8.1 Representation Revisited

In this section, we discuss the nature of representation languages. Programming languages (such as C++ or Java or Python) are the largest class of formal languages in common use. Data structures within programs can be used to represent facts; for example, a program could use a 4×4 array to represent the contents of the wumpus world. Thus, the programming language statement $World[2,2] \leftarrow Pit$ is a fairly natural way to assert that there is a pit in square [2,2]. Putting together a string of such statements is sufficient for running a simulation of the wumpus world.

What programming languages lack is a general mechanism for deriving facts from other facts; each update to a data structure is done by a domain-specific procedure whose details are derived by the programmer from his or her own knowledge of the domain. This procedural approach can be contrasted with the **declarative** nature of propositional logic, in which knowledge and inference are separate, and inference is entirely domain independent. SQL databases take a mix of declarative and procedural knowledge.

A second drawback of data structures in programs (and of databases) is the lack of any easy way to say, for example, "There is a pit in [2,2] or [3,1]" or "If the wumpus is in [1,1] then he is not in [2,2]." Programs can store a single value for each variable, and some systems allow the value to be "unknown," but they lack the expressiveness required to directly handle partial information.

Propositional logic is a declarative language because its semantics is based on a truth relation between sentences and possible worlds. It also has sufficient expressive power to deal with partial information, using disjunction and negation. Propositional logic has a third property that is desirable in representation languages, namely, **compositionality**. In a compositional language, the meaning of a sentence is a function of the meaning of its parts. For example, the meaning of "$S_{1,4} \wedge S_{1,2}$" is related to the meanings of "$S_{1,4}$" and "$S_{1,2}$". It would be very strange if "$S_{1,4}$" meant that there is a stench in square [1,4] and "$S_{1,2}$" meant that there is a stench in square [1,2], but "$S_{1,4} \wedge S_{1,2}$" meant that France and Poland drew 1–1 in last week's ice hockey qualifying match.

However, propositional logic, as a factored representation, lacks the expressive power to *concisely* describe an environment with many objects. For example, we were forced to write a separate rule about breezes and pits for each square, such as

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}).$$

In English, on the other hand, it seems easy enough to say, once and for all, "Squares adjacent to pits are breezy." The syntax and semantics of English make it possible to describe the environment concisely: English, like first-order logic, is a structured representation.

8.1.1 The language of thought

Natural languages (such as English or Spanish) are very expressive indeed. We managed to write almost this whole book in natural language, with only occasional lapses into other languages (mainly mathematics and diagrams). There is a long tradition in linguistics and the philosophy of language that views natural language as a declarative knowledge representation language. If we could uncover the rules for natural language, we could use them in representation and reasoning systems and gain the benefit of the billions of pages that have been written in natural language.

The modern view of natural language is that it serves as a medium for *communication* rather than pure representation. When a speaker points and says, "Look!" the listener comes to know that, say, Superman has finally appeared over the rooftops. Yet we would not want to say that the sentence "Look!" represents that fact. Rather, the meaning of the sentence depends both on the sentence itself and on the *context* in which the sentence was spoken. Clearly, one could not store a sentence such as "Look!" in a knowledge base and expect to recover its meaning without also storing a representation of the context—which raises the question of how the context itself can be represented.

Natural languages also suffer from *ambiguity*, a problem for a representation language. As Pinker (1995) puts it: "When people think about *spring*, surely they are not confused as to

whether they are thinking about a season or something that goes *boing*—and if one word can correspond to two thoughts, thoughts can't be words."

The famous **Sapir–Whorf hypothesis** Whorf (1956) claims that our understanding of the world is strongly influenced by the language we speak. It is certainly true that different speech communities divide up the world differently. The French have two words "chaise" and "fauteuil," for a concept that English speakers cover with one: "chair." But English speakers can easily recognize the category *fauteuil* and give it a name—roughly "open-arm chair"—so does language really make a difference? Whorf relied mainly on intuition and speculation, and his ideas have been largely dismissed, but in the intervening years we actually have real data from anthropological, psychological, and neurological studies.

For example, can you remember which of the following two phrases formed the opening of Section 8.1⬚?

> "In this section, we discuss the nature of representation languages . . ."

> "This section covers the topic of knowledge representation languages . . ."

Wanner (1974) did a similar experiment and found that subjects made the right choice at chance level—about 50% of the time—but remembered the content of what they read with better than 90% accuracy. This suggests that people interpret the words they read and form an internal *nonverbal* representation, and that the exact words are not consequential.

More interesting is the case in which a concept is completely absent in a language. Speakers of the Australian aboriginal language Guugu Yimithirr have no words for relative (or *egocentric*) directions, such as front, back, right, or left. Instead they use absolute directions, saying, for example, the equivalent of "I have a pain in my north arm." This difference in language makes a difference in behavior: Guugu Yimithirr speakers are better at navigating in open terrain, while English speakers are better at placing the fork to the right of the plate.

Language also seems to influence thought through seemingly arbitrary grammatical features such as the gender of nouns. For example, "bridge" is masculine in Spanish and feminine in German. Boroditsky (2003) asked subjects to choose English adjectives to describe a

photograph of a particular bridge. Spanish speakers chose *big, dangerous, strong*, and *towering*, whereas German speakers chose *beautiful, elegant, fragile*, and *slender*.

Words can serve as anchor points that affect how we perceive the world. Loftus and Palmer (1974) showed experimental subjects a movie of an auto accident. Subjects who were asked "How fast were the cars going when they contacted each other?" reported an average of 32 mph, while subjects who were asked the question with the word "smashed" instead of "contacted" reported 41mph for the same cars in the same movie. Overall, there are measurable but small differences in cognitive processing by speakers of different languages, but no convincing evidence that this leads to a major difference in world view.

In a logical reasoning system that uses conjunctive normal form (CNF), we can see that the linguistic forms "$\neg(A \vee B)$" and "$\neg A \wedge \neg B$" are the same because we can look inside the system and see that the two sentences are stored as the same canonical CNF form. It is starting to become possible to do something similar with the human brain. Mitchell *et al.* (2008) put subjects in an functional magnetic resonance imaging (fMRI) machine, showed them words such as "celery," and imaged their brains. A machine learning program trained on (word, image) pairs was able to predict correctly 77% of the time on binary choice tasks (e.g., "celery" or "airplane"). The system can even predict at above-chance levels for words it has never seen an fMRI image of before (by considering the images of related words) and for people it has never seen before (proving that fMRI reveals some level of common representation across people). This type of work is still in its infancy, but fMRI (and other imaging technology such as intracranial electrophysiology (Sahin *et al.*, 2009)) promises to give us much more concrete ideas of what human knowledge representations are like.

From the viewpoint of formal logic, representing the same knowledge in two different ways makes absolutely no difference; the same facts will be derivable from either representation. In practice, however, one representation might require fewer steps to derive a conclusion, meaning that a reasoner with limited resources could get to the conclusion using one representation but not the other. For *nondeductive* tasks such as learning from experience, outcomes are *necessarily* dependent on the form of the representations used. We show in Chapter 19 that when a learning program considers two possible theories of the world, both of which are consistent with all the data, the most common way of breaking the tie is to choose the most succinct theory—and that depends on the language used to represent

theories. Thus, the influence of language on thought is unavoidable for any agent that does learning.

8.1.2 Combining the best of formal and natural languages

We can adopt the foundation of propositional logic—a declarative, compositional semantics that is context-independent and unambiguous—and build a more expressive logic on that foundation, borrowing representational ideas from natural language while avoiding its drawbacks. When we look at the syntax of natural language, the most obvious elements are nouns and noun phrases that refer to **objects** (squares, pits, wumpuses) and verbs and verb phrases along with adjectives and adverbs that refer to **relations** among objects (is breezy, is adjacent to, shoots). Some of these relations are **functions**—relations in which there is only one "value" for a given "input." It is easy to start listing examples of objects, relations, and functions:

Object

Relation

Function

- Objects: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries …
- Relations: these can be unary relations or **properties** such as red, round, bogus, prime, multistoried …, or more general n-ary relations such as brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, …

Property

- Functions: father of, best friend, third inning of, one more than, beginning of …

Indeed, almost any assertion can be thought of as referring to objects and properties or relations. Some examples follow:

- "One plus two equals three." Objects: one, two, three, one plus two; Relation: equals; Function: plus. ("One plus two" is a name for the object that is obtained by applying the function "plus" to the objects "one" and "two." "Three" is another name for this object.)
- "Squares neighboring the wumpus are smelly." Objects: wumpus, squares; Property: smelly; Relation: neighboring.
- "Evil King John ruled England in 1200." Objects: John, England, 1200; Relation: ruled during; Properties: evil, king.

The language of **first-order logic**, whose syntax and semantics we define in the next section, is built around objects and relations. It has been important to mathematics, philosophy, and artificial intelligence precisely because those fields—and indeed, much of everyday human existence—can be usefully thought of as dealing with objects and the relations among them. First-order logic can also express facts about *some* or *all* of the objects in the universe. This enables one to represent general laws or rules, such as the statement "Squares neighboring the wumpus are smelly."

The primary difference between propositional and first-order logic lies in the **ontological commitment** made by each language—that is, what it assumes about the nature of *reality*. Mathematically, this commitment is expressed through the nature of the formal models with respect to which the truth of sentences is defined. For example, propositional logic assumes that there are facts that either hold or do not hold in the world. Each fact can be in one of two states—true or false—and each model assigns *true* or *false* to each proposition symbol (see Section 7.4.2🗗). First-order logic assumes more; namely, that the world consists of objects with certain relations among them that do or do not hold. (See Figure 8.1🗗.) The formal models are correspondingly more complicated than those for propositional logic.

Figure 8.1

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Formal languages and their ontological and epistemological commitments.

Ontological commitment

This ontological commitment is a great strength of logic (both propositional and first-order), because it allows us to start with true statements and infer other true statements. It is especially powerful in domains where every proposition has clear boundaries, such as mathematics or the wumpus world, where a square either does or doesn't have a pit; there is no possibility of a square with a vaguely pit-like indentation. But in the real world, many propositions have vague boundaries: Is Vienna a large city? Does this restaurant serve delicious food? Is that person tall? It depends who you ask, and their answer might be "kind of."

One response is to refine the representation: if a crude line dividing cities into "large" and "not large" leaves out too much information for the application in question, then one can increase the number of size categories or use a *Population* function symbol. Another proposed solution comes from **Fuzzy logic**, which makes the ontological commitment that propositions have a **degree of truth** between 0 and 1. For example, the sentence "Vienna is a large city" might be true to degree 0.8 in fuzzy logic, while "Paris is a large city" might be true to degree 0.9. This corresponds better to our intuitive conception of the world, but it makes it harder to do inference: instead of one rule to determine the truth of $A \wedge B$, fuzzy logic needs different rules depending on the domain. Another possibility, covered in Section 24.1 , is to assign each concept to a point in a multidimensional space, and then measure the distance between the concept "large city" and the concept "Vienna" or "Paris."

Fuzzy logic

Degree of truth

Various special-purpose logics make still further ontological commitments; for example, **temporal logic** assumes that facts hold at particular *times* and that those times (which may be points or intervals) are ordered. Thus, special-purpose logics give certain kinds of objects (and the axioms about them) "first class" status within the logic, rather than simply defining them within the knowledge base. **Higher-order logic** views the relations and functions referred to by first-order logic as objects in themselves. This allows one to make assertions about *all* relations—for example, one could wish to define what it means for a relation to be transitive. Unlike most special-purpose logics, higher-order logic is strictly more expressive than first-order logic, in the sense that some sentences of higher-order logic cannot be expressed by any finite number of first-order logic sentences.

Temporal logic

Higher-order logic

A logic can also be characterized by its **epistemological commitments**—the possible states of knowledge that it allows with respect to each fact. In both propositional and first-order logic, a sentence represents a fact and the agent either believes the sentence to be true, believes it to be false, or has no opinion. These logics therefore have three possible states of knowledge regarding any sentence.

Systems using **probability theory**, on the other hand, can have any *degree of belief*, or *subjective likelihood*, ranging from 0 (total disbelief) to 1 (total belief). It is important not to confuse the degree of belief in probability theory with the degree of truth in fuzzy logic. Indeed, some fuzzy systems allow uncertainty (degree of belief) about degrees of truth. For example, a probabilistic wumpus-world agent might believe that the wumpus is in [1,3] with probability 0.75 and in [2, 3] with probability 0.25 (although the wumpus is definitely in one particular square).

8.2 Syntax and Semantics of First-Order Logic

We begin this section by specifying more precisely the way in which the possible worlds of first-order logic reflect the ontological commitment to objects and relations. Then we introduce the various elements of the language, explaining their semantics as we go along. The main points are how the language facilitates concise representations and how its semantics leads to sound reasoning procedures.

8.2.1 Models for first-order logic

Chapter 7 □ said that the models of a logical language are the formal structures that constitute the possible worlds under consideration. Each model links the vocabulary of the logical sentences to elements of the possible world, so that the truth of any sentence can be determined. Thus, models for propositional logic link proposition symbols to predefined truth values.

Models for first-order logic are much more interesting. First, they have objects in them! The **domain** of a model is the set of objects or **domain elements** it contains. The domain is required to be *nonempty*—every possible world must contain at least one object. (See Exercise 8.ЕМРТ for a discussion of empty worlds.) Mathematically speaking, it doesn't matter *what* these objects are—all that matters is *how many* there are in each particular model—but for pedagogical purposes we'll use a concrete example. Figure 8.2 □ shows a model with five objects: Richard the Lionheart, King of England from 1189 to 1199; his younger brother, the evil King John, who ruled from 1199 to 1215; the left legs of Richard and John; and a crown.

Figure 8.2
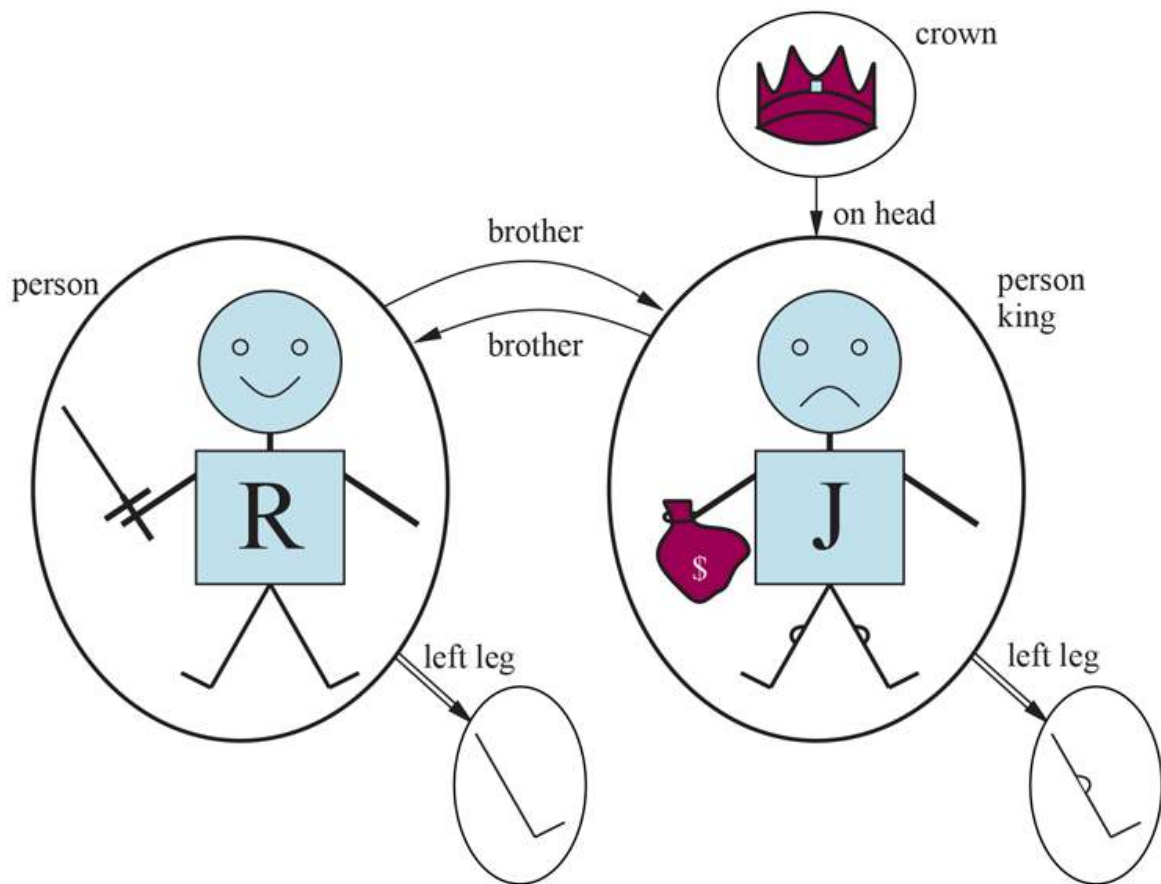
A model containing five objects, two binary relations (brother and on-head), three unary relations (person, king, and crown), and one unary function (left-leg).

Domain

Domain elements

The objects in the model may be *related* in various ways. In the figure, Richard and John are brothers. Formally speaking, a relation is just the set of **tuples** of objects that are related. (A tuple is a collection of objects arranged in a fixed order and is written with angle brackets surrounding the objects.) Thus, the brotherhood relation in this model is the set

(8.1) $\{\langle \text{Richard the Lionheart, King John}\rangle, \langle \text{King John, Richard the Lionheart}\rangle\}\,.$

Tuple

(Here we have named the objects in English, but you may, if you wish, mentally substitute the pictures for the names.) The crown is on King John's head, so the "on head" relation contains just one tuple, ⟨the crown, King John⟩. The "brother" and "on head" relations are binary relations—that is, they relate pairs of objects. The model also contains unary relations, or properties: the "person" property is true of both Richard and John; the "king" property is true only of John (presumably because Richard is dead at this point); and the "crown" property is true only of the crown.

Certain kinds of relationships are best considered as functions, in that a given object must be related to exactly one object in this way. For example, each person has one left leg, so the model has a unary "left leg" function—a mapping from a one-element tuple to an object— that includes the following mappings:

(8.2)

$$\langle \text{Richard the Lionheart}\rangle \rightarrow \text{Richard's left leg}$$
$$\langle \text{King John}\rangle \rightarrow \text{John's left leg.}$$

Strictly speaking, models in first-order logic require **total functions**, that is, there must be a value for every input tuple. Thus the crown must have a left leg and so must each of the left legs. There is a technical solution to this awkward problem involving an additional "invisible" object that is the left leg of everything that has no left leg, including itself. Fortunately, as long as one makes no assertions about the left legs of things that have no left legs, these technicalities are of no import.

Total functions

So far, we have described the elements that populate models for first-order logic. The other essential part of a model is the link between those elements and the vocabulary of the logical sentences, which we explain next.

8.2.2 Symbols and interpretations

We turn now to the syntax of first-order logic. The impatient reader can obtain a complete description from the formal grammar in Figure 8.3⬚.

Figure 8.3

$$
\begin{aligned}
Sentence \;\rightarrow\;& AtomicSentence \mid ComplexSentence \\[4pt]
AtomicSentence \;\rightarrow\;& Predicate \mid Predicate(Term, \ldots) \mid Term = Term \\[4pt]
ComplexSentence \;\rightarrow\;& (\,Sentence\,) \\
\mid\;& \neg\, Sentence \\
\mid\;& Sentence \wedge Sentence \\
\mid\;& Sentence \vee Sentence \\
\mid\;& Sentence \Rightarrow Sentence \\
\mid\;& Sentence \Leftrightarrow Sentence \\
\mid\;& Quantifier\; Variable, \ldots\; Sentence \\[8pt]
Term \;\rightarrow\;& Function(Term, \ldots) \\
\mid\;& Constant \\
\mid\;& Variable \\[8pt]
Quantifier \;\rightarrow\;& \forall \mid \exists \\
Constant \;\rightarrow\;& A \mid X_1 \mid John \mid \cdots \\
Variable \;\rightarrow\;& a \mid x \mid s \mid \cdots \\
Predicate \;\rightarrow\;& True \mid False \mid After \mid Loves \mid Raining \mid \cdots \\
Function \;\rightarrow\;& Mother \mid LeftLeg \mid \cdots
\end{aligned}
$$

OPERATOR PRECEDENCE : $\quad \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$

The syntax of first-order logic with equality, specified in Backus–Naur form (see page 1030 if you are not familiar with this notation). Operator precedences are specified, from highest to lowest. The precedence of quantifiers is such that a quantifier holds over everything to the right of it.

The basic syntactic elements of first-order logic are the symbols that stand for objects, relations, and functions. The symbols, therefore, come in three kinds: **constant symbols**, which stand for objects; **predicate symbols**, which stand for relations; and **function symbols**, which stand for functions. We adopt the convention that these symbols will begin with uppercase letters. For example, we might use the constant symbols *Richard* and *John*; the predicate symbols *Brother*, *OnHead*, *Person*, *King*, and *Crown*; and the function symbol *LeftLeg*. As with proposition symbols, the choice of names is entirely up to the user. Each predicate and function symbol comes with an **arity** that fixes the number of arguments.

Constant symbol

Predicate symbol

Function symbol

Arity

Every model must provide the information required to determine if any given sentence is true or false. Thus, in addition to its objects, relations, and functions, each model includes an **interpretation** that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols. One possible interpretation for our example—which a logician would call the **intended interpretation**—is as follows:
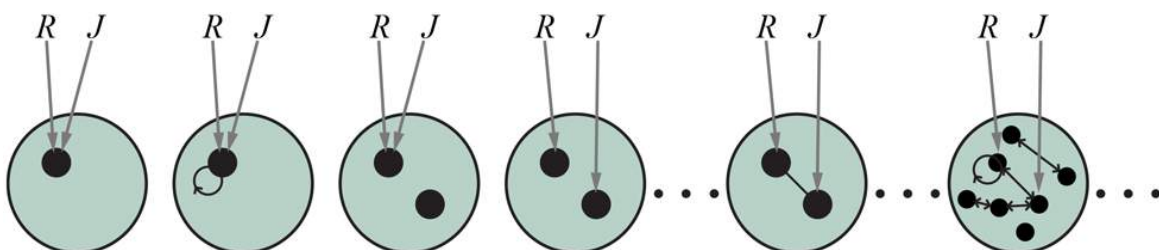

Interpretation

- *Richard* refers to Richard the Lionheart and *John* refers to the evil King John.
- *Brother* refers to the brotherhood relation—that is, the set of tuples of objects given in Equation (8.1)⬚; *OnHead* is a relation that holds between the crown and King John; *Person*, *King*, and *Crown* are unary relations that identify persons, kings, and crowns.
- *LeftLeg* refers to the "left leg" function as defined in Equation (8.2)⬚.

There are many other possible interpretations, of course. For example, one interpretation maps *Richard* to the crown and *John* to King John's left leg. There are five objects in the model, so there are 25 possible interpretations just for the constant symbols *Richard* and *John*. Notice that not all the objects need have a name—for example, the intended interpretation does not name the crown or the legs. It is also possible for an object to have several names; there is an interpretation under which both *Richard* and *John* refer to the crown.[2] If you find this possibility confusing, remember that, in propositional logic, it is perfectly possible to have a model in which *Cloudy* and *Sunny* are both true; it is the job of the knowledge base to rule out models that are inconsistent with our knowledge.

---

2 Later, in Section **8.2.8**⬚, we examine a semantics in which every object must have exactly one name.

In summary, a model in first-order logic consists of a set of objects and an interpretation that maps constant symbols to objects, function symbols to functions on those objects, and predicate symbols to relations. Just as with propositional logic, entailment, validity, and so on are defined in terms of *all possible models*. To get an idea of what the set of all possible models looks like, see Figure 8.4⬚. It shows that models vary in how many objects they contain—from one to infinity—and in the way the constant symbols map to objects.

Figure 8.4

Some members of the set of all models for a language with two constant symbols, R and J, and one binary relation symbol. The interpretation of each constant symbol is shown by a gray arrow. Within each model, the related objects are connected by arrows.

Because the number of first-order models is unbounded, we cannot check entailment by enumerating them all (as we did for propositional logic). Even if the number of objects is restricted, the number of combinations can be very large. (See Exercise 8.MCNT.) For the example in Figure 8.4⬚, there are 137,506,194,466 models with six or fewer objects.

8.2.3 Terms

A **term** is a logical expression that refers to an object. Constant symbols are terms, but it is not always convenient to have a distinct symbol to name every object. In English we might use the expression "King John's left leg" rather than giving a name to his leg. This is what function symbols are for: instead of using a constant symbol, we use *LeftLeg*(*John*).[3]⬚

3 λ-**expressions** (lambda expressions) provide a useful notation in which new function symbols are constructed "on the fly." For example, the function that squares its argument can be written as $(\lambda x : x \times x)$ and can be applied to arguments just like any other function symbol. A λ-expression can also be defined and used as a predicate symbol. The lambda operator in Lisp and Python plays exactly the same role. Notice that the use of λ in this way does *not* increase the formal expressive power of first-order logic, because any sentence that includes a λ-expression can be rewritten by "plugging in" its arguments to yield an equivalent sentence.

Term

In the general case, a complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol. It is important to remember

that a complex term is just a complicated kind of name. It is not a "subroutine call" that "returns a value." There is no *LeftLeg* subroutine that takes a person as input and returns a leg. We can reason about left legs (e.g., stating the general rule that everyone has one and then deducing that John must have one) without ever providing a definition of *LeftLeg*. This is something that cannot be done with subroutines in programming languages.

The formal semantics of terms is straightforward. Consider a term $f(t_1, \ldots, t_n)$. The function symbol f refers to some function in the model (call it F); the argument terms refer to objects in the domain (call them $d_1, \ldots, d_n$); and the term as a whole refers to the object that is the value of the function F applied to $d_1, \ldots, d_n$. For example, suppose the *LeftLeg* function symbol refers to the function shown in Equation (8.2)⬚ and *John* refers to King John, then *LeftLeg(John)* refers to King John's left leg. In this way, the interpretation fixes the referent of every term.

8.2.4 Atomic sentences

Now that we have terms for referring to objects and predicate symbols for referring to relations, we can combine them to make **atomic sentences** that state facts. An **atomic sentence** (or **atom** for short) is formed from a predicate symbol optionally followed by a parenthesized list of terms, such as

$$Brother(Richard, John).$$

Atomic sentence

Atom

This states, under the intended interpretation given earlier, that Richard the Lionheart is the brother of King John.[4]⬚ Atomic sentences can have complex terms as arguments. Thus,

4 We usually follow the argument-ordering convention that $P(x, y)$ is read as "x is a P of y."

$$Married(Father(Richard),Mother(John))$$

states that Richard the Lionheart's father is married to King John's mother (again, under a suitable interpretation).[5]

5 This ontology only recognizes one father and one mother for each person. A more complex ontology could recognize biological mother, birth mother, adoptive mother, etc.

*An atomic sentence is **true** in a given model if the relation referred to by the predicate symbol holds among the objects referred to by the arguments.*

8.2.5 Complex sentences

We can use **logical connectives** to construct more complex sentences, with the same syntax and semantics as in propositional calculus. Here are four sentences that are true in the model of Figure 8.2 under our intended interpretation:

$$\neg Brother(LeftLeg(Richard),John)$$
$$Brother(Richard,John) \land Brother(John,Richard)$$
$$King(Richard) \lor King(Richard)$$
$$\neg King(Richard) \Rightarrow King(John).$$

8.2.6 Quantifiers

Once we have a logic that allows objects, it is only natural to want to express properties of entire collections of objects, instead of enumerating the objects by name. **Quantifiers** let us do this. First-order logic contains two standard quantifiers, called *universal* and *existential*.

Quantifier

Universal quantification (∀)

Recall the difficulty we had in with the expression of general rules in propositional logic. Rules such as "Squares neighboring the wumpus are smelly" and "All kings are persons" are the bread and butter of first-order logic. We deal with the first of these in . The second rule, "All kings are persons," is written in first-order logic as

$$\forall x \ King(x) \Rightarrow Person(x).$$

The **universal quantifier** $\forall$ is usually pronounced "For all ...". (Remember that the upside-down A stands for "all.") Thus, the sentence says, "For all x, if x is a king, then x is a person." The symbol x is called a **variable**. By convention, variables are lowercase letters. A variable is a term all by itself, and as such can also serve as the argument of a function—for example, $LeftLeg(x)$. A term with no variables is called a **ground term**.

Universal quantifier

Variable

Ground term

Intuitively, the sentence $\forall x\, P$, where P is any logical sentence, says that P is true for every object x. More precisely, $\forall x\, P$ is true in a given model if P is true in all possible **extended interpretations** constructed from the interpretation given in the model, where each extended interpretation specifies a domain element to which x refers.

Extended interpretation

This sounds complicated, but it is really just a careful way of stating the intuitive meaning of universal quantification. Consider the model shown in Figure 8.2⬚ and the intended interpretation that goes with it. We can extend the interpretation in five ways:

$$x \rightarrow \text{Richard the Lionheart,}$$
$$x \rightarrow \text{King John,}$$
$$x \rightarrow \text{Richard's left leg,}$$
$$x \rightarrow \text{John's left leg,}$$
$$x \rightarrow \text{the crown.}$$

The universally quantified sentence $\forall x \; King(x) \Rightarrow Person(x)$ is true in the original model if the sentence $King(x) \Rightarrow Person(x)$ is true under each of the five extended interpretations. That is, the universally quantified sentence is equivalent to asserting the following five sentences:

Richard the Lionheart is a king $\Rightarrow$ Richard the Lionheart is a person.
King John is a king $\Rightarrow$ King John is a person.
Richard's left leg is a king $\Rightarrow$ Richard's left leg is a person.
John's left leg is a king $\Rightarrow$ John's left leg is a person.
The crown is a king $\Rightarrow$ the crown is a person.

Let us look carefully at this set of assertions. Since, in our model, King John is the only king, the second sentence asserts that he is a person, as we would hope. But what about the other four sentences, which appear to make claims about legs and crowns? Is that part of the meaning of "All kings are persons"? In fact, the other four assertions are true in the model, but make no claim whatsoever about the personhood qualifications of legs, crowns, or indeed Richard. This is because none of these objects is a king. Looking at the truth table for $\Rightarrow$ (Figure 7.8⬚ on page 219), we see that the implication is true whenever its premise is false— *regardless* of the truth of the conclusion. Thus, by asserting the universally quantified sentence, which is equivalent to asserting a whole list of individual implications, we end up asserting the conclusion of the rule just for those objects for which the premise is true and saying nothing at all about those objects for which the premise is false. Thus, the truth-table definition of $\Rightarrow$ turns out to be perfect for writing general rules with universal quantifiers.

A common mistake, made frequently even by diligent readers who have read this paragraph several times, is to use conjunction instead of implication. The sentence

$$\forall x \; King(x) \land Person(x)$$

would be equivalent to asserting

> Richard the Lionheart is a king∧ Richard the Lionheart is a person,
> King John is a king∧ King John is a person,
> Richard's left leg is a king∧ Richard's left leg is a person,

and so on. Obviously, this does not capture what we want.

Existential quantification (∃)

Universal quantification makes statements about every object. Similarly, we can make a statement about *some* object without naming it, by using an **existential quantifier**. To say, for example, that King John has a crown on his head, we write

$$\exists x \ \ Crown(x) \wedge OnHead(x, John) \,.$$

Existential quantifier

$\exists x$ is pronounced "There exists an x such that ..." or "For some $x \ldots$".

Intuitively, the sentence $\exists x \ P$ says that P is true for at least one object x. More precisely, $\exists x \ P$ is true in a given model if P is true in *at least one* extended interpretation that assigns x to a domain element. That is, at least one of the following is true:

> Richard the Lionheart is a crown∧ Richard the Lionheart is on John's head;
> King John is a crown∧ King John is on John's head;
> Richard's left leg is a crown ∧ Richard's left leg is on John's head;
> John's left leg is a crown∧ John's left leg is on John's head;
> The crown is a crown∧ the crown is on John's head.

The fifth assertion is true in the model, so the original existentially quantified sentence is true in the model. Notice that, by our definition, the sentence would also be true in a model in which King John was wearing two crowns. This is entirely consistent with the original sentence "King John has a crown on his head." [6]

6 There is a variant of the existential quantifier, usually written $\exists^1$ or $\exists!$, that means "There exists exactly one." The same meaning can be expressed using equality statements.

Just as $\Rightarrow$ appears to be the natural connective to use with $\forall, \wedge$ is the natural connective to use with $\exists$. Using $\wedge$ as the main connective with $\forall$ led to an overly strong statement in the example in the previous section; using $\Rightarrow$ with $\exists$ usually leads to a very weak statement, indeed. Consider the following sentence:

$$\exists x \ \ Crown(x) \Rightarrow OnHead(x, John).$$

On the surface, this might look like a reasonable rendition of our sentence. Applying the semantics, we see that the sentence says that at least one of the following assertions is true:

Richard the Lionheart is a crown $\Rightarrow$ Richard the Lionheart is on John's head;
King John is a crown $\Rightarrow$ King John is on John's head;
Richard's left leg is a crown $\Rightarrow$ Richard's left leg is on John's head;

and so on. An implication is true if both premise and conclusion are true, *or if its premise is false*; so if Richard the Lionheart is not a crown, then the first assertion is true and the existential is satisfied. So, an existentially quantified implication sentence is true whenever *any* object fails to satisfy the premise; hence such sentences really do not say much at all.

Nested quantifiers

We will often want to express more complex sentences using multiple quantifiers. The simplest case is where the quantifiers are of the same type. For example, "Brothers are siblings" can be written as

$$\forall x \ \ \forall y \ \ Brother(x,y) \Rightarrow Sibling(x,y) \,.$$

Consecutive quantifiers of the same type can be written as one quantifier with several variables. For example, to say that siblinghood is a symmetric relationship, we can write

$$\forall x,y \ \ Sibling(x,y) \Leftrightarrow Sibling(y,x) \,.$$

In other cases we will have mixtures. "Everybody loves somebody" means that for every person, there is someone that person loves:

$$\forall x \ \exists y \ Loves(x,y)\,.$$

On the other hand, to say "There is someone who is loved by everyone," we write

$$\exists y \ \forall x \ Loves(x,y)\,.$$

The order of quantification is therefore very important. It becomes clearer if we insert parentheses. $\forall x \ (\exists y \ Loves(x,y))$ says that *everyone* has a particular property, namely, the property that they love someone. On the other hand, $\exists y \ (\forall x \ Loves(x,y))$ says that *someone* in the world has a particular property, namely the property of being loved by everybody.

Some confusion can arise when two quantifiers are used with the same variable name. Consider the sentence

$$\forall x \ (Crown(x) \lor (\exists x \ Brother(Richard,x)))\,.$$

Here the x in $Brother(Richard,x)$ is *existentially* quantified. The rule is that the variable belongs to the innermost quantifier that mentions it; then it will not be subject to any other quantification. Another way to think of it is this: $\exists x \ Brother(Richard,x)$ is a sentence about Richard (that he has a brother), not about x; so putting a $\forall x$ outside it has no effect. It could equally well have been written $\exists z \ Brother(Richard,z)$. Because this can be a source of confusion, we will always use different variable names with nested quantifiers.

Connections between ∀ and ∃

The two quantifiers are actually intimately connected with each other, through negation. Asserting that everyone dislikes parsnips is the same as asserting there does not exist someone who likes them, and vice versa:

$$\forall x \ \neg Likes(x,Parsnips) \quad \text{is equivalent to} \quad \neg \exists x \ Likes(x,Parsnips)\,.$$

We can go one step further: "Everyone likes ice cream" means that there is no one who does not like ice cream:

$$\forall x \ Likes(x,IceCream) \quad \text{is equivalent to} \quad \neg \exists x \ \neg Likes(x,IceCream)\,.$$

Because $\forall$ is really a conjunction over the universe of objects and $\exists$ is a disjunction, it should not be surprising that they obey De Morgan's rules. The De Morgan rules for quantified and unquantified sentences are as follows:

$$
\begin{array}{llllllll}
\neg\exists x & P & \equiv & \forall x & \neg P & \neg(P \vee Q) & \equiv & \neg P \wedge \neg Q \\
\neg\forall x & P & \equiv & \exists x & \neg P & \neg(P \wedge Q) & \equiv & \neg P \vee \neg Q \\
\forall x & P & \equiv & \neg\exists x & \neg P & P \wedge Q & \equiv & \neg(\neg P \wedge \neg Q) \\
\exists x & P & \equiv & \neg\forall x & \neg P & P \vee Q & \equiv & \neg(\neg P \wedge \neg Q) \, .
\end{array}
$$

Thus, we do not really need both $\forall$ and $\exists$, just as we do not really need both $\wedge$ and $\vee$. Still, readability is more important than parsimony, so we will keep both of the quantifiers.

8.2.7 Equality

First-order logic includes one more way to make atomic sentences, other than using a predicate and terms as described earlier. We can use the **equality symbol** to signify that two terms refer to the same object. For example,

$$
Father(John) = Henry
$$

Equality symbol

says that the object referred to by *Father(John)* and the object referred to by *Henry* are the same. Because an interpretation fixes the referent of any term, determining the truth of an equality sentence is simply a matter of seeing that the referents of the two terms are the same object.

The equality symbol can be used to state facts about a given function, as we just did for the *Father* symbol. It can also be used with negation to insist that two terms are not the same object. To say that Richard has at least two brothers, we would write

$$
\exists x,y \;\; Brother(x,Richard) \wedge Brother(y,Richard) \wedge \neg(x = y) \, .
$$

The sentence

$$\exists\, x,y \;\; Brother(x,Richard) \land Brother(y,Richard)$$

does not have the intended meaning. In particular, it is true in the model of Figure 8.2 , where Richard has only one brother. To see this, consider the extended interpretation in which both x and y are assigned to King John. The addition of $\neg(x = y)$ rules out such models. The notation $x \neq y$ is sometimes used as an abbreviation for $\neg(x = y)$.

8.2.8 Database semantics

Continuing the example from the previous section, suppose that we believe that Richard has two brothers, John and Geoffrey.[7] We could write

7 Actually he had four, the others being William and Henry.

(8.3)

$$Brother(John,Richard) \land Brother(Geoffrey,Richard) \,,$$

but that wouldn't completely capture the state of affairs. First, this assertion is true in a model where Richard has only one brother—we need to add $John \neq Geoffrey$. Second, the sentence doesn't rule out models in which Richard has many more brothers besides John and Geoffrey. Thus, the correct translation of "Richard's brothers are John and Geoffrey" is as follows:

$$Brother(John,Richard) \land Brother(Geoffrey,Richard) \land John \neq Geoffrey$$
$$\land\, \forall x\; Brother(x,Richard) \Rightarrow (x = John \lor x = Geoffrey)$$

This logical sentence seems much more cumbersome than the corresponding English sentence. But if we fail to translate the English properly, our logical reasoning system will make mistakes. Can we devise a semantics that allows a more straightforward logical sentence?

One proposal that is very popular in database systems works as follows. First, we insist that every constant symbol refer to a distinct object—the **unique-names assumption**. Second, we assume that atomic sentences not known to be true are in fact false—the **closed-world assumption**. Finally, we invoke **domain closure**, meaning that each model contains no more domain elements than those named by the constant symbols.

*Unique-names assumption*

Closed-world assumption

Domain closure

Under the resulting semantics, Equation (8.3)⬚ does indeed state that Richard has exactly two brothers, John and Geoffrey. We call this **database semantics** to distinguish it from the standard semantics of first-order logic. Database semantics is also used in logic programming systems, as explained in Section 9.4.4⬚.

Database semantics

It is instructive to consider the set of all possible models under database semantics for the same case as shown in Figure 8.4⬚ (page 259). Figure 8.5⬚ shows some of the models, ranging from the model with no tuples satisfying the relation to the model with all tuples satisfying the relation. With two objects, there are four possible two-element tuples, so there are $2^4 = 6$ different subsets of tuples that can satisfy the relation. Thus, there are 16 possible models in all—a lot fewer than the infinitely many models for the standard first-order semantics. On the other hand, the database semantics requires definite knowledge of what the world contains.

Figure 8.5
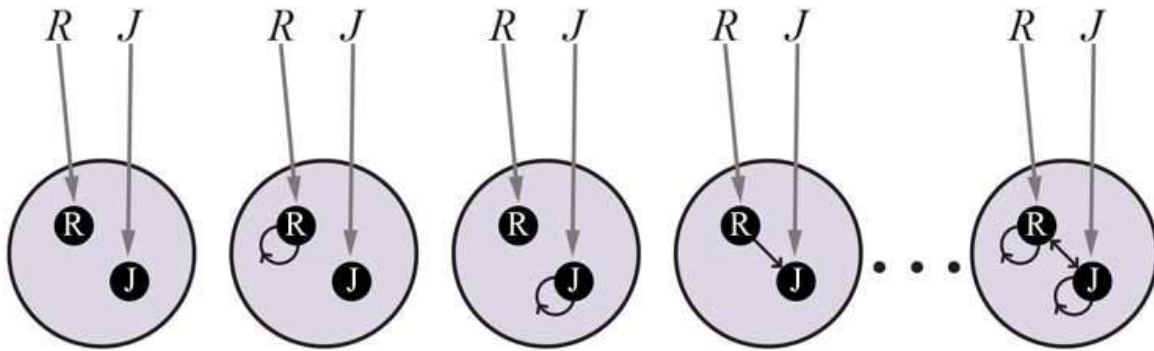
Some members of the set of all models for a language with two constant symbols, R and J, and one binary relation symbol, under database semantics. The interpretation of the constant symbols is fixed, and there is a distinct object for each constant symbol.

This example brings up an important point: there is no one "correct" semantics for logic. The usefulness of any proposed semantics depends on how concise and intuitive it makes the expression of the kinds of knowledge we want to write down, and on how easy and natural it is to develop the corresponding rules of inference. Database semantics is most useful when we are certain about the identity of all the objects described in the knowledge base and when we have all the facts at hand; in other cases, it is quite awkward. For the rest of this chapter, we assume the standard semantics while noting instances in which this choice leads to cumbersome expressions.