COUNT NOUNS
MASS NOUNS

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. We will describe just one; the others are covered in the historical notes section.

To represent stuff properly, we begin with the obvious. We will 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 same butter that was 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 will 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:

$$x \in Butter \wedge PartOf(y, x) \Rightarrow y \in Butter\ .$$

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

$$x \in Butter \Rightarrow MeltingPoint(x, Centigrade(30))\ .$$

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. On the other hand, the category $PoundOfButter$, which includes as members all butter-objects weighing one pound, is not a substance! If we cut a pound of butter in half, we do not, alas, get two pounds of butter.

INTRINSIC
EXTRINSIC

What is actually going on is this: there are some properties that are **intrinsic**: they belong to the very substance of the object, rather than to the object as a whole. When you cut a substance in half, the two pieces retain the same set of intrinsic properties—things like density, boiling point, flavor, color, ownership, and so on. On the other hand, **extrinsic** properties are the opposite: properties such as weight, length, shape, function, and so on are not retained under subdivision.

A class 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. The category *Stuff* is the most general substance category, specifying no intrinsic properties. The category *Thing* is the most general discrete object category, specifying no extrinsic properties. All physical objects belong to both categories, so the categories are coextensive—they refer to the same entities.

10.3 Actions, Situations, and Events

Reasoning about the results of actions is central to the operation of a knowledge-based agent. Chapter 7 gave examples of propositional sentences describing how actions affect the wumpus world—for example, Equation (7.3) on page 227 states how the agent's location is changed by forward motion. One drawback of propositional logic is the need to have a different copy of the action description for each time at which the action might be executed. This section describes a representation method that uses first-order logic to avoid that problem.

The ontology of situation calculus

One obvious way to avoid multiple copies of axioms is simply to quantify over time—to say, "$\forall t$, such-and-such is the result at $t+1$ of doing the action at t." Instead of dealing with explicit times like $t+1$, we will concentrate in this section on *situations*, which denote the states resulting from executing actions. This approach is called **situation calculus** and involves the following ontology:

SITUATION CALCULUS

- As in Chapter 8, actions are logical terms such as $Forward$ and $Turn(Right)$. For now, we will assume that the environment contains only one agent. (If there is more than one, an additional argument can be inserted to say which agent is doing the action.)

SITUATIONS

- **Situations** are logical terms consisting of the initial situation (usually called S_0) and all situations that are generated by applying an action to a situation. The function $Result(a, s)$ (sometimes called Do) names the situation that results when action a is executed in situation s. Figure 10.2 illustrates this idea.

FLUENTS

- **Fluents** are functions and predicates that vary from one situation to the next, such as the location of the agent or the aliveness of the wumpus. The dictionary says a fluent is something that flows, like a liquid. In this use, it means flowing or changing across situations. By convention, the situation is always the last argument of a fluent. For example, $\neg Holding(G_1, S_0)$ says that the agent is not holding the gold G_1 in the initial situation S_0. $Age(Wumpus, S_0)$ refers to the wumpus's age in S_0.
- **Atemporal** or **eternal** predicates and functions are also allowed. Examples include the predicate $Gold(G_1)$ and the function $LeftLegOf(Wumpus)$.

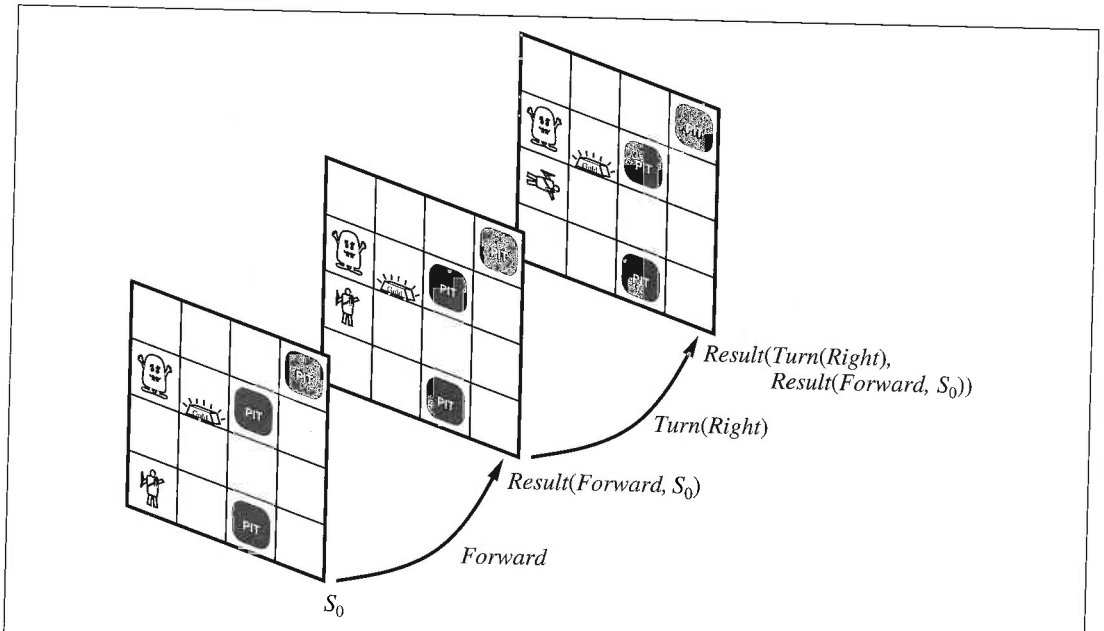

Figure 10.2 In situation calculus, each situation (except S_0) is the result of an action.

In addition to single actions, it is also helpful to reason about action sequences. We can define the results of sequences in terms of the results of individual actions. First, we say that executing an empty sequence leaves the situation unchanged:

$$Result([\,], s) = s\ .$$

Executing a nonempty sequence is the same as executing the first action and then executing the rest in the resulting situation:

$$Result([a|seq], s) = Result(seq, Result(a, s))\ .$$

PROJECTION
PLANNING

A situation calculus agent should be able to deduce the outcome of a given sequence of actions; this is the **projection** task. With a suitable constructive inference algorithm, it should also be able to *find* a sequence that achieves a desired effect; this is the **planning** task.

We will use an example from a modified version of the wumpus world where we do not worry about the agent's orientation and where the agent can Go from one location to an adjacent one. Suppose the agent is at $[1,1]$ and the gold is at $[1,2]$. The aim is to have the gold in $[1,1]$. The fluent predicates are $At(o,x,s)$ and $Holding(o,s)$. Then the initial knowledge base might include the following description:

$$At(Agent, [1,1], S_0) \wedge At(G_1, [1,2], S_0)\ .$$

This is not quite enough, however, because it doesn't say what *isn't* true in S_0. (See page 355 for further discussion of this point.) The complete description is as follows:

$$At(o,x,S_0) \Leftrightarrow [(o = Agent \wedge x = [1,1]) \vee (o = G_1 \wedge x = [1,2])]\ .$$
$$\neg Holding(o, S_0)\ .$$

We also need to state that G_1 is gold and that $[1,1]$ and $[1,2]$ are adjacent:

$$Gold(G_1) \wedge Adjacent([1,1],[1,2]) \wedge Adjacent([1,2],[1,1])\ .$$

One would like to be able to prove that the agent achieves its aim by going to $[1,2]$, grabbing the gold, and returning to $[1,1]$. That is,

$$At(G_1, [1,1], Result([Go([1,1],[1,2]), Grab(G_1), Go([1,2],[1,1])], S_0))\ .$$

More interesting is the possibility of constructing a plan to get the gold, which is achieved by answering the query "what sequence of actions results in the gold being at [1,1]?"

$$\exists\, seq\ \ At(G_1, [1,1], Result(seq, S_0))\ .$$

Let us see what has to go into the knowledge base for these queries to be answered.

Describing actions in situation calculus

POSSIBILITY AXIOM
EFFECT AXIOM

In the simplest version of situation calculus, each action is described by two axioms: a **possibility axiom** that says when it is possible to execute the action, and an **effect axiom** that says what happens when a possible action is executed. We will use $Poss(a,s)$ to mean that it is possible to execute action a in situation s. The axioms have the following form:

POSSIBILITY AXIOM: $Preconditions \Rightarrow Poss(a,s)$.
EFFECT AXIOM: $Poss(a,s) \Rightarrow$ *Changes that result from taking action.*

We will present these axioms for the modified wumpus world. To shorten our sentences, we will omit universal quantifiers whose scope is the entire sentence. We assume that the variable s ranges over situations, a ranges over actions, o over objects (including agents), g over gold, and x and y over locations.

The possibility axioms for this world state that an agent can go between adjacent locations, grab a piece of gold in the current location, and release some gold that it is holding:

$$\begin{aligned} &At(Agent, x, s) \wedge Adjacent(x, y) && \Rightarrow \quad Poss(Go(x, y), s) . \\ &Gold(g) \wedge At(Agent, x, s) \wedge At(g, x, s) && \Rightarrow \quad Poss(Grab(g), s) . \\ &Holding(g, s) && \Rightarrow \quad Poss(Release(g), s) . \end{aligned}$$

The effect axioms state that, if an action is possible, then certain properties (fluents) will hold in the situation that results from executing the action. Going from x to y results in being at y, grabbing the gold results in holding the gold, and releasing the gold results in not holding it:

$$\begin{aligned} &Poss(Go(x, y), s) && \Rightarrow \quad At(Agent, y, Result(Go(x, y), s)) . \\ &Poss(Grab(g), s) && \Rightarrow \quad Holding(g, Result(Grab(g), s)) . \\ &Poss(Release(g), s) && \Rightarrow \quad \neg Holding(g, Result(Release(g), s)) . \end{aligned}$$

Having stated these axioms, can we prove that our little plan achieves the goal? Unfortunately not! At first, everything works fine; $Go([1, 1], [1, 2])$ is indeed possible in S_0 and the effect axiom for Go allows us to conclude that the agent reaches [1,2]:

$$At(Agent, [1, 2], Result(Go([1, 1], [1, 2]), S_0)) .$$

Now we consider the $Grab(G_1)$ action. We have to show that it is possible in the new situation—that is,

$$At(G_1, [1, 2], Result(Go([1, 1], [1, 2]), S_0)) .$$

Alas, nothing in the knowledge base justifies such a conclusion. Intuitively, we understand that the agent's Go action should have no effect on the gold's location, so it should still be at $[1, 2]$, where it was in S_0. *The problem is that the effect axioms say what changes, but don't say what stays the same.*

FRAME PROBLEM

Representing all the things that stay the same is called the **frame problem**. We must find an efficient solution to the frame problem because, in the real world, almost everything stays the same almost all the time. Each action affects only a tiny fraction of all fluents.

FRAME AXIOM

One approach is to write explicit **frame axioms** that *do* say what stays the same. For example, the agent's movements leave other objects stationary unless they are held:

$$At(o, x, s) \wedge (o \neq Agent) \wedge \neg Holding(o, s) \Rightarrow At(o, x, Result(Go(y, z), s)) .$$

If there are F fluent predicates and A actions, then we will need $O(AF)$ frame axioms. On the other hand, if each action has at most E effects, where E is typically much less than F, then we should be able to represent what happens with a much smaller knowledge base of size $O(AE)$. This is the **representational frame problem**. The closely related **inferential frame problem** is to project the results of a t-step sequence of actions in time $O(Et)$, rather than time $O(Ft)$ or $O(AEt)$. We will address each problem in turn. Even then, another problem remains—that of ensuring that *all* necessary conditions for an action's success have been specified. For example, Go fails if the agent dies *en route*. This is the **qualification problem**, for which there is no complete solution.

REPRESENTATIONAL FRAME PROBLEM

INFERENTIAL FRAME PROBLEM

QUALIFICATION PROBLEM

Solving the representational frame problem

The solution to the representational frame problem involves just a slight change in viewpoint on how to write the axioms. Instead of writing out the effects of each action, we consider how each fluent predicate evolves over time.[3] The axioms we use are called **successor-state axioms**. They have the following form:

SUCCESSOR-STATE AXIOM

SUCCESSOR-STATE AXIOM:
Action is possible $\Rightarrow$
(*Fluent is true in result state* $\Leftrightarrow$ *Action's effect made it true*
$\vee$ *It was true before and action left it alone*) .

After the qualification that we are not considering impossible actions, notice that this definition uses $\Leftrightarrow$, not $\Rightarrow$. This means that the axiom says that the fluent will be true if *and only if* the right-hand side holds. Put another way, we are specifying the truth value of each fluent in the next state as a function of the action and the truth value in the current state. This means that the next state is completely specified from the current state and hence that there are no additional frame axioms needed.

The successor-state axiom for the agent's location says that the agent is at y after executing an action either if the action is possible and consists of moving to y or if the agent was already at y and the action is not a move to somewhere else:

$$Poss(a, s) \Rightarrow$$
$$(At(Agent, y, Result(a, s)) \Leftrightarrow a = Go(x, y) \vee (At(Agent, y, s) \wedge a \neq Go(y, z))) .$$

The axiom for *Holding* says that the agent is holding g after executing an action if the action was a grab of g and the grab is possible or if the agent was already holding g, and the action is not releasing it:

$$Poss(a, s) \Rightarrow$$
$$(Holding(g, Result(a, s)) \Leftrightarrow a = Grab(g) \vee (Holding(g, s) \wedge a \neq Release(g))) .$$

Successor-state axioms solve the representational frame problem because the total size of the axioms is $O(AE)$ literals: each of the E effects of each of the A actions is mentioned exactly once. The literals are spread over F different axioms, so the axioms have average size AE/F.

The astute reader will have noticed that these axioms handle the At fluent for the agent, but not for the gold; thus, we still cannot prove that the three-step plan achieves the goal of having the gold in $[1, 1]$. We need to say that an **implicit effect** of an agent moving from x to y is that any gold it is carrying will move too (as will any ants on the gold, any bacteria on the ants, etc.). Dealing with implicit effects is called the **ramification problem**. We will discuss the problem in general later, but for this specific domain, it can be solved by writing a more general successor-state axiom for At. The new axiom, which subsumes the previous version, says that an object o is at y if the agent went to y and o is the agent or something the

IMPLICIT EFFECT

RAMIFICATION PROBLEM

[3] This is essentially the approach we took in building the Boolean circuit-based agent in Chapter 7. Indeed, axioms such as Equation (7.4) and Equation (7.5) can be viewed as successor-state axioms.

agent was holding; or if o was already at y and the agent didn't go elsewhere, with o being the agent or something the agent was holding.

$$\begin{aligned} &Poss(a, s) \Rightarrow \\ &At(o, y, Result(a, s)) \quad \Leftrightarrow \quad (a = Go(x, y) \wedge (o = Agent \vee Holding(o, s))) \\ &\qquad\qquad \vee (At(o, y, s) \wedge \neg(\exists z \;\; y \neq z \wedge a = Go(y, z) \wedge \\ &\qquad\qquad\quad (o = Agent \vee Holding(o, s)))) \,. \end{aligned}$$

There is one more technicality: an inference process that uses these axioms must be able to prove nonidentities. The simplest kind of nonidentity is between constants—for example, $Agent \neq G_1$. The general semantics of first-order logic allows distinct constants to refer to the same object, so the knowledge base must include an axiom to prevent this. The **unique names axiom** states a disequality for every pair of constants in the knowledge base. When this is assumed by the theorem prover, rather than written down in the knowledge base, it is called a **unique names assumption**. We also need to state disequalities between action terms: $Go([1, 1], [1, 2])$ is a different action from $Go([1, 2], [1, 1])$ or $Grab(G_1)$. First, we say that each type of action is distinct—that no Go action is a $Grab$ action. For each pair of action names A and B, we would have

UNIQUE NAMES AXIOM

$$A(x_1, \ldots, x_m) \neq B(y_1, \ldots, y_n) \,.$$

Next, we say that two action terms with the same action name refer to the same action only if they involve all the same objects:

$$A(x_1, \ldots, x_m) = A(y_1, \ldots, y_m) \;\Leftrightarrow\; x_1 = y_1 \wedge \ldots \wedge x_m = y_m \,.$$

These are called, collectively, the **unique action axioms**. The combination of initial state description, successor-state axioms, unique name axiom, and unique action axioms suffices to prove that the proposed plan achieves the goal.

UNIQUE ACTION AXIOMS

Solving the inferential frame problem

Successor-state axioms solve the representational frame problem, but not the inferential frame problem. Consider a t-step plan p such that $S_t = Result(p, S_0)$. To decide which fluents are true in S_t, we need to consider each of the F frame axioms on each of the t time steps. Because the axioms have average size AE/F, this gives us $O(AEt)$ inferential work. Most of the work involves copying fluents unchanged from one situation to the next.

To solve the inferential frame problem, we have two possibilities. First, we could discard situation calculus and invent a new formalism for writing axioms. This has been done with formalisms such as the **fluent calculus**. Second, we could alter the inference mechanism to handle frame axioms more efficiently. A hint that this should be possible is that the simple approach is $O(AEt)$; why should it depend on the number of actions, A, when we know exactly which one action is executed at each time step? To see how to improve matters, we first look at the format of the frame axioms:

$$\begin{aligned} &Poss(a, s) \Rightarrow \\ &F_i(Result(a, s)) \quad \Leftrightarrow \quad (a = A_1 \vee a = A_2 \ldots) \\ &\qquad\qquad\qquad\quad \vee F_i(s) \wedge (a \neq A_3) \wedge (a \neq A_4) \ldots \end{aligned}$$

That is, each axiom mentions several actions that can make the fluent true and several actions that can make it false. We can formalize this by introducing the predicate $PosEffect(a, F_i)$, meaning that action a causes F_i to become true, and $NegEffect(a, F_i)$ meaning that a causes F_i to become false. Then we can rewrite the foregoing axiom schema as:

$$Poss(a, s) \Rightarrow$$
$$F_i(Result(a, s)) \Leftrightarrow PosEffect(a, F_i) \vee [F_i(s) \wedge \neg NegEffect(a, F_i)]$$
$$PosEffect(A_1, F_i)$$
$$PosEffect(A_2, F_i)$$
$$NegEffect(A_3, F_i)$$
$$NegEffect(A_4, F_i) .$$

Whether this can be done automatically depends on the exact format of the frame axioms. To make an efficient inference procedure using axioms like this, we need to do three things:

1. Index the $PosEffect$ and $NegEffect$ predicates by their first argument so that when we are given an action that occurs at time t, we can find its effects in $O(1)$ time.
2. Index the axioms so that once you know that F_i is an effect of an action, you can find the axiom for F_i in $O(1)$ time. Then you need not even consider the axioms for fluents that are not an effect of the action.
3. Represent each situation as a previous situation plus a delta. Thus, if nothing changes from one step to the next, we need do no work at all. In the old approach, we would need to do $O(F)$ work in generating an assertion for each fluent $F_i(Result(a, s))$ from the preceding $F_i(s)$ assertions.

Thus at each time step, we look at the current action, fetch its effects, and update the set of true fluents. Each time step will have an average of E of these updates, for a total complexity of $O(Et)$. This constitutes a solution to the inferential frame problem.

Time and event calculus

Situation calculus works well when there is a single agent performing instantaneous, discrete actions. When actions have duration and can overlap with each other, situation calculus becomes somewhat awkward. Therefore, we will cover those topics with an alternative formalism known as **event calculus**, which is based on points in time rather than on situations. (The terms "event" and "action" may be used interchangeably. Informally, "event" connotes a wider class of actions, including ones with no explicit agent. These are easier to handle in event calculus than in situation calculus.)

EVENT CALCULUS

In event calculus, fluents hold at points in time rather than at situations, and the calculus is designed to allow reasoning over intervals of time. The event calculus axiom says that a fluent is true at a point in time if the fluent was initiated by an event at some time in the past and was not terminated by an intervening event. The $Initiates$ and $Terminates$ relations play a role similar to the $Result$ relation in situation calculus; $Initiates(e, f, t)$ means that the occurrence of event e at time t causes fluent f to become true, while $Terminates(w, f, t)$ means that f ceases to be true. We use $Happens(e, t)$ to mean that event e happens at time t,