I can fit on the tray table of an economy-class airline seat.” It is impossible to enumerate in advance all the ways a category can be named, so the agent will have to be able to do additional reasoning in some cases to determine if the Name relation holds. In the worst case, this requires full natural language understanding, a topic that we will defer to Chapter 22. In practice, a few simple rules—such as allowing ‘laptop’ to match a category named ‘laptops’—go a long way. Exercise 12.10 asks you to develop a set of such rules after doing some research into online stores.

Given the logical definitions from the preceding paragraphs and suitable knowledge bases of product categories and naming conventions, are we ready to apply an inference algorithm to obtain a set of relevant offers for our query? Not quite! The missing element is the Contents (uri) function, which refers to the HTML page at a given URL. The agent doesn’t have the page contents of every URL in its knowledge base; nor does it have explicit rules for deducing what those contents might be. Instead, we can arrange for the right HTTP procedure to be executed whenever a subgoal involves the Contents function. In this way, it appears to the inference engine as if the entire Web is inside the knowledge base. This is an example of a general technique called procedural attachment, whereby particular predicates and functions can be handled by special-purpose methods.

12.7.2 Comparing offers

Let us assume that the reasoning processes of the preceding section have produced a set of offer pages for our ‘laptops’ query. To compare those offers, the agent must extract the relevant information—price, speed, disk size, weight, and so on—from the offer pages. This can be a difficult task with real Web pages, for all the reasons mentioned previously. A common way of dealing with this problem is to use programs called wrappers to extract information from a page. The technology of information extraction is discussed in Section 22.4. For now we assume that wrappers exist, and when given a page and a knowledge base, they add assertions to the knowledge base. Typically, a hierarchy of wrappers would be applied to a page: a very general one to extract dates and prices, a more specific one to extract attributes for computer-related products, and if necessary a site-specific one that knows the format of a particular store. Given a page on the example .com site with the text

IBM ThinkBook 970. Our price: $399.00

followed by various technical specifications, we would like a wrapper to extract information such as the following:

\[
\exists c, \text{offer } e. \text{LaptopComputers }A \text{offer } e \text{ProductOffers }A \\
\text{Manufacture*, IBM }A \text{Model(c, ThinkBook970) }A \\
\text{ScreenSize(c, Inches (14)) }A \text{ScreenType(c, ColorLCD) }A \\
\text{MemorySize(c, Gigabytes (2)) }A \text{CPUSpeed(c, GHz (1.2)) }A \\
\text{OfferedProduct(offer, c)} A \text{Store(offer, GenStore) }A \\
\text{URL(offer, "example.com/computers/34356.html") }A \\
\text{Price(offer, $(399)) }A \text{Date(offer, Today).}
\]

This example illustrates several issues that arise when we take seriously the task of knowledge engineering for commercial transactions. For example, notice that the price is an attribute of
the offer, not the product itself. This is important because the offer at a given store may change from day to day even for the same individual laptop; for some categories—such as houses and paintings—the same individual object may even be offered simultaneously by different intermediaries at different prices. There are still more complications that we have not handled, such as the possibility that the price depends on the method of payment and on the buyer’s qualifications for certain discounts. The final task is to compare the offers that have been extracted. For example, consider these three offers:

A: 1.4 GHz CPU, 2GB RAM, 250 GB disk, $299.
B: 1.2 GHz CPU, 4GB RAM, 350 GB disk, $300.
C: 1.2 GHz CPU, 2GB RAM, 250 GB disk, $399.

C is dominated by A; that is, A is cheaper and faster, and they are otherwise the same. In general, X dominates Y if X has a better value on at least one attribute, and is not worse on any attribute. But neither A nor B dominates the other. To decide which is better we need to know how the buyer weighs CPU speed and price against memory and disk space. The general topic of preferences among multiple attributes is addressed in Section 16.4; for now, our shopping agent will simply return a list of all undominated offers that meet the buyer’s description. In this example, both A and B are undominated. Notice that this outcome relies on the assumption that everyone prefers cheaper prices, faster processors, and more storage. Some attributes, such as screen size on a notebook, depend on the user’s particular preference (portability versus visibility); for these, the shopping agent will just have to ask the user.

The shopping agent we have described here is a simple one; many refinements are possible. Still, it has enough capability that with the right domain-specific knowledge it can actually be of use to a shopper. Because of its declarative construction, it extends easily to more complex applications. The main point of this section is to show that some knowledge representation—in particular, the product hierarchy—is necessary for such an agent, and that once we have some knowledge in this form, the rest follows naturally.

12.8 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.