

CHAPTER 10

KNOWLEDGE REPRESENTATION

In which we show how to represent diverse facts about the real world in a form that can be used to reason and solve problems.

The previous chapters showed how an agent with a knowledge base can make inferences that enable it to act appropriately. In this chapter we address the question of what *content* to put into such an agent's knowledge base—how to represent facts about the world. We will use first-order logic as the representation language, but later chapters will introduce different representation formalisms such as hierarchical task networks for reasoning about plans (Chapter 11), Bayesian networks for reasoning with uncertainty (Chapter 13), Markov models for reasoning over time (Chapter 16), and deep neural networks for reasoning about images, sounds, and other data (Chapter 22). But no matter what representation you use, the facts about the world still need to be handled, and this chapter gives you a feeling for the issues.

Section 10.1 introduces the idea of a general ontology, which organizes everything in the world into a hierarchy of categories. Section 10.2 covers the basic categories of objects, substances, and measures; Section 10.3 covers events; and Section 10.4 discusses knowledge about beliefs. We then return to consider the technology for reasoning with this content: Section 10.5 discusses reasoning systems designed for efficient inference with categories, and Section 10.6 discusses reasoning with default information.

10.1 Ontological Engineering

In “toy” domains, the choice of representation is not that important; many choices will work. Complex domains such as shopping on the Internet or driving a car in traffic require more general and flexible representations. This chapter shows how to create these representations, concentrating on general concepts—such as *Events*, *Time*, *Physical Objects*, and *Beliefs*—that occur in many different domains. Representing these abstract concepts is sometimes called **ontological engineering**.

We cannot hope to represent *everything* in the world, even a 1000-page textbook, but we will leave placeholders where new knowledge for any domain can fit in. For example, we will define what it means to be a physical object, and the details of different types of objects—robots, televisions, books, or whatever—can be filled in later. This is analogous to the way that designers of an object-oriented programming framework (such as the Java Swing graphical framework) define general concepts like *Window*, expecting users to use these to define more specific concepts like *SpreadsheetWindow*. The general framework of concepts is called an **upper ontology** because of the convention of drawing graphs with the general concepts at the top and the more specific concepts below them, as in Figure 10.1.

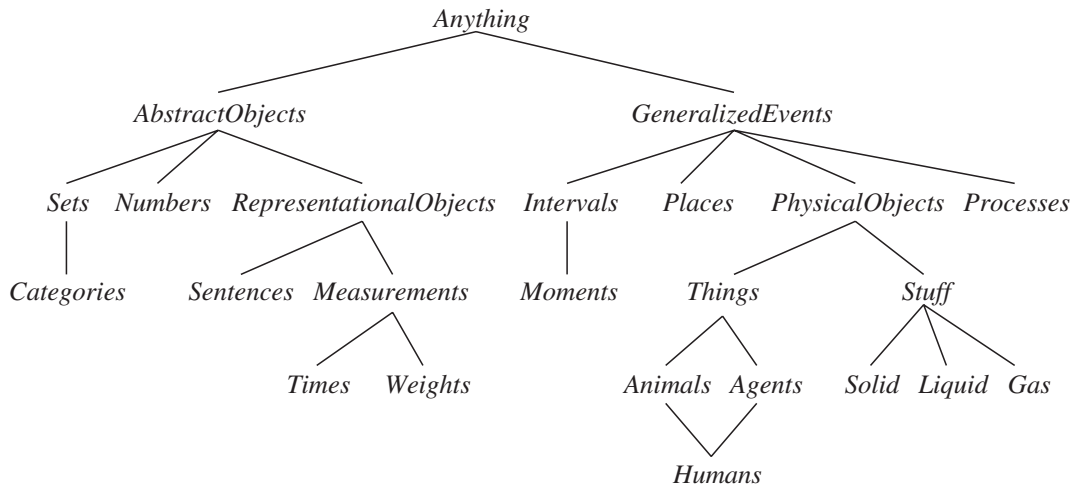


Figure 10.1 The upper ontology of the world, showing the topics to be covered later in the chapter. Each link indicates that the lower concept is a specialization of the upper one. Specializations are not necessarily disjoint—a human is both an animal and an agent. We will see in Section 10.3.2 why physical objects come under generalized events.

Before considering the ontology further, we should state one important caveat. We have elected to use first-order logic to discuss the content and organization of knowledge, although certain aspects of the real world are hard to capture in FOL. The principal difficulty is that most generalizations have exceptions or hold only to a degree. For example, although “tomatoes are red” is a useful rule, some tomatoes are green, yellow, or orange. Similar exceptions can be found to almost all the rules in this chapter. The ability to handle exceptions and uncertainty is extremely important, but is orthogonal to the task of understanding the general ontology. For this reason, we delay the discussion of exceptions until Section 10.5 of this chapter, and the more general topic of reasoning with uncertainty until Chapter 12.

Of what use is an upper ontology? Consider the ontology for circuits in Section 8.4.2. It makes many simplifying assumptions: time is omitted completely; signals are fixed and do not propagate; the structure of the circuit remains constant. A more general ontology would consider signals at particular times, and would include the wire lengths and propagation delays. This would allow us to simulate the timing properties of the circuit, and indeed such simulations are often carried out by circuit designers.

We could also introduce more interesting classes of gates, for example, by describing the technology (TTL, CMOS, and so on) as well as the input–output specification. If we wanted to discuss reliability or diagnosis, we would include the possibility that the structure of the circuit or the properties of the gates might change spontaneously. To account for stray capacitances, we would need to represent where the wires are on the board.

If we look at the wumpus world, similar considerations apply. Although we do represent time, it has a simple structure: Nothing happens except when the agent acts, and all changes are instantaneous. A more general ontology, better suited for the real world, would allow for simultaneous changes extended over time. We also used a *Pit* predicate to say which squares have pits. We could have allowed for different kinds of pits by having several individuals

belonging to the class of pits, each having different properties. Similarly, we might want to allow for other animals besides wumpuses. It might not be possible to pin down the exact species from the available percepts, so we would need to build up a biological taxonomy to help the agent predict the behavior of cave dwellers from scanty clues.

For any special-purpose ontology, it is possible to make changes like these to move toward greater generality. An obvious question then arises: do all these ontologies converge on a general-purpose ontology? After centuries of philosophical and computational investigation, the answer is “Maybe.” In this section, we present one general-purpose ontology that synthesizes ideas from those centuries. Two major characteristics of general-purpose ontologies distinguish them from collections of special-purpose ontologies:

- A general-purpose ontology should be applicable in more or less any special-purpose domain (with the addition of domain-specific axioms). This means that no representational issue can be finessed or swept under the carpet.
- In any sufficiently demanding domain, different areas of knowledge must be *unified*, because reasoning and problem solving could involve several areas simultaneously. A robot circuit-repair system, for instance, needs to reason about circuits in terms of electrical connectivity and physical layout, and about time, both for circuit timing analysis and estimating labor costs. The sentences describing time therefore must be capable of being combined with those describing spatial layout and must work equally well for nanoseconds and minutes and for angstroms and meters.

We should say up front that the enterprise of general ontological engineering has so far had only limited success. None of the top AI applications (as listed in Chapter 1) make use of a general ontology—they all use special-purpose knowledge engineering and machine learning. Social/political considerations can make it difficult for competing parties to agree on an ontology. As Tom Gruber (2004) says, “Every ontology is a treaty—a social agreement—among people with some common motive in sharing.” When competing concerns outweigh the motivation for sharing, there can be no common ontology. The smaller the number of stakeholders, the easier it is to create an ontology, and thus it is harder to create a general-purpose ontology than a limited-purpose one, such as the Open Biomedical Ontology (Smith *et al.*, 2007). Those ontologies that do exist have been created along four routes:

1. By a team of trained ontologists or logicians, who architect the ontology and write axioms. The CYC system was mostly built this way (Lenat and Guha, 1990).
2. By importing categories, attributes, and values from an existing database or databases. DBPEDIA was built by importing structured facts from Wikipedia (Bizer *et al.*, 2007).
3. By parsing text documents and extracting information from them. TEXTRUNNER was built by reading a large corpus of Web pages (Banko and Etzioni, 2008).
4. By enticing unskilled amateurs to enter commonsense knowledge. The OPENMIND system was built by volunteers who proposed facts in English (Singh *et al.*, 2002; Chklovski and Gil, 2005).

As an example, the Google Knowledge Graph uses semistructured content from Wikipedia, combining it with other content gathered from across the web under human curation. It contains over 70 billion facts and provides answers for about a third of Google searches (Dong *et al.*, 2014).

10.2 Categories and Objects

Category



The organization of objects into **categories** is a vital part of knowledge representation. Although interaction with the world takes place at the level of individual objects, *much reasoning takes place at the level of categories*. For example, a shopper would normally have the goal of buying a basketball, rather than a *particular* basketball such as BB_9 . Categories also serve to make predictions about objects once they are classified. One infers the presence of certain objects from perceptual input, infers category membership from the perceived properties of the objects, and then uses category information to make predictions about the objects. For example, from its green and yellow mottled skin, one-foot diameter, ovoid shape, red flesh, black seeds, and presence in the fruit aisle, one can infer that an object is a watermelon; from this, one infers that it would be useful for fruit salad.

Reification

There are two choices for representing categories in first-order logic: predicates and objects. That is, we can use the predicate $Basketball(b)$, or we can **reify**¹ the category as an object, $Basketballs$. We could then say $Member(b, Basketballs)$, which we will abbreviate as $b \in Basketballs$, to say that b is a member of the category of basketballs. We say $Subset(Basketballs, Balls)$, abbreviated as $Basketballs \subset Balls$, to say that $Basketballs$ is a **subcategory** of $Balls$. We will use subcategory, subclass, and subset interchangeably.

Subcategory

Inheritance

Categories organize knowledge through **inheritance**. If we say that all instances of the category $Food$ are edible, and if we assert that $Fruit$ is a subclass of $Food$ and $Apples$ is a subclass of $Fruit$, then we can infer that every apple is edible. We say that the individual apples **inherit** the property of edibility, in this case from their membership in the $Food$ category.

Taxonomic hierarchy

Subclass relations organize categories into a **taxonomic hierarchy** or **taxonomy**. Taxonomies have been used explicitly for centuries in technical fields. The largest such taxonomy organizes about 10 million living and extinct species, many of them beetles,² into a single hierarchy; library science has developed a taxonomy of all fields of knowledge, encoded as the Dewey Decimal system; and tax authorities and other government departments have developed extensive taxonomies of occupations and commercial products.

First-order logic makes it easy to state facts about categories, either by relating objects to categories or by quantifying over their members. Here are some example facts:

- An object is a member of a category.
 $BB_9 \in Basketballs$
- A category is a subclass of another category.
 $Basketballs \subset Balls$
- All members of a category have some properties.
 $(x \in Basketballs) \Rightarrow Spherical(x)$
- Members of a category can be recognized by some properties.
 $Orange(x) \wedge Round(x) \wedge Diameter(x) = 9.5'' \wedge x \in Balls \Rightarrow x \in Basketballs$
- A category as a whole has some properties.
 $Dogs \in DomesticatedSpecies$

¹ Turning a proposition into an object is called **reification**, from the Latin word *res*, or thing. John McCarthy proposed the term “thingification,” but it never caught on.

² When asked what one could deduce about the Creator from the study of nature, biologist J. B. S. Haldane said “An inordinate fondness for beetles.”

Notice that because *Dogs* is a category and is a member of *DomesticatedSpecies*, the latter must be a category of categories. Of course there are exceptions to many of the above rules (punctured basketballs are not spherical); we deal with these exceptions later.

Although subclass and member relations are the most important ones for categories, we also want to be able to state relations between categories that are not subclasses of each other. For example, if we just say that *Undergraduates* and *GraduateStudents* are subclasses of *Students*, then we have not said that an undergraduate cannot also be a graduate student. We say that two or more categories are **disjoint** if they have no members in common. We may also want to say that the classes undergrad and graduate student form an **exhaustive decomposition** of university students. A exhaustive decomposition of disjoint sets is known as a **partition**. Here are some more examples of these three concepts:

Disjoint

Exhaustive
decomposition
Partition

Disjoint({*Animals*, *Vegetables*})
ExhaustiveDecomposition({*Americans*, *Canadians*, *Mexicans*},
NorthAmericans)
Partition({*Animals*, *Plants*, *Fungi*, *Protista*, *Monera*},
LivingThings).

(Note that the *ExhaustiveDecomposition* of *NorthAmericans* is not a *Partition*, because some people have dual citizenship.) The three predicates are defined as follows:

$Disjoint(s) \Leftrightarrow (\forall c_1, c_2 \ c_1 \in s \wedge c_2 \in s \wedge c_1 \neq c_2 \Rightarrow Intersection(c_1, c_2) = \{ \})$
 $ExhaustiveDecomposition(s, c) \Leftrightarrow (\forall i \ i \in c \Leftrightarrow \exists c_2 \ c_2 \in s \wedge i \in c_2)$
 $Partition(s, c) \Leftrightarrow Disjoint(s) \wedge ExhaustiveDecomposition(s, c).$

Categories can also be *defined* by providing necessary and sufficient conditions for membership. For example, a bachelor is an unmarried adult male:

$x \in Bachelors \Leftrightarrow Unmarried(x) \wedge x \in Adults \wedge x \in Males.$

As we discuss in the sidebar on natural kinds on page 338, strict logical definitions for categories are usually possible only for artificial formal terms, not for ordinary objects. But definitions are not always necessary.

10.2.1 Physical composition

The idea that one object can be part of another is a familiar one. One's nose is part of one's head, Romania is part of Europe, and this chapter is part of this book. We use the general *PartOf* relation to say that one thing is part of another. Objects can be grouped into *PartOf* hierarchies, reminiscent of the *Subset* hierarchy:

PartOf(*Bucharest*, *Romania*)
PartOf(*Romania*, *EasternEurope*)
PartOf(*EasternEurope*, *Europe*)
PartOf(*Europe*, *Earth*).

The *PartOf* relation is transitive and reflexive; that is,

$PartOf(x, y) \wedge PartOf(y, z) \Rightarrow PartOf(x, z)$
 $PartOf(x, x).$

Composite object

Therefore, we can conclude *PartOf*(*Bucharest*, *Earth*). Categories of **composite objects** are often characterized by structural relations among parts. For example, a biped is an object

with exactly two legs attached to a body:

$$\begin{aligned} Biped(a) \Rightarrow & \exists l_1, l_2, b \text{ Leg}(l_1) \wedge \text{Leg}(l_2) \wedge \text{Body}(b) \wedge \\ & \text{PartOf}(l_1, a) \wedge \text{PartOf}(l_2, a) \wedge \text{PartOf}(b, a) \wedge \\ & \text{Attached}(l_1, b) \wedge \text{Attached}(l_2, b) \wedge \\ & l_1 \neq l_2 \wedge [\forall l_3 \text{ Leg}(l_3) \wedge \text{PartOf}(l_3, a) \Rightarrow (l_3 = l_1 \vee l_3 = l_2)]. \end{aligned}$$

The notation for “exactly two” is a little awkward; we are forced to say that there are two legs, that they are not the same, and that if anyone proposes a third leg, it must be the same as one of the other two. In Section 10.5.2, we describe a formalism called description logic that makes it easier to represent constraints like “exactly two.”

We can define a *PartPartition* relation analogous to the *Partition* relation for categories. (See Exercise 10.DECM.) An object is composed of the parts in its *PartPartition* and can be viewed as deriving some properties from those parts. For example, the mass of a composite object is the sum of the masses of the parts. Notice that this is not the case with categories, which have no mass, even though their elements might.

It is also useful to define composite objects with definite parts but no particular structure. For example, we might want to say “The apples in this bag weigh two pounds.” The temptation would be to ascribe this weight to the *set* of apples in the bag, but this would be a mistake because the set is an abstract mathematical concept that has elements but does not have weight. Instead, we need a new concept, which we will call a **bunch**. For example, if the apples are *Apple*₁, *Apple*₂, and *Apple*₃, then Bunch

$$BunchOf(\{Apple_1, Apple_2, Apple_3\})$$

denotes the composite object with the three apples as parts (not elements). We can then use the bunch as a normal, albeit unstructured, object. Notice that $BunchOf(\{x\}) = x$. Furthermore, $BunchOf(Apples)$ is the composite object consisting of all apples—not to be confused with *Apples*, the category or set of all apples.

We can define *BunchOf* in terms of the *PartOf* relation. Obviously, each element of *s* is part of *BunchOf(s)*:

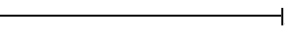
$$\forall x \ x \in s \Rightarrow \text{PartOf}(x, BunchOf(s)).$$

Furthermore, *BunchOf(s)* is the smallest object satisfying this condition. In other words, *BunchOf(s)* must be part of any object that has all the elements of *s* as parts:

$$\forall y \ [\forall x \ x \in s \Rightarrow \text{PartOf}(x, y)] \Rightarrow \text{PartOf}(BunchOf(s), y).$$

These axioms are an example of a general technique called **logical minimization**, which Logical minimization means defining an object as the smallest one satisfying certain conditions.

10.2.2 Measurements

In both scientific and commonsense theories of the world, objects have height, mass, cost, and so on. The values that we assign for these properties are called **measures**. Measure Ordinary quantitative measures are quite easy to represent. We imagine that the universe includes abstract “measure objects,” such as the *length* that is the length of this line segment: . We can call this length 1.5 inches or 3.81 centimeters. Thus, the same length has different names in our language. We represent the length with a **units function** that takes a number as argument. (An alternative is explored in Exercise 10.ALT.M.) Units function

Natural Kinds

Some categories have strict definitions: an object is a triangle if and only if it is a polygon with three sides. On the other hand, most categories in the real world have no clear-cut definition; these are called **natural kind** categories. For example, tomatoes tend to be a dull scarlet; roughly spherical; with an indentation at the top where the stem was; about two to four inches in diameter; with a thin but tough skin; and with flesh, seeds, and juice inside. However, there is variation: some tomatoes are yellow or orange, unripe tomatoes are green, some are smaller or larger than average, and cherry tomatoes are uniformly small. Rather than having a complete definition of tomatoes, we have a set of features that serves to identify objects that are clearly typical tomatoes, but might not definitively identify other objects. (Could there be a tomato that is fuzzy like a peach?)

This poses a problem for a logical agent. The agent cannot be sure that an object it has perceived is a tomato, and even if it were sure, it could not be certain which of the properties of typical tomatoes this one has. This problem is an inevitable consequence of operating in partially observable environments.

One useful approach is to separate what is true of all instances of a category from what is true only of typical instances. So in addition to the category *Tomatoes*, we will also have the category *Typical(Tomatoes)*. Here, the *Typical* function maps a category to the subclass that contains only typical instances:

$$\text{Typical}(c) \subseteq c.$$

Most knowledge about natural kinds will actually be about their typical instances:

$$x \in \text{Typical}(\text{Tomatoes}) \Rightarrow \text{Red}(x) \wedge \text{Round}(x).$$

Thus, we can write down useful facts about categories without exact definitions. The difficulty of providing exact definitions for most natural categories was explained in depth by Wittgenstein (1953). He used the example of *games* to show that members of a category shared “family resemblances” rather than necessary and sufficient characteristics: what strict definition encompasses chess, tag, solitaire, and dodgeball?

The utility of the notion of strict definition was also challenged by Quine (1953). He pointed out that even the definition of “bachelor” as an unmarried adult male is suspect; one might, for example, question a statement such as “the Pope is a bachelor.” While not strictly *false*, this usage is certainly *infelicitous* because it induces unintended inferences on the part of the listener. The tension could perhaps be resolved by distinguishing between logical definitions suitable for internal knowledge representation and the more nuanced criteria for felicitous linguistic usage. The latter may be achieved by “filtering” the assertions derived from the former. It is also possible that failures of linguistic usage serve as feedback for modifying internal definitions, so that filtering becomes unnecessary.

If the line segment is called L_1 , we can write

$$\text{Length}(L_1) = \text{Inches}(1.5) = \text{Centimeters}(3.81).$$

Conversion between units is done by equating multiples of one unit to another:

$$\text{Centimeters}(2.54 \times d) = \text{Inches}(d).$$

Similar axioms can be written for pounds and kilograms, seconds and days, and dollars and cents. Measures can be used to describe objects as follows:

$$\text{Diameter}(\text{Basketball}_{12}) = \text{Inches}(9.5)$$

$$\text{ListPrice}(\text{Basketball}_{12}) = \$ (19)$$

$$\text{Weight}(\text{BunchOf}(\{\text{Apple}_1, \text{Apple}_2, \text{Apple}_3\})) = \text{Pounds}(2)$$

$$d \in \text{Days} \Rightarrow \text{Duration}(d) = \text{Hours}(24).$$

Note that $\$(1)$ is *not* a dollar bill—it is a price. One can have two dollar bills, but there is only one object named $\$(1)$. Note also that, while $\text{Inches}(0)$ and $\text{Centimeters}(0)$ refer to the same zero length, they are not identical to other zero measures, such as $\text{Seconds}(0)$.

Simple, quantitative measures are easy to represent. Other measures present more of a problem, because they have no agreed scale of values. Exercises have difficulty, desserts have deliciousness, and poems have beauty, yet numbers cannot be assigned to these qualities. One might, in a moment of pure accountancy, dismiss such properties as useless for the purpose of logical reasoning; or, still worse, attempt to impose a numerical scale on beauty. This would be a grave mistake, because it is unnecessary. The most important aspect of measures is not the particular numerical values, but the fact that measures can be *ordered*.

Although measures are not numbers, we can still compare them, using an ordering symbol such as $>$. For example, we might well believe that Norvig's exercises are tougher than Russell's, and that one scores less on tougher exercises:

$$e_1 \in \text{Exercises} \wedge e_2 \in \text{Exercises} \wedge \text{Wrote}(\text{Norvig}, e_1) \wedge \text{Wrote}(\text{Russell}, e_2) \Rightarrow$$

$$\text{Difficulty}(e_1) > \text{Difficulty}(e_2).$$

$$e_1 \in \text{Exercises} \wedge e_2 \in \text{Exercises} \wedge \text{Difficulty}(e_1) > \text{Difficulty}(e_2) \Rightarrow$$

$$\text{ExpectedScore}(e_1) < \text{ExpectedScore}(e_2).$$

This is enough to allow one to decide which exercises to do, even though no numerical values for difficulty were ever used. (One does, however, have to discover who wrote which exercises.) These sorts of monotonic relationships among measures form the basis for the field of **qualitative physics**, a subfield of AI that investigates how to reason about physical systems without plunging into detailed equations and numerical simulations. Qualitative physics is discussed in the historical notes section.

10.2.3 Objects: Things and stuff

The real world can be seen as consisting of primitive objects (e.g., atomic particles) and composite objects built from them. By reasoning at the level of large objects such as apples and cars, we can overcome the complexity involved in dealing with vast numbers of primitive objects individually. There is, however, a significant portion of reality that seems to defy any obvious **individuation**—division into distinct objects. We give this portion the generic name **stuff**. For example, suppose I have some butter and an aardvark in front of me. I can say there is one aardvark, but there is no obvious number of “butter-objects,” because any part of a butter-object is also a butter-object, at least until we get to very small parts indeed. This is

Individuation
Stuff

the major distinction between *stuff* and *things*. If we cut an aardvark in half, we do not get two aardvarks (unfortunately).

The English language distinguishes clearly between *stuff* and *things*. We say “an aardvark,” but, except in pretentious California restaurants, one cannot say “a butter.” Linguists distinguish between **count nouns**, such as aardvarks, holes, and theorems, and **mass nouns**, such as butter, water, and energy. Several competing ontologies claim to handle this distinction. Here we describe just one; the others are covered in the historical notes section.

To represent *stuff* properly, we begin with the obvious. We need to have as objects in our ontology at least the gross “lumps” of *stuff* we interact with. For example, we might recognize a lump of butter as the one left on the table the night before; we might pick it up, weigh it, sell it, or whatever. In these senses, it is an object just like the aardvark. Let us call it *Butter*₃. We also define the category *Butter*. Informally, its elements will be all those things of which one might say “It’s butter,” including *Butter*₃. With some caveats about very small parts that we will omit for now, any part of a butter-object is also a butter-object:

$$b \in \textit{Butter} \wedge \textit{PartOf}(p, b) \Rightarrow p \in \textit{Butter}.$$

We can now say that butter melts at around 30 degrees centigrade:

$$b \in \textit{Butter} \Rightarrow \textit{MeltingPoint}(b, \textit{Centigrade}(30)).$$

We could go on to say that butter is yellow, is less dense than water, is soft at room temperature, has a high fat content, and so on. On the other hand, butter has no particular size, shape, or weight. We can define more specialized categories of butter such as *UnsaltedButter*, which is also a kind of *stuff*. Note that the category *PoundOfButter*, which includes as members all butter-objects weighing one pound, is not a kind of *stuff*. If we cut a pound of butter in half, we do not, alas, get two pounds of butter.

What is actually going on is this: some properties are **intrinsic**: they belong to the very substance of the object, rather than to the object as a whole. When you cut an instance of *stuff* in half, the two pieces retain the intrinsic properties—things like density, boiling point, flavor, color, ownership, and so on. On the other hand, their **extrinsic** properties—weight, length, shape, and so on—are not retained under subdivision. A category of objects that includes in its definition only *intrinsic* properties is then a substance, or mass noun; a class that includes *any* extrinsic properties in its definition is a count noun. *Stuff* and *Thing* are the most general substance and object categories, respectively.

10.3 Events

In Section 7.7.1 we discussed actions: things that happen, such as *Shoot*_{*t*}; and fluents: aspects of the world that change, such as *HaveArrow*_{*t*}. Both were represented as propositions, and we used successor-state axioms to say that a fluent will be true at time $t + 1$ if the action at time t caused it to be true, or if it was already true at time t and the action did not cause it to be false. That was for a world in which actions are discrete, instantaneous, happen one at a time, and have no variation in how they are performed (that is, there is only one kind of *Shoot* action, there is no distinction between shooting quickly, slowly, nervously, etc.).

But as we move from simplistic domains to the real world, there is a much richer range of actions or events³ to deal with. Consider a continuous action, such as filling a bathtub. A

³ The terms “event” and “action” may be used interchangeably—they both mean “something that can happen.”

Count nouns
Mass noun

Intrinsic

Extrinsic

successor-state axiom can say that the tub is empty before the action and full when the action is done, but it can't talk about what happens *during* the action. It also can't easily describe two actions happening at the same time—such as brushing one's teeth while waiting for the tub to fill. To handle such cases we introduce an approach known as **event calculus**.

Event calculus

The objects of event calculus are events, fluents, and time points. $At(Shankar, Berkeley)$ is a fluent: an object that refers to the fact of Shankar being in Berkeley. The event E_1 of Shankar flying from San Francisco to Washington, D.C., is described as

$$E_1 \in Flyings \wedge Flyer(E_1, Shankar) \wedge Origin(E_1, SF) \wedge Destination(E_1, DC).$$

where *Flyings* is the category of all flying events. By reifying events we make it possible to add any amount of arbitrary information about them. For example, we can say that Shankar's flight was bumpy with $Bumpy(E_1)$. In an ontology where events are n -ary predicates, there would be no way to add extra information like this; moving to an $n + 1$ -ary predicate isn't a scalable solution.

To assert that a fluent is actually true starting at some point in time t_1 and continuing to time t_2 , we use the predicate T , as in $T(At(Shankar, Berkeley), t_1, t_2)$. Similarly, we use $Happens(E_1, t_1, t_2)$ to say that the event E_1 actually happened, starting at time t_1 and ending at time t_2 . The complete set of predicates for one version of the event calculus⁴ is:

$T(f, t_1, t_2)$	Fluent f is true for all times between t_1 and t_2
$Happens(e, t_1, t_2)$	Event e starts at time t_1 and ends at t_2
$Initiates(e, f, t)$	Event e causes fluent f to become true at time t
$Terminates(e, f, t)$	Event e causes fluent f to cease to be true at time t
$Initiated(f, t_1, t_2)$	Fluent f become true at some point between t_1 and t_2
$Terminated(f, t_1, t_2)$	Fluent f cease to be true at some point between t_1 and t_2
$t_1 < t_2$	Time point t_1 occurs before time t_2

We can describe the effects of a flying event:

$$E = Flyings(a, here, there) \wedge Happens(E, t_1, t_2) \Rightarrow \\ Terminates(E, At(a, here), t_1) \wedge Initiates(E, At(a, there), t_2)$$

We assume a distinguished event, *Start*, that describes the initial state by saying which fluents are true (using *Initiates*) or false (using *Terminated*) at the start time. We can then describe what fluents are true at what points in time with a pair of axioms for T and $\neg T$ that follow the same general format as the successor-state axioms: Assume an event happens between time t_1 and t_3 , and at t_2 somewhere in that time interval the event changes the value of fluent f , either initiating it (making it true) or terminating it (making it false). Then at time t_4 in the future, if no other intervening event has changed the fluent (either terminated or initiated it, respectively), then the fluent will have maintained its value. Formally, the axioms are:

$$Happens(e, t_1, t_3) \wedge Initiates(e, f, t_2) \wedge \neg Terminated(f, t_2, t_4) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \Rightarrow \\ T(f, t_2, t_4) \\ Happens(e, t_1, t_3) \wedge Terminates(e, f, t_2) \wedge \neg Initiated(f, t_2, t_4) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \Rightarrow \\ \neg T(f, t_2, t_4)$$

⁴ Our version is based on Shanahan (1999), but with some alterations.

where *Terminated* and *Initiated* are defined by:

$$\begin{aligned}
 \text{Terminated}(f, t_1, t_5) &\Leftrightarrow \\
 &\exists e, t_2, t_3, t_4 \text{ Happens}(e, t_2, t_4) \wedge \text{Terminates}(e, f, t_3) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5 \\
 \text{Initiated}(f, t_1, t_5) &\Leftrightarrow \\
 &\exists e, t_2, t_3, t_4 \text{ Happens}(e, t_2, t_4) \wedge \text{Initiates}(e, f, t_3) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5
 \end{aligned}$$

We can extend event calculus to represent simultaneous events (such as two people being necessary to ride a seesaw), exogenous events (such as the wind moving an object), continuous events (such as the rising of the tide), nondeterministic events (such as flipping a coin and having it come up heads or tails), and other complications.

10.3.1 Time

Event calculus opens us up to the possibility of talking about time points and time intervals. We will consider two kinds of time intervals: moments and extended intervals. The distinction is that only moments have zero duration:

$$\begin{aligned}
 &\text{Partition}(\{\text{Moments}, \text{ExtendedIntervals}\}, \text{Intervals}) \\
 &i \in \text{Moments} \Leftrightarrow \text{Duration}(i) = \text{Seconds}(0).
 \end{aligned}$$

Next we invent a time scale and associate points on that scale with moments, giving us absolute times. The time scale is arbitrary; we will measure it in seconds and say that the moment at midnight (GMT) on January 1, 1900, has time 0. The functions *Begin* and *End* pick out the earliest and latest moments in an interval, and the function *Time* delivers the point on the time scale for a moment. The function *Duration* gives the difference between the end time and the start time.

$$\begin{aligned}
 \text{Interval}(i) &\Rightarrow \text{Duration}(i) = (\text{Time}(\text{End}(i)) - \text{Time}(\text{Begin}(i))). \\
 \text{Time}(\text{Begin}(\text{AD1900})) &= \text{Seconds}(0). \\
 \text{Time}(\text{Begin}(\text{AD2001})) &= \text{Seconds}(3187324800). \\
 \text{Time}(\text{End}(\text{AD2001})) &= \text{Seconds}(3218860800). \\
 \text{Duration}(\text{AD2001}) &= \text{Seconds}(31536000).
 \end{aligned}$$

To make these numbers easier to read, we also introduce a function *Date*, which takes six arguments (hours, minutes, seconds, day, month, and year) and returns a time point:

$$\begin{aligned}
 \text{Time}(\text{Begin}(\text{AD2001})) &= \text{Date}(0, 0, 0, 1, \text{Jan}, 2001) \\
 \text{Date}(0, 20, 21, 24, 1, 1995) &= \text{Seconds}(3000000000).
 \end{aligned}$$

Two intervals *Meet* if the end time of the first equals the start time of the second. The complete set of interval relations (Allen, 1983) is shown below and in Figure 10.2:

$$\begin{aligned}
 \text{Meet}(i, j) &\Leftrightarrow \text{End}(i) = \text{Begin}(j) \\
 \text{Before}(i, j) &\Leftrightarrow \text{End}(i) < \text{Begin}(j) \\
 \text{After}(j, i) &\Leftrightarrow \text{Before}(i, j) \\
 \text{During}(i, j) &\Leftrightarrow \text{Begin}(j) < \text{Begin}(i) < \text{End}(i) < \text{End}(j) \\
 \text{Overlap}(i, j) &\Leftrightarrow \text{Begin}(i) < \text{Begin}(j) < \text{End}(i) < \text{End}(j) \\
 \text{Starts}(i, j) &\Leftrightarrow \text{Begin}(i) = \text{Begin}(j) \\
 \text{Finishes}(i, j) &\Leftrightarrow \text{End}(i) = \text{End}(j) \\
 \text{Equals}(i, j) &\Leftrightarrow \text{Begin}(i) = \text{Begin}(j) \wedge \text{End}(i) = \text{End}(j)
 \end{aligned}$$

These all have their intuitive meaning, with the exception of *Overlap*: we tend to think of overlap as symmetric (if *i* overlaps *j* then *j* overlaps *i*), but in this definition, *Overlap*(*i*, *j*) only is true if *i* begins before *j*. Experience has shown that this definition is more useful for

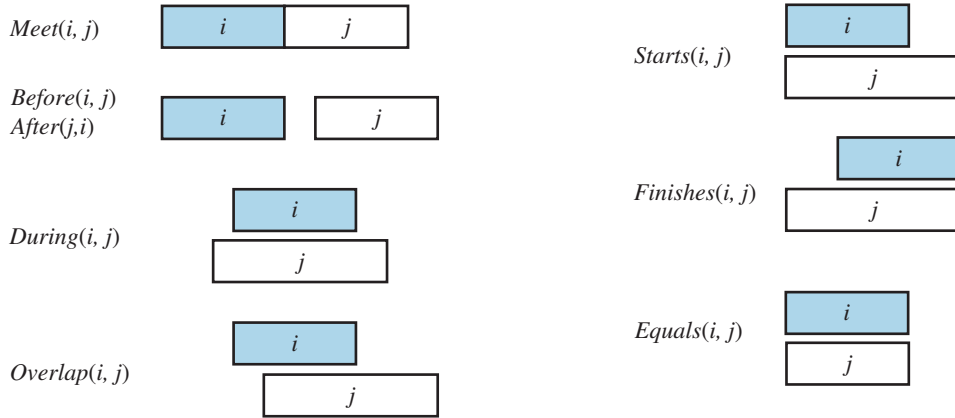


Figure 10.2 Predicates on time intervals.

writing axioms. To say that the reign of Elizabeth II immediately followed that of George VI, and the reign of Elvis overlapped with the 1950s, we can write the following:

Meets(*ReignOf*(*GeorgeVI*), *ReignOf*(*ElizabethII*)).
Overlap(*Fifties*, *ReignOf*(*Elvis*)).
Begin(*Fifties*) = *Begin*(*AD1950*).
End(*Fifties*) = *End*(*AD1959*).

10.3.2 Fluents and objects

Physical objects can be viewed as generalized events, in the sense that a physical object is a chunk of space–time. For example, *USA* can be thought of as an event that began in 1776 as a union of 13 states and is still in progress today as a union of 50. We can describe the changing properties of *USA* using state fluents, such as *Population*(*USA*). A property of *USA* that changes every four or eight years, barring mishaps, is its president. One might propose that *President*(*USA*) is a logical term that denotes a different object at different times.

Unfortunately, this is not possible, because a term denotes exactly one object in a given model structure. (The term *President*(*USA*, *t*) can denote different objects, depending on the value of *t*, but our ontology keeps time indices separate from fluents.) The only possibility is that *President*(*USA*) denotes a single object that consists of different people at different times. It is the object that is George Washington from 1789 to 1797, John Adams from 1797 to 1801, and so on, as in Figure 10.3. To say that George Washington was president throughout 1790, we can write

$T(\text{Equals}(\text{President}(\text{USA}), \text{GeorgeWashington}), \text{Begin}(\text{AD1790}), \text{End}(\text{AD1790})).$

We use the function symbol *Equals* rather than the standard logical predicate =, because we cannot have a predicate as an argument to *T*, and because the interpretation is *not* that *GeorgeWashington* and *President*(*USA*) are logically identical in 1790; logical identity is not something that can change over time. The identity is between the subevents of the objects *President*(*USA*) and *GeorgeWashington* that are defined by the period 1790.

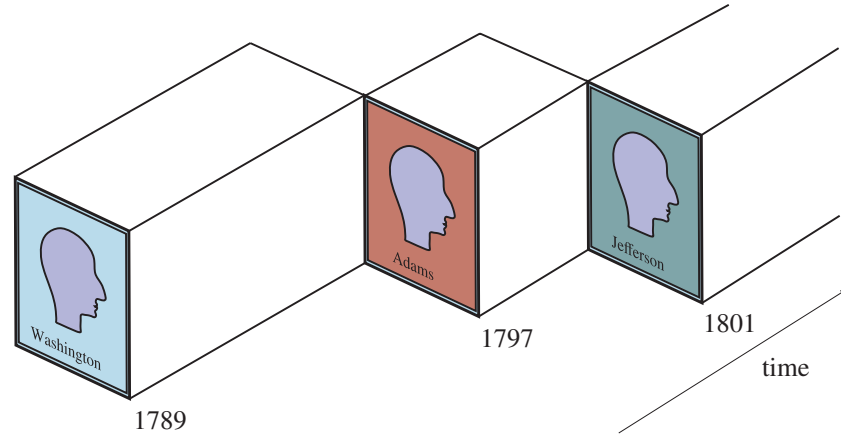


Figure 10.3 A schematic view of the object *President(USA)* for the early years.

10.4 Mental Objects and Modal Logic

The agents we have constructed so far have beliefs and can deduce new beliefs. Yet none of them has any knowledge *about* beliefs or *about* deduction. Knowledge about one's own knowledge and reasoning processes is useful for controlling inference. For example, suppose Alice asks "what is the square root of 1764" and Bob replies "I don't know." If Alice insists "think harder," Bob should realize that with some more thought, this question can in fact be answered. On the other hand, if the question were "Is the president sitting down right now?" then Bob should realize that thinking harder is unlikely to help. Knowledge about the knowledge of other agents is also important; Bob should realize that the president does know.

What we need is a model of the mental objects that are in someone's head (or something's knowledge base) and of the mental processes that manipulate those mental objects. The model does not have to be detailed. We do not have to be able to predict how many milliseconds it will take for a particular agent to make a deduction. We will be happy just to be able to conclude that mother knows whether or not she is sitting.

We begin with the **propositional attitudes** that an agent can have toward mental objects: attitudes such as *Believes*, *Knows*, *Wants*, and *Informs*. The difficulty is that these attitudes do not behave like "normal" predicates. For example, suppose we try to assert that Lois knows that Superman can fly:

$$\text{Knows}(\text{Lois}, \text{CanFly}(\text{Superman})).$$

One minor issue with this is that we normally think of *CanFly(Superman)* as a sentence, but here it appears as a term. That issue can be patched up by reifying *CanFly(Superman)*; making it a fluent. A more serious problem is that, if it is true that Superman is Clark Kent, then we must conclude that Lois knows that Clark can fly, which is wrong because (in most versions of the story) Lois does *not* know that Clark is Superman.

$$\begin{aligned} &(\text{Superman} = \text{Clark}) \wedge \text{Knows}(\text{Lois}, \text{CanFly}(\text{Superman})) \\ &\models \text{Knows}(\text{Lois}, \text{CanFly}(\text{Clark})) \end{aligned}$$

This is a consequence of the fact that equality reasoning is built into logic. Normally that is

a good thing; if our agent knows that $2 + 2 = 4$ and $4 < 5$, then we want our agent to know that $2 + 2 < 5$. This property is called **referential transparency**—it doesn't matter what term a logic uses to refer to an object, what matters is the object that the term names. But for propositional attitudes like *believes* and *knows*, we would like to have referential opacity—the terms used *do* matter, because not all agents know which terms are co-referential.

Referential transparency

We could patch this up with even more reification: we could have one object to represent Clark/Superman, another object to represent the person that Lois knows as Clark, and yet another for the person Lois knows as Superman. However, this proliferation of objects means that the sentences we want to write quickly become verbose and clumsy.

Modal logic is designed to address this problem. Regular logic is concerned with a single modality, the modality of truth, allowing us to express “ P is true” or “ P is false.” Modal logic includes special **modal operators** that take sentences (rather than terms) as arguments. For example, “ A knows P ” is represented with the notation $\mathbf{K}_A P$, where \mathbf{K} is the modal operator for knowledge. It takes two arguments, an agent (written as the subscript) and a sentence. The syntax of modal logic is the same as first-order logic, except that sentences can also be formed with modal operators.

Modal logic

Modal operators

The semantics of modal logic is more complicated. In first-order logic a **model** contains a set of objects and an interpretation that maps each name to the appropriate object, relation, or function. In modal logic we want to be able to consider both the possibility that Superman's secret identity is Clark and the possibility that it isn't.

Therefore, we will need a more complicated model, one that consists of a collection of **possible worlds** rather than just one true world. The worlds are connected in a graph by **accessibility relations**, one relation for each modal operator. We say that world w_1 is accessible from world w_0 with respect to the modal operator \mathbf{K}_A if everything in w_1 is consistent with what A knows in w_0 . As an example, in the real world, Bucharest is the capital of Romania, but for an agent that did not know that, a world where the capital of Romania is, say, Sofia is accessible. Hopefully a world where $2 + 2 = 5$ would not be accessible to any agent.

Possible world

Accessibility relation

In general, a knowledge atom $\mathbf{K}_A P$ is true in world w if and only if P is true in every world accessible from w . The truth of more complex sentences is derived by recursive application of this rule and the normal rules of first-order logic. That means that modal logic can be used to reason about nested knowledge sentences: what one agent knows about another agent's knowledge. For example, we can say that even though Lois doesn't know whether Superman's secret identity is Clark Kent, she does know that Clark knows:

$$\mathbf{K}_{\text{Lois}}[\mathbf{K}_{\text{Clark}}\text{Identity}(\text{Superman}, \text{Clark}) \vee \mathbf{K}_{\text{Clark}}\neg\text{Identity}(\text{Superman}, \text{Clark})]$$

Modal logic solves some tricky issues with the interplay of quantifiers and knowledge. The English sentence “Bond knows that someone is a spy” is ambiguous. The first reading is that there is a particular someone who Bond knows is a spy; we can write this as

$$\exists x \mathbf{K}_{\text{Bond}}\text{Spy}(x),$$

which in modal logic means that there is an x that, in all accessible worlds, Bond knows to be a spy. The second reading is that Bond just knows that there is at least one spy:

$$\mathbf{K}_{\text{Bond}}\exists x \text{Spy}(x).$$

The modal logic interpretation is that in each accessible world there is an x that is a spy, but it need not be the same x in each world.

Now that we have a modal operator for knowledge, we can write axioms for it. First, we can say that agents are able to draw conclusions; if an agent knows P and knows that P implies Q , then the agent knows Q :

$$(\mathbf{K}_a P \wedge \mathbf{K}_a(P \Rightarrow Q)) \Rightarrow \mathbf{K}_a Q.$$

From this (and a few other rules about logical identities) we can establish that $\mathbf{K}_A(P \vee \neg P)$ is a tautology; every agent knows every proposition P is either true or false. On the other hand, $(\mathbf{K}_A P) \vee (\mathbf{K}_A \neg P)$ is not a tautology; in general, there will be lots of propositions that an agent does not know to be true and does not know to be false.

It is said (going back to Plato) that knowledge is justified true belief. That is, if it is true, if you believe it, and if you have an unassailably good reason, then you know it. That means that if you know something, it must be true, and we have the axiom:

$$\mathbf{K}_a P \Rightarrow P.$$

Furthermore, logical agents (but not all people) are able to introspect on their own knowledge. If they know something, then they know that they know it:

$$\mathbf{K}_a P \Rightarrow \mathbf{K}_a(\mathbf{K}_a P).$$

Logical omniscience

We can define similar axioms for belief (often denoted by \mathbf{B}) and other modalities. However, one problem with the modal logic approach is that it assumes **logical omniscience** on the part of agents. That is, if an agent knows a set of axioms, then it knows all consequences of those axioms. This is on shaky ground even for the somewhat abstract notion of knowledge, but it seems even worse for belief, because belief has more connotation of referring to things that are physically represented in the agent, not just potentially derivable.

There have been attempts to define a form of limited rationality for agents—to say that agents believe only those assertions that can be derived with the application of no more than k reasoning steps, or no more than s seconds of computation. These attempts have been generally unsatisfactory.

10.4.1 Other modal logics

Many modal logics have been proposed, for different modalities besides knowledge. One proposal is to add modal operators for *possibility* and *necessity*: it is possibly true that one of the authors of this book is sitting down right now, and it is necessarily true that $2 + 2 = 4$.

Linear temporal logic

As mentioned in Section 8.1.2, some logicians favor modalities related to time. In **linear temporal logic**, we add the following modal operators:

- $\mathbf{X} P$: “ P will be true in the next time step”
- $\mathbf{F} P$: “ P will eventually (Finally) be true in some future time step”
- $\mathbf{G} P$: “ P is always (Globally) true”
- $P \mathbf{U} Q$: “ P remains true until Q occurs”

Sometimes there are additional operators that can be derived from these. Adding these modal operators makes the logic itself more complex (and thus makes it harder for a logical inference algorithm to find a proof). But the operators also allow us to state certain facts in a more succinct form (which makes logical inference faster). The choice of which logic to use is similar to the choice of which programming language to use: pick one that is appropriate to your task, that is familiar to you and the others who will share your work, and that is efficient enough for your purposes.

10.5 Reasoning Systems for Categories

Categories are the primary building blocks of large-scale knowledge representation schemes. This section describes systems specially designed for organizing and reasoning with categories. There are two closely related families of systems: **semantic networks** provide graphical aids for visualizing a knowledge base and efficient algorithms for inferring properties of an object on the basis of its category membership; and **description logics** provide a formal language for constructing and combining category definitions and efficient algorithms for deciding subset and superset relationships between categories.

Semantic networks

Description logics

10.5.1 Semantic networks

In 1909, Charles S. Peirce proposed a graphical notation of nodes and edges called **existential graphs** that he called “the logic of the future.” Thus began a long-running debate between advocates of “logic” and advocates of “semantic networks.” Unfortunately, the debate obscured the fact that semantic networks *are* a form of logic. The notation that semantic networks provide for certain kinds of sentences is often more convenient, but if we strip away the “human interface” issues, the underlying concepts—objects, relations, quantification, and so on—are the same.

Existential graphs

There are many variants of semantic networks, but all are capable of representing individual objects, categories of objects, and relations among objects. A typical graphical notation displays object or category names in ovals or boxes, and connects them with labeled links. For example, Figure 10.4 has a *MemberOf* link between *Mary* and *FemalePersons*, corresponding to the logical assertion $Mary \in FemalePersons$; similarly, the *SisterOf* link between *Mary* and *John* corresponds to the assertion $SisterOf(Mary, John)$. We can connect categories using *SubsetOf* links, and so on. It is such fun drawing bubbles and arrows that one can get carried away. For example, we know that persons have female persons as mothers, so can we draw a *HasMother* link from *Persons* to *FemalePersons*? The answer is no, because *HasMother* is a relation between a person and his or her mother, and categories do not have mothers.⁵

For this reason, we have used a special notation—the double-boxed link—in Figure 10.4. This link asserts that

$$\forall x \ x \in Persons \Rightarrow [\forall y \ HasMother(x, y) \Rightarrow y \in FemalePersons].$$

We might also want to assert that persons have two legs—that is,

$$\forall x \ x \in Persons \Rightarrow Legs(x, 2).$$

As before, we need to be careful not to assert that a category has legs; the single-boxed link in Figure 10.4 is used to assert properties of every member of a category.

The semantic network notation makes it convenient to perform **inheritance** reasoning of the kind introduced in Section 10.2. For example, by virtue of being a person, Mary inherits the property of having two legs. Thus, to find out how many legs Mary has, the inheritance algorithm follows the *MemberOf* link from *Mary* to the category she belongs to, and then

⁵ Several early systems failed to distinguish between properties of members of a category and properties of the category as a whole. This can lead directly to inconsistencies, as pointed out by Drew McDermott (1976) in his article “Artificial Intelligence Meets Natural Stupidity.” Another common problem was the use of *IsA* links for both subset and membership relations, in correspondence with English usage: “a cat is a mammal” and “Fifi is a cat.” See Exercise 10.NATS for more on these issues.

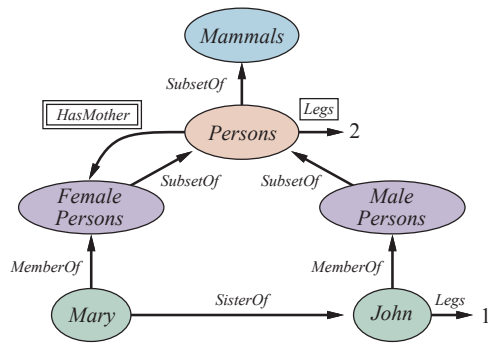


Figure 10.4 A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

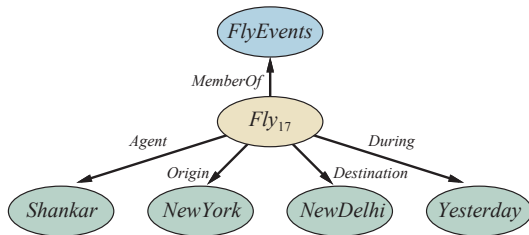


Figure 10.5 A fragment of a semantic network showing the representation of the logical assertion *Fly(Shankar, NewYork, NewDelhi, Yesterday)*.

follows *SubsetOf* links up the hierarchy until it finds a category for which there is a boxed *Legs* link—in this case, the *Persons* category. The simplicity and efficiency of this inference mechanism, compared with semidecidable logical theorem proving, has been one of the main attractions of semantic networks.

Multiple inheritance

Inheritance becomes complicated when an object can belong to more than one category or when a category can be a subset of more than one other category; this is called **multiple inheritance**. In such cases, the inheritance algorithm might find two or more conflicting values answering the query. For this reason, multiple inheritance is banned in some **object-oriented programming** (OOP) languages, such as Java, that use inheritance in a class hierarchy. It is usually allowed in semantic networks, but we defer discussion of that until Section 10.6.

The reader might have noticed an obvious drawback of semantic network notation, compared to first-order logic: the fact that links between bubbles represent only *binary* relations. For example, the sentence *Fly(Shankar, NewYork, NewDelhi, Yesterday)* cannot be asserted directly in a semantic network. Nonetheless, we *can* obtain the effect of *n*-ary assertions by reifying the proposition itself as an event belonging to an appropriate event category. Figure 10.5 shows the semantic network structure for this particular event. Notice that the restriction to binary relations forces the creation of a rich ontology of reified concepts.

Reification of propositions makes it possible to represent every ground, function-free atomic sentence of first-order logic in the semantic network notation. Certain kinds of universally quantified sentences can be asserted using inverse links and the singly boxed and doubly boxed arrows applied to categories, but that still leaves us a long way short of full first-order logic. Negation, disjunction, nested function symbols, and existential quantification are all missing. Now it is *possible* to extend the notation to make it equivalent to first-order logic—as in Peirce’s existential graphs—but doing so negates one of the main advantages of semantic networks, which is the simplicity and transparency of the inference processes. Designers can build a large network and still have a good idea about what queries will be efficient, because (a) it is easy to visualize the steps that the inference procedure will go through and (b) in some cases the query language is so simple that difficult queries cannot be posed.

In cases where the expressive power proves to be too limiting, many semantic network systems provide for **procedural attachment** to fill in the gaps. Procedural attachment is a technique whereby a query about (or sometimes an assertion of) a certain relation results in a call to a special procedure designed for that relation rather than a general inference algorithm.

Procedural attachment

One of the most important aspects of semantic networks is their ability to represent **default values** for categories. Examining Figure 10.4 carefully, one notices that John has one leg, despite the fact that he is a person and all persons have two legs. In a strictly logical KB, this would be a contradiction, but in a semantic network, the assertion that all persons have two legs has only default status; that is, a person is assumed to have two legs unless this is contradicted by more specific information. The default semantics is enforced naturally by the inheritance algorithm, because it follows links upwards from the object itself (John in this case) and stops as soon as it finds a value. We say that the default is **overridden** by the more specific value. Notice that we could also override the default number of legs by creating a category of *OneLeggedPersons*, a subset of *Persons* of which *John* is a member.

Default value

Overriding

We can retain a strictly logical semantics for the network if we say that the *Legs* assertion for *Persons* includes an exception for John:

$$\forall x \ x \in \text{Persons} \wedge x \neq \text{John} \Rightarrow \text{Legs}(x, 2).$$

For a *fixed* network, this is semantically adequate but will be much less concise than the network notation itself if there are lots of exceptions. For a network that will be updated with more assertions, however, such an approach fails—we really want to say that any persons as yet unknown with one leg are exceptions too. Section 10.6 goes into more depth on this issue and on default reasoning in general.

10.5.2 Description logics

The syntax of first-order logic is designed to make it easy to say things about objects. **Description logics** are notations that are designed to make it easier to describe definitions and properties of categories. Description logic systems evolved from semantic networks in response to pressure to formalize what the networks mean while retaining the emphasis on taxonomic structure as an organizing principle.

Description logic

The principal inference tasks for description logics are **subsumption** (checking if one category is a subset of another by comparing their definitions) and **classification** (checking whether an object belongs to a category). Some systems also include **consistency** of a category definition—whether the membership criteria are logically satisfiable.

Subsumption

Classification

Consistency

```

Concept → Thing | ConceptName
           | And(Concept,...)
           | All(RoleName,Concept)
           | AtLeast(Integer,RoleName)
           | AtMost(Integer,RoleName)
           | Fills(RoleName,IndividualName,...)
           | SameAs(Path,Path)
           | OneOf(IndividualName,...)
Path → [RoleName,...]
ConceptName → Adult | Female | Male | ...
RoleName → Spouse | Daughter | Son | ...

```

Figure 10.6 The syntax of descriptions in a subset of the CLASSIC language.

The CLASSIC language (Borgida *et al.*, 1989) is a typical description logic. The syntax of CLASSIC descriptions is shown in Figure 10.6.⁶ For example, to say that bachelors are unmarried adult males we would write

$$\text{Bachelor} = \text{And}(\text{Unmarried}, \text{Adult}, \text{Male}).$$

The equivalent in first-order logic would be

$$\text{Bachelor}(x) \Leftrightarrow \text{Unmarried}(x) \wedge \text{Adult}(x) \wedge \text{Male}(x).$$

Notice that the description logic has an algebra of operations on predicates, which of course we can't do in first-order logic. Any description in CLASSIC can be translated into an equivalent first-order sentence, but some descriptions are more straightforward in CLASSIC. For example, to describe the set of men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments, we would use

$$\begin{aligned} &\text{And}(\text{Man}, \text{AtLeast}(3, \text{Son}), \text{AtMost}(2, \text{Daughter}), \\ &\quad \text{All}(\text{Son}, \text{And}(\text{Unemployed}, \text{Married}, \text{All}(\text{Spouse}, \text{Doctor}))), \\ &\quad \text{All}(\text{Daughter}, \text{And}(\text{Professor}, \text{Fills}(\text{Department}, \text{Physics}, \text{Math}))))). \end{aligned}$$

We leave it as an exercise to translate this into first-order logic.

Perhaps the most important aspect of description logics is their emphasis on tractability of inference. A problem instance is solved by describing it and then asking if it is subsumed by one of several possible solution categories. In standard first-order logic systems, predicting the solution time is often impossible. It is frequently left to the user to engineer the representation to detour around sets of sentences that seem to be causing the system to take several

⁶ Notice that the language does *not* allow one to simply state that one concept, or category, is a subset of another. This is a deliberate policy: subsumption between categories must be derivable from some aspects of the descriptions of the categories. If not, then something is missing from the descriptions.

weeks to solve a problem. The thrust in description logics, on the other hand, is to ensure that subsumption-testing can be solved in time polynomial in the size of the descriptions.⁷

This sounds wonderful in principle, until one realizes that it can only have one of two consequences: either hard problems cannot be stated at all, or they require exponentially large descriptions! However, the tractability results do shed light on what sorts of constructs cause problems and thus help the user to understand how different representations behave. For example, description logics usually lack *negation* and *disjunction*. Each forces first-order logical systems to go through a potentially exponential case analysis in order to ensure completeness. CLASSIC allows only a limited form of disjunction in the *Fills* and *OneOf* constructs, which permit disjunction over explicitly enumerated individuals but not over descriptions. With disjunctive descriptions, nested definitions can lead easily to an exponential number of alternative routes by which one category can subsume another.

10.6 Reasoning with Default Information

In the preceding section, we saw a simple example of an assertion with default status: people have two legs. This default can be overridden by more specific information, such as that Long John Silver has one leg. We saw that the inheritance mechanism in semantic networks implements the overriding of defaults in a simple and natural way. In this section, we study defaults more generally, with a view toward understanding the *semantics* of defaults rather than just providing a procedural mechanism.

10.6.1 Circumscription and default logic

We have seen two examples of reasoning processes that violate the **monotonicity** property of logic that was proved in Chapter 7.⁸ In this chapter we saw that a property inherited by all members of a category in a semantic network could be overridden by more specific information for a subcategory. In Section 9.4.4, we saw that under the closed-world assumption, if a proposition α is not mentioned in KB then $KB \models \neg\alpha$, but $KB \wedge \alpha \models \alpha$.

Monotonicity

Simple introspection suggests that these failures of monotonicity are widespread in commonsense reasoning. It seems that humans often “jump to conclusions.” For example, when one sees a car parked on the street, one is normally willing to believe that it has four wheels even though only three are visible. Now, probability theory can certainly provide a conclusion that the fourth wheel exists with high probability; yet, for most people, the possibility that the car does not have four wheels *will not arise unless some new evidence presents itself*. Thus, it seems that the four-wheel conclusion is reached *by default*, in the absence of any reason to doubt it. If new evidence arrives—for example, if one sees the owner carrying a wheel and notices that the car is jacked up—then the conclusion can be retracted. This kind of reasoning is said to exhibit **nonmonotonicity**, because the set of beliefs does not grow monotonically over time as new evidence arrives. **Nonmonotonic logics** have been devised with modified notions of truth and entailment in order to capture such behavior. We will look at two such logics that have been studied extensively: circumscription and default logic.

Nonmonotonicity

Nonmonotonic logic

Circumscription can be seen as a more powerful and precise version of the closed-world

Circumscription

⁷ CLASSIC provides efficient subsumption testing in practice, but the worst-case run time is exponential.

⁸ Recall that monotonicity requires all entailed sentences to remain entailed after new sentences are added to the KB. That is, if $KB \models \alpha$ then $KB \wedge \beta \models \alpha$.

assumption. The idea is to specify particular predicates that are assumed to be “as false as possible”—that is, false for every object except those for which they are known to be true. For example, suppose we want to assert the default rule that birds fly. We would introduce a predicate, say $Abnormal_1(x)$, and write

$$Bird(x) \wedge \neg Abnormal_1(x) \Rightarrow Flies(x).$$

If we say that $Abnormal_1$ is to be **circumscribed**, a circumscriptive reasoner is entitled to assume $\neg Abnormal_1(x)$ unless $Abnormal_1(x)$ is known to be true. This allows the conclusion $Flies(Tweety)$ to be drawn from the premise $Bird(Tweety)$, but the conclusion no longer holds if $Abnormal_1(Tweety)$ is asserted.

Model preference

Circumscription can be viewed as an example of a **model preference** logic. In such logics, a sentence is entailed (with default status) if it is true in all *preferred* models of the KB, as opposed to the requirement of truth in *all* models in classical logic. For circumscription, one model is preferred to another if it has fewer abnormal objects.⁹ Let us see how this idea works in the context of multiple inheritance in semantic networks. The standard example for which multiple inheritance is problematic is called the “Nixon diamond.” It arises from the observation that Richard Nixon was both a Quaker (and hence by default a pacifist) and a Republican (and hence by default not a pacifist). We can write this as follows:

$$\begin{aligned} & Republican(Nixon) \wedge Quaker(Nixon). \\ & Republican(x) \wedge \neg Abnormal_2(x) \Rightarrow \neg Pacifist(x). \\ & Quaker(x) \wedge \neg Abnormal_3(x) \Rightarrow Pacifist(x). \end{aligned}$$

If we circumscribe $Abnormal_2$ and $Abnormal_3$, there are two preferred models: one in which $Abnormal_2(Nixon)$ and $Pacifist(Nixon)$ are true and one in which $Abnormal_3(Nixon)$ and $\neg Pacifist(Nixon)$ are true. Thus, the circumscriptive reasoner remains properly agnostic as to whether Nixon was a pacifist. If we wish, in addition, to assert that religious beliefs take precedence over political beliefs, we can use a formalism called **prioritized circumscription** to give preference to models where $Abnormal_3$ is minimized.

Prioritized circumscription

Default logic is a formalism in which **default rules** can be written to generate contingent, nonmonotonic conclusions. A default rule looks like this:

$$Bird(x) : Flies(x) / Flies(x).$$

This rule means that if $Bird(x)$ is true, and if $Flies(x)$ is consistent with the knowledge base, then $Flies(x)$ may be concluded by default. In general, a default rule has the form

$$P : J_1, \dots, J_n / C$$

where P is called the prerequisite, C is the conclusion, and J_i are the justifications—if any one of them can be proven false, then the conclusion cannot be drawn. Any variable that appears in J_i or C must also appear in P . The Nixon-diamond example can be represented in default logic with one fact and two default rules:

$$\begin{aligned} & Republican(Nixon) \wedge Quaker(Nixon). \\ & Republican(x) : \neg Pacifist(x) / \neg Pacifist(x). \\ & Quaker(x) : Pacifist(x) / Pacifist(x). \end{aligned}$$

⁹ For the closed-world assumption, one model is preferred to another if it has fewer true atoms—that is, preferred models are **minimal** models. There is a natural connection between the closed-world assumption and definite-clause KBs, because the fixed point reached by forward chaining on definite-clause KBs is the unique minimal model. See page 249 for more on this point.

Default logic Default rules

Extension

To interpret what the default rules mean, we define the notion of an **extension** of a default theory to be a maximal set of consequences of the theory. That is, an extension S consists of the original known facts and a set of conclusions from the default rules, such that no additional conclusions can be drawn from S , and the justifications of every default conclusion in S are consistent with S . As in the case of the preferred models in circumscription, we have two possible extensions for the Nixon diamond: one wherein he is a pacifist and one wherein he is not. Prioritized schemes exist in which some default rules can be given precedence over others, allowing some ambiguities to be resolved.

Since 1980, when nonmonotonic logics were first proposed, a great deal of progress has been made in understanding their mathematical properties. There are still unresolved questions, however. For example, if “Cars have four wheels” is false, what does it mean to have it in one’s knowledge base? What is a good set of default rules to have? If we cannot decide, for each rule separately, whether it belongs in our knowledge base, then we have a serious problem of nonmodularity. Finally, how can beliefs that have default status be used to make decisions? This is probably the hardest issue for default reasoning.

Decisions often involve tradeoffs, and one therefore needs to compare the *strengths* of belief in the outcomes of different actions, and the *costs* of making a wrong decision. In cases where the same kinds of decisions are being made repeatedly, it is possible to interpret default rules as “threshold probability” statements. For example, the default rule “My brakes are always OK” really means “The probability that my brakes are OK, given no other information, is sufficiently high that the optimal decision is for me to drive without checking them.” When the decision context changes—for example, when one is driving a heavily laden truck down a steep mountain road—the default rule suddenly becomes inappropriate, even though there is no new evidence of faulty brakes. These considerations have led researchers to consider how to embed default reasoning within probability theory or utility theory.

10.6.2 Truth maintenance systems

We have seen that many of the inferences drawn by a knowledge representation system will have only default status, rather than being absolutely certain. Inevitably, some of these inferred facts will turn out to be wrong and will have to be retracted in the face of new information. This process is called **belief revision**.¹⁰ Suppose that a knowledge base KB contains a sentence P —perhaps a default conclusion recorded by a forward-chaining algorithm, or perhaps just an incorrect assertion—and we want to execute $\text{TELL}(KB, \neg P)$. To avoid creating a contradiction, we must first execute $\text{RETRACT}(KB, P)$. This sounds easy enough. Problems arise, however, if any *additional* sentences were inferred from P and asserted in the KB. For example, the implication $P \Rightarrow Q$ might have been used to add Q . The obvious “solution”—retracting all sentences inferred from P —fails because such sentences may have other justifications besides P . For example, if R and $R \Rightarrow Q$ are also in the KB, then Q does not have to be removed after all. **Truth maintenance systems**, or TMSs, are designed to handle exactly these kinds of complications.

Belief revision

Truth maintenance system

One simple approach to truth maintenance is to keep track of the order in which sentences are told to the knowledge base by numbering them from P_1 to P_n . When the call

¹⁰ Belief revision is often contrasted with **belief update**, which occurs when a knowledge base is revised to reflect a change in the world rather than new information about a fixed world. Belief update combines belief revision with reasoning about time and change; it is also related to the process of **filtering** described in Chapter 14.

RETRACT(KB, P_i) is made, the system reverts to the state just before P_i was added, thereby removing both P_i and any inferences that were derived from P_i . The sentences P_{i+1} through P_n can then be added again. This is simple, and it guarantees that the knowledge base will be consistent, but retracting P_i requires retracting and reasserting $n - i$ sentences as well as undoing and redoing all the inferences drawn from those sentences. For systems to which many facts are being added—such as large commercial databases—this is impractical.

JTMS

Justification

A more efficient approach is the justification-based truth maintenance system, or **JTMS**. In a JTMS, each sentence in the knowledge base is annotated with a **justification** consisting of the set of sentences from which it was inferred. For example, if the knowledge base already contains $P \Rightarrow Q$, then TELL(P) will cause Q to be added with the justification $\{P, P \Rightarrow Q\}$. In general, a sentence can have any number of justifications. Justifications make retraction efficient. Given the call RETRACT(P), the JTMS will delete exactly those sentences for which P is a member of every justification. So, if a sentence Q had the single justification $\{P, P \Rightarrow Q\}$, it would be removed; if it had the additional justification $\{P, P \vee R \Rightarrow Q\}$, it would still be removed; but if it also had the justification $\{R, P \vee R \Rightarrow Q\}$, then it would be spared. In this way, the time required for retraction of P depends only on the number of sentences derived from P rather than on the number of sentences added after P .

The JTMS assumes that sentences that are considered once will probably be considered again, so rather than deleting a sentence from the knowledge base entirely when it loses all justifications, we merely mark the sentence as being *out* of the knowledge base. If a subsequent assertion restores one of the justifications, then we mark the sentence as being back *in*. In this way, the JTMS retains all the inference chains that it uses and need not rederive sentences when a justification becomes valid again.

In addition to handling the retraction of incorrect information, TMSs can be used to speed up the analysis of multiple hypothetical situations. Suppose, for example, that the Romanian Olympic Committee is choosing sites for the swimming, athletics, and equestrian events at the 2048 Games to be held in Romania. For example, let the first hypothesis be *Site(Swimming, Pitesti)*, *Site(Athletics, Bucharest)*, and *Site(Equestrian, Arad)*.

A great deal of reasoning must then be done to work out the logistical consequences and hence the desirability of this selection. If we want to consider *Site(Athletics, Sibiu)* instead, the TMS avoids the need to start again from scratch. Instead, we simply retract *Site(Athletics, Bucharest)* and assert *Site(Athletics, Sibiu)* and the TMS takes care of the necessary revisions. Inference chains generated from the choice of Bucharest can be reused with Sibiu, provided that the conclusions are the same.

ATMS

An assumption-based truth maintenance system, or **ATMS**, makes this type of context-switching between hypothetical worlds particularly efficient. In a JTMS, the maintenance of justifications allows you to move quickly from one state to another by making a few retractions and assertions, but at any time only one state is represented. An ATMS represents *all* the states that have ever been considered at the same time. Whereas a JTMS simply labels each sentence as being *in* or *out*, an ATMS keeps track, for each sentence, of which assumptions would cause the sentence to be true. In other words, each sentence has a label that consists of a set of assumption sets. The sentence is true just in those cases in which all the assumptions in one of the assumption sets are true.

Explanation

Truth maintenance systems also provide a mechanism for generating **explanations**. Technically, an explanation of a sentence P is a set of sentences E such that E entails P . If the

sentences in E are already known to be true, then E simply provides a sufficient basis for proving that P must be the case. But explanations can also include **assumptions**—sentences that are not known to be true, but would suffice to prove P if they were true. For example, if your car won't start, you probably don't have enough information to definitively prove the reason for the problem. But a reasonable explanation might include the assumption that the battery is dead. This, combined with knowledge of how cars operate, explains the observed nonbehavior. In most cases, we will prefer an explanation E that is minimal, meaning that there is no proper subset of E that is also an explanation. An ATMS can generate explanations for the “car won't start” problem by making assumptions (such as “no gas in car” or “battery dead”) in any order we like, even if some assumptions are contradictory. Then we look at the label for the sentence “car won't start” to read off the sets of assumptions that would justify the sentence.

Assumption

The exact algorithms used to implement truth maintenance systems are a little complicated, and we do not cover them here. The computational complexity of the truth maintenance problem is at least as great as that of propositional inference—that is, NP-hard. Therefore, you should not expect truth maintenance to be a panacea. When used carefully, however, a TMS can provide a substantial increase in the ability of a logical system to handle complex environments and hypotheses.

Summary

By delving into the details of how one represents a variety of knowledge, we hope we have given the reader a sense of how real knowledge bases are constructed and a feeling for the interesting philosophical issues that arise. The major points are as follows:

- Large-scale knowledge representation requires a general-purpose ontology to organize and tie together the various specific domains of knowledge.
- A general-purpose ontology needs to cover a wide variety of knowledge and should be capable, in principle, of handling any domain.
- Building a large, general-purpose ontology is a significant challenge that has yet to be fully realized, although current frameworks seem to be quite robust.
- We presented an **upper ontology** based on categories and the event calculus. We covered categories, subcategories, parts, structured objects, measurements, substances, events, time and space, change, and beliefs.
- Natural kinds cannot be defined completely in logic, but properties of natural kinds can be represented.
- Actions, events, and time can be represented with the event calculus. Such representations enable an agent to construct sequences of actions and make logical inferences about what will be true when these actions happen.
- Special-purpose representation systems, such as **semantic networks** and **description logics**, have been devised to help in organizing a hierarchy of categories. **Inheritance** is an important form of inference, allowing the properties of objects to be deduced from their membership in categories.

- The **closed-world assumption**, as implemented in logic programs, provides a simple way to avoid having to specify lots of negative information. It is best interpreted as a **default** that can be overridden by additional information.
- **Nonmonotonic logics**, such as **circumscription** and **default logic**, are intended to capture default reasoning in general.
- **Truth maintenance systems** handle knowledge updates and revisions efficiently.
- It is difficult to construct large ontologies by hand; extracting knowledge from text makes the job easier.

Bibliographical and Historical Notes

Briggs (1985) claims that knowledge representation research began with first millennium BCE Indian theorizing about the grammar of Shastric Sanskrit. Western philosophers trace their work on the subject back to c. 300 BCE in Aristotle's *Metaphysics* (literally, what comes after the book on physics). The development of technical terminology in any field can be regarded as a form of knowledge representation.

Early discussions of representation in AI tended to focus on “*problem* representation” rather than “*knowledge* representation.” (See, for example, Amarel's (1968) discussion of the “Missionaries and Cannibals” problem.) In the 1970s, AI emphasized the development of “expert systems” (also called “knowledge-based systems”) that could, if given the appropriate domain knowledge, match or exceed the performance of human experts on narrowly defined tasks. For example, the first expert system, DENDRAL (Feigenbaum *et al.*, 1971; Lindsay *et al.*, 1980), interpreted the output of a mass spectrometer (a type of instrument used to analyze the structure of organic chemical compounds) as accurately as expert chemists. Although the success of DENDRAL was instrumental in convincing the AI research community of the importance of knowledge representation, the representational formalisms used in DENDRAL are highly specific to the domain of chemistry.

Over time, researchers became interested in standardized knowledge representation formalisms and ontologies that could assist in the creation of new expert systems. This brought them into territory previously explored by philosophers of science and of language. The discipline imposed in AI by the need for one's theories to “work” has led to more rapid and deeper progress than when these problems were the exclusive domain of philosophy (although it has at times also led to the repeated reinvention of the wheel).

But to what extent can we trust expert knowledge? As far back as 1955, Paul Meehl (see also Grove and Meehl, 1996) studied the decision-making processes of trained experts at subjective tasks such as predicting the success of a student in a training program or the recidivism of a criminal. In 19 out of the 20 studies he looked at, Meehl found that simple statistical learning algorithms (such as linear regression or naive Bayes) predict better than the experts. Tetlock (2017) also studies expert knowledge and finds it lacking in difficult cases. The Educational Testing Service has used an automated program to grade millions of essay questions on the GMAT exam since 1999. The program agrees with human graders 97% of the time, about the same level that two human graders agree (Burstein *et al.*, 2001). (This does not mean the program understands essays, just that it can distinguish good ones from bad ones about as well as human graders can.)

The creation of comprehensive taxonomies or classifications dates back to ancient times. Aristotle (384–322 BCE) strongly emphasized classification and categorization schemes. His *Organon*, a collection of works on logic assembled by his students after his death, included a treatise called *Categories* in which he attempted to construct what we would now call an upper ontology. He also introduced the notions of **genus** and **species** for lower-level classification. Our present system of biological classification, including the use of “binomial nomenclature” (classification via genus and species in the technical sense), was invented by the Swedish biologist Carolus Linnaeus, or Carl von Linne (1707–1778). The problems associated with natural kinds and inexact category boundaries have been addressed by Wittgenstein (1953), Quine (1953), Lakoff (1987), and Schwartz (1977), among others.

See Chapter 25 for a discussion of deep neural network representations of words and concepts that escape some of the problems of a strict ontology, but also sacrifice some of the precision. We still don’t know the best way to combine the advantages of neural networks and logical semantics for representation.

Interest in larger-scale ontologies is increasing, as documented by the *Handbook on Ontologies* (Staab, 2004). The OPENCYC project (Lenat and Guha, 1990; Matuszek *et al.*, 2006) has released a 150,000-concept ontology, with an upper ontology similar to the one in Figure 10.1 as well as specific concepts like “OLED Display” and “iPhone,” which is a type of “cellular phone,” which in turn is a type of “consumer electronics,” “phone,” “wireless communication device,” and other concepts. The NEXTKB project extends CYC and other resources including FrameNet and WordNet into a knowledge base with almost 3 million facts, and provides a reasoning engine, FIRE to go with it (Forbus *et al.*, 2010).

The DBPEDIA project extracts structured data from Wikipedia, specifically from Infoboxes: the attribute/value pairs that accompany many Wikipedia articles (Wu and Weld, 2008; Bizer *et al.*, 2007). As of 2015, DBPEDIA contained 400 million facts about 4 million objects in the English version alone; counting all 110 languages yields 1.5 billion facts (Lehmann *et al.*, 2015).

The IEEE working group P1600.1 created SUMO, the Suggested Upper Merged Ontology (Niles and Pease, 2001; Pease and Niles, 2002), with about 1000 terms in the upper ontology and links to over 20,000 domain-specific terms. Stoffel *et al.* (1997) describe algorithms for efficiently managing a very large ontology. A survey of techniques for extracting knowledge from Web pages is given by Etzioni *et al.* (2008).

On the Web, representation languages are emerging. RDF (Brickley and Guha, 2004) allows for assertions to be made in the form of relational triples and provides some means for evolving the meaning of names over time. OWL (Smith *et al.*, 2004) is a description logic that supports inferences over these triples. So far, usage seems to be inversely proportional to representational complexity: the traditional HTML and CSS formats account for over 99% of Web content, followed by the simplest representation schemes, such as RDFa (Adida and Birbeck, 2008), and microformats (Khare, 2006; Patel-Schneider, 2014) which use HTML and XHTML markup to add attributes to text on web pages. Usage of sophisticated RDF and OWL ontologies is not yet widespread, and the full vision of the Semantic Web (Berners-Lee *et al.*, 2001) has not been realized. The conferences on *Formal Ontology in Information Systems* (FOIS) covers both general and domain-specific ontologies.

The taxonomy used in this chapter was developed by the authors and is based in part on their experience in the CYC project and in part on work by Hwang and Schubert (1993)

and Davis (1990, 2005). An inspirational discussion of the general project of commonsense knowledge representation appears in Hayes's (1978, 1985b) "Naive Physics Manifesto."

Successful deep ontologies within a specific field include the Gene Ontology project (Gene Ontology Consortium, 2008) and the Chemical Markup Language (Murray-Rust *et al.*, 2003). Doubts about the feasibility of a single ontology for *all* knowledge are expressed by Doctorow (2001), Gruber (2004), Halevy *et al.* (2009), and Smith (2004).

The event calculus was introduced by Kowalski and Sergot (1986) to handle continuous time, and there have been several variations (Sadri and Kowalski, 1995; Shanahan, 1997) and overviews (Shanahan, 1999; Mueller, 2006). James Allen introduced time intervals for the same reason (Allen, 1984), arguing that intervals were much more natural than situations for reasoning about extended and concurrent events. In van Lambalgen and Hamm (2005) we see how the logic of events maps onto the language we use to talk about events. An alternative to the event and situation calculi is the fluent calculus (Thielscher, 1999), which reifies the facts out of which states are composed.

Peter Ladkin (1986a, 1986b) introduced "concave" time intervals (intervals with gaps—essentially, unions of ordinary "convex" time intervals) and applied the techniques of mathematical abstract algebra to time representation. Allen (1991) systematically investigates the wide variety of techniques available for time representation; van Beek and Manchak (1996) analyze algorithms for temporal reasoning. There are significant commonalities between the event-based ontology given in this chapter and an analysis of events due to the philosopher Donald Davidson (1980). The **histories** in Pat Hayes's (1985a) ontology of liquids and the **chronicles** in McDermott's (1985) theory of plans were also important influences on the field and on this chapter.

The question of the ontological status of substances has a long history. Plato proposed that substances were abstract entities entirely distinct from physical objects; he would say *MadeOf(Butter₃, Butter)* rather than $Butter_3 \in Butter$. This leads to a substance hierarchy in which, for example, *UnsaltedButter* is a more specific substance than *Butter*. The position adopted in this chapter, in which substances are categories of objects, was championed by Richard Montague (1973). It has also been adopted in the CYC project. Copeland (1993) mounts a serious, but not invincible, attack.

The alternative approach mentioned in the chapter, in which butter is one object consisting of all buttery objects in the universe, was proposed originally by the Polish logician Leśniewski (1916). His **mereology** (the name is derived from the Greek word for "part") used the part-whole relation as a substitute for mathematical set theory, with the aim of eliminating abstract entities such as sets. A more readable exposition of these ideas is given by Leonard and Goodman (1940), and Goodman's *The Structure of Appearance* (1977) applies the ideas to various problems in knowledge representation.

While some aspects of the mereological approach are awkward—for example, the need for a separate inheritance mechanism based on part-whole relations—the approach gained the support of Quine (1960). Harry Bunt (1985) has provided an extensive analysis of its use in knowledge representation. Casati and Varzi (1999) cover parts, wholes, and a general theory of spatial locations.

There are three main approaches to the study of mental objects. The one taken in this chapter, based on modal logic and possible worlds, is the classical approach from philosophy (Hintikka, 1962; Kripke, 1963; Hughes and Cresswell, 1996). The book *Reasoning about*

Knowledge (Fagin *et al.*, 1995) provides a thorough introduction, and Gordon and Hobbs (2017) provide *A Formal Theory of Commonsense Psychology*.

The second approach is a first-order theory in which mental objects are fluents. Davis (2005) and Davis and Morgenstern (2005) describe this approach. It relies on the possible-worlds formalism, and builds on work by Robert Moore (1980, 1985).

The third approach is a **syntactic theory**, in which mental objects are represented by character strings. A string is just a complex term denoting a list of symbols, so *CanFly(Clark)* can be represented by the list of symbols [*C, a, n, F, l, y, (, C, l, a, r, k,)*]. The syntactic theory of mental objects was first studied in depth by Kaplan and Montague (1960), who showed that it led to paradoxes if not handled carefully. Ernie Davis (1990) provides an excellent comparison of the syntactic and modal theories of knowledge. Pnueli (1977) describes a temporal logic used to reason about programs, work that won him the Turing Award and which was expanded upon by Vardi (1996). Littman *et al.* (2017) show that a temporal logic can be a good language for specifying goals to a reinforcement learning robot in a way that is easy for a human to specify, and generalizes well to different environments.

The Greek philosopher Porphyry (c. 234–305 CE), commenting on Aristotle's *Categories*, drew what might qualify as the first semantic network. Charles S. Peirce (1909) developed existential graphs as the first semantic network formalism using modern logic. Ross Quillian (1961), driven by an interest in human memory and language processing, initiated work on semantic networks within AI. An influential paper by Marvin Minsky (1975) presented a version of semantic networks called **frames**; a frame was a representation of an object or category, with attributes and relations to other objects or categories.

The question of semantics arose quite acutely with respect to Quillian's semantic networks (and those of others who followed his approach), with their ubiquitous and very vague "IS-A links." Bill Woods's (1975) famous article "What's In a Link?" drew the attention of AI researchers to the need for precise semantics in knowledge representation formalisms. Ron Brachman (1979) elaborated on this point and proposed solutions. Patrick Hayes's (1979) "The Logic of Frames" cut even deeper, claiming that "Most of 'frames' is just a new syntax for parts of first-order logic." Drew McDermott's (1978b) "Tarskian Semantics, or, No Notation without Denotation!" argued that the model-theoretic approach to semantics used in first-order logic should be applied to all knowledge representation formalisms. This remains a controversial idea; notably, McDermott himself has reversed his position in "A Critique of Pure Reason" (McDermott, 1987). Selman and Levesque (1993) discuss the complexity of inheritance with exceptions, showing that in most formulations it is NP-complete.

Description logics were developed as a useful subset of first-order logic for which inference is computationally tractable. Hector Levesque and Ron Brachman (1987) showed that certain uses of disjunction and negation were primarily responsible for the intractability of logical inference. This led to a better understanding of the interaction between complexity and expressiveness in reasoning systems. Calvanese *et al.* (1999) summarize the state of the art, and Baader *et al.* (2007) present a comprehensive handbook of description logic.

The three main formalisms for dealing with nonmonotonic inference—circumscription (McCarthy, 1980), default logic (Reiter, 1980), and modal nonmonotonic logic (McDermott and Doyle, 1980)—were all introduced in one special issue of the *AI Journal*. Delgrande and Schaub (2003) discuss the merits of the variants, given 25 years of hindsight. Answer set programming can be seen as an extension of negation as failure or as a refinement of

circumscription; the underlying theory of stable model semantics was introduced by Gelfond and Lifschitz (1988), and the leading answer set programming systems are DLV (Eiter *et al.*, 1998) and SMODELS (Niemelä *et al.*, 2000). Brewka *et al.* (1997) give a good overview of the various approaches to nonmonotonic logic. Clark (1978) covers the negation-as-failure approach to logic programming and Clark completion. Lifschitz (2001) discusses the application of answer set programming to planning. A variety of nonmonotonic reasoning systems based on logic programming are documented in the proceedings of the conferences on *Logic Programming and Nonmonotonic Reasoning* (LPNMR).

The study of truth maintenance systems began with the TMS (Doyle, 1979) and RUP (McAllester, 1980) systems, both of which were essentially JTMSs. Forbus and de Kleer (1993) explain in depth how TMSs can be used in AI applications. Nayak and Williams (1997) show how an efficient incremental TMS called an ITMS makes it feasible to plan the operations of a NASA spacecraft in real time.

This chapter could not cover *every* area of knowledge representation in depth. The three principal topics omitted are the following:

Qualitative physics

Qualitative physics: Qualitative physics is a subfield of knowledge representation concerned specifically with constructing a logical, nonnumeric theory of physical objects and processes. The term was coined by Johan de Kleer (1975), although the enterprise could be said to have started in Fahlman's (1974) BUILD, a sophisticated planner for constructing complex towers of blocks. Fahlman discovered in the process of designing it that most of the effort (80%, by his estimate) went into modeling the physics of the blocks world to calculate the stability of various subassemblies of blocks, rather than into planning per se. He sketches a hypothetical naive-physics-like process to explain why young children can solve BUILD-like problems without access to the high-speed floating-point arithmetic used in BUILD's physical modeling. Hayes (1985a) uses "histories"—four-dimensional slices of space-time similar to Davidson's events—to construct a fairly complex naive physics of liquids. Davis (2008) gives an update to the ontology of liquids that describes the pouring of liquids into containers.

De Kleer and Brown (1985), Ken Forbus (1985), and Benjamin Kuipers (1985) independently and almost simultaneously developed systems that can reason about a physical system based on qualitative abstractions of the underlying equations. Qualitative physics soon developed to the point where it became possible to analyze an impressive variety of complex physical systems (Yip, 1991). Qualitative techniques have been used to construct novel designs for clocks, windshield wipers, and six-legged walkers (Subramanian and Wang, 1994). The collection *Readings in Qualitative Reasoning about Physical Systems* (Weld and de Kleer, 1990), an encyclopedia article by Kuipers (2001), and a handbook article by Davis (2007) provide good introductions to the field.

Spatial reasoning

Spatial reasoning: The reasoning necessary to navigate in the wumpus world is trivial in comparison to the rich spatial structure of the real world. The earliest serious attempt to capture commonsense reasoning about space appears in the work of Ernest Davis (1986, 1990). The region connection calculus of Cohn *et al.* (1997) supports a form of qualitative spatial reasoning and has led to new kinds of geographical information systems; see also (Davis, 2006). As with qualitative physics, an agent can go a long way, so to speak, without resorting to a full metric representation.

Psychological reasoning: Psychological reasoning involves the development of a working *psychology* for artificial agents to use in reasoning about themselves and other agents. This is often based on so-called folk psychology, the theory that humans in general are believed to use in reasoning about themselves and other humans. When AI researchers provide their artificial agents with psychological theories for reasoning about other agents, the theories are frequently based on the researchers' description of the logical agents' own design. Psychological reasoning is currently most useful within the context of natural language understanding, where divining the speaker's intentions is of paramount importance.

Minker (2001) collects papers by leading researchers in knowledge representation, summarizing 40 years of work in the field. The proceedings of the international conferences on *Principles of Knowledge Representation and Reasoning* provide the most up-to-date sources for work in this area. *Readings in Knowledge Representation* (Brachman and Levesque, 1985) and *Formal Theories of the Commonsense World* (Hobbs and Moore, 1985) are excellent anthologies on knowledge representation; the former focuses more on historically important papers in representation languages and formalisms, the latter on the accumulation of the knowledge itself. Davis (1990), Stefik (1995), and Sowa (1999) provide textbook introductions to knowledge representation, van Harmelen *et al.* (2007) contributes a handbook, and Davis and Morgenstern (2004) edited a special issue of the AI Journal on the topic. Davis (2017) gives a survey of logic for commonsense reasoning. The biennial conference on *Theoretical Aspects of Reasoning About Knowledge* (TARK) covers applications of the theory of knowledge in AI, economics, and distributed systems.