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 use of **logical connectives**. As can be seen from Fig. 14.3, logical connectives let

logical connectives

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 309 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.