerateRestaurant} \equiv \mathsf{Restaurant} \sqcap \mathsf{hasPriceRange}.\mathsf{ModeratePrices} \tag{19.56}$$

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:

$$\begin{aligned} \text{VegetarianRestaurant} \equiv\ & \text{Restaurant} \\ & \sqcap \exists \text{hasCuisine.VegetarianCuisine} \\ & \sqcap \forall \text{hasCuisine.VegetarianCuisine} \end{aligned} \tag{19.57}$$

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

instance checking

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 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:

$$\text{IlFornaio} \sqsubseteq \text{ModerateRestaurant} \sqcap \exists \text{hasCuisine.ItalianCuisine} \tag{19.58}$$

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:

$$\text{IlFornaio} \sqsubseteq \text{ItalianRestaurant} \tag{19.59}$$

$$\text{IlFornaio} \sqsubseteq \text{VegetarianRestaurant} \tag{19.60}$$

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. 19.6 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

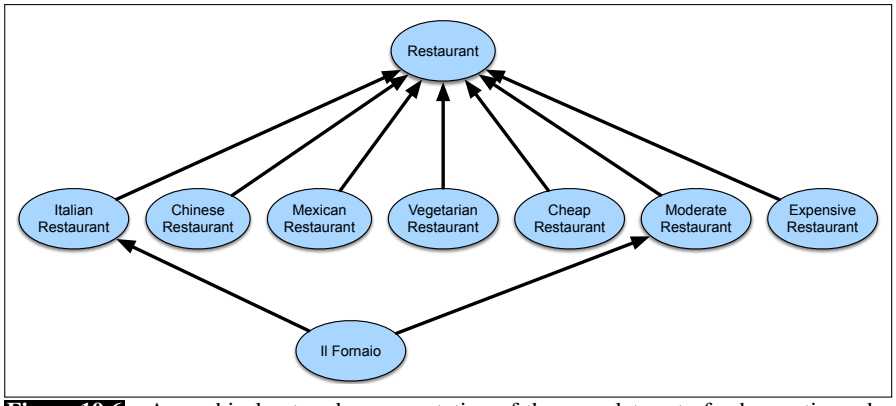

Figure 19.6 A graphical network representation of the complete set of subsumption relations in the restaurant domain given the current set of assertions in the TBox.

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

Web Ontology Language

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.

19.6 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, Wilks 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).

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

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

19.2 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.

19.3 The following rule, which we gave as a translation for Example 19.25, is not a reasonable definition of what it means to be a vegetarian restaurant.

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

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

19.4 Give FOL translations for the following sentences:

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

19.5 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.

19.6 On page 12, we gave the representation $\mathit{Near}(\mathit{Centro}, \mathit{Bacaro})$ 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*.