Given the percept and rules from the preceding paragraphs, this would yield the desired conclusion

\[ \text{BestAction} (\text{Grab}, 5) \rightarrow \text{Grab} \text{ is the right thing to do.} \]

We have represented the agent's inputs and outputs; now it is time to represent the environment itself. Let us begin with objects. Obvious candidates are squares, pits, and the wumpus. We could name each square—Square1,2 and so on—but then the fact that Square1,2 and Square1,3 are adjacent would have to be an "extra" fact, and we would need one such fact for each pair of squares. It is better to use a complex term in which the row and column appear as integers; for example, we can simply use the list term \([1, 2]\). Adjacency of any two squares can be defined as

\[ \forall x, y, a, b \: \text{Adjacent}([x, y], [a, b]) \iff (x = a \land (y = b - 1 \lor y = b - 1)) \lor (y = b \land (x = a - 1 \lor x = a + 1)). \]

We could name each pit, but this would be inappropriate for a different reason: there is no reason to distinguish among pits. It is simpler to use a unary predicate \(\text{Pit}\) that is true of squares containing pits. Finally, since there is exactly one wumpus, a constant \(\text{Wumpus}\) is just as good as a unary predicate (and perhaps more dignified from the wumpus's viewpoint).

The agent's location changes over time, so we write \(\forall t \: \text{At}(\text{Agent}, s, t)\) to mean that the agent is at square \(s\) at time \(t\). We can fix the wumpus's location with \(\forall t \: \text{At}(\text{Wumpus}, [2, 2], t)\).

We can then say that objects can only be at one location at a time:

\[ \forall s_1, s_2, t \: \text{At}(x, s_1, t) \land \text{At}(x, s_2, t) = s_1 = s_2. \]

Given its current location, the agent can infer properties of the square from properties of its current percept. For example, if the agent is at a square and perceives a breeze, then that square is breezy:

\[ \forall s, t \: \text{At}(\text{Agent}, s, t) \land \text{Breeze}(t) = \text{Breezy}(s). \]

It is useful to know that a square is breezy because we know that the pits cannot move about. Notice that \(\text{Breezy}\) has no time argument.

Having discovered which places are breezy (or smelly) and, very important, not breezy (or not smelly), the agent can deduce where the pits are (and where the wumpus is). Whereas propositional logic necessitates a separate axiom for each square (see R2 and R3 on page 247) and would need a different set of axioms for each geographical layout of the world, first-order logic just needs one axiom:

\[ \forall s \: \text{Breezy}(s) \iff \exists r \: \text{Adjacent}(x, s) \land \text{Pit}(r) \tag{8.4} \]

Similarly, in first-order logic we can quantify over time, so we need just one successor-state axiom for each predicate, rather than a different copy for each time step. For example, the axiom for the arrow (Equation (7.2) on page 267) becomes

\[ \forall t \: \text{HaveArrow}(t + 1) \iff (\text{HaveArrow}(t) \land \neg \text{Action}(\text{Shoot}, t)). \]

From these two example sentences, we can see that the first-order logic formulation is no less concise than the original English-language description given in Chapter 7. The reader

Similarly, most of us do not name each bird that flies overhead as it migrates to warmer regions in winter. An ornithologist wishing to study migration patterns, survival rates, and so on does name each bird, by means of a ring on its leg, because individual birds must be tracked.
is invited to construct analogous axioms for the agent's location and orientation; in these cases, the axioms quantify over both space and time. As in the case of propositional state estimation, an agent can use logical inference with axioms of this kind to keep track of aspects of the world that are not directly observed. Chapter 10 goes into more depth on the subject of first-order successor-state axioms and their uses for constructing plans.

8.4 KNOWLEDGE ENGINEERING IN FIRST-ORDER LOGIC

The preceding section illustrated the use of first-order logic to represent knowledge in three simple domains. This section describes the general process of knowledge-base construction—a process called knowledge engineering. A knowledge engineer is someone who investigates a particular domain, learns what concepts are important in that domain, and creates a formal representation of the objects and relations in the domain. We illustrate the knowledge engineering process in an electronic circuit domain that should already be fairly familiar, so that we can concentrate on the representational issues involved. The approach we take is suitable for developing special-purpose knowledge bases whose domain is carefully circumscribed and whose range of queries is known in advance. General-purpose knowledge bases, which cover a broad range of human knowledge and are intended to support tasks such as natural language understanding, are discussed in Chapter 12.

8.4.1 The knowledge-engineering process

Knowledge engineering projects vary widely in content, scope, and difficulty, but all such projects include the following steps:

1. **Identify the task.** The knowledge engineer must delineate the range of questions that the knowledge base will support and the kinds of facts that will be available for each specific problem instance. For example does the wumpus knowledge base need to be able to choose actions or is it required to answer questions only about the contents of the environment? Will the sensor facts include the current location? The task will determine what knowledge must be represented in order to connect problem instances to answers. This step is analogous to the PEAS process for designing agents in Chapter 2.

2. **Assemble the relevant knowledge.** The knowledge engineer might already be an expert in the domain, or might need to work with real experts to extract what they know—a process called knowledge acquisition. At this stage, the knowledge is not represented formally. The idea is to understand the scope of the knowledge base, as determined by the task, and to understand how the domain actually works.

For the wumpus world, which is defined by an artificial set of rules, the relevant knowledge is easy to identify. (Notice, however, that the definition of adjacency was not supplied explicitly in the wumpus-world rules.) For real domains, the issue of relevance can be quite difficult—for example, a system for simulating VLSI designs might or might not need to take into account stray capacitances and skin effects.