THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

Chapter 2 - Intelligent Agents

CS-4365: Artificial Intelligence

Chris Irwin Davis, Ph.D.

- Chapter 1 identified the concept of **rational agents** as central to our approach to artificial intelligence.
- In this chapter, we make this notion more concrete. We will see that the concept of rationality can be applied to a wide variety of agents operating in any imaginable environment.
- Our plan in this book is to use this concept to develop a small set of design principles for building successful agents—systems that can reasonably be called **intelligent**.

- We begin by examining agents, environments, and the coupling between them.
- The observation that some agents behave better than others leads naturally to the idea of a rational agent—one that behaves as well as possible.
- How well an agent can behave depends on the nature of the environment; some environments are more difficult than others.

- We give a crude categorization of environments and show how properties of an environment influence the design of suitable agents for that environment.
- We describe a number of basic “skeleton” agent designs, which we flesh out in the rest of the book.

1.1 Agents and Environments

Agents and Environments

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. This simple idea is illustrated in **Figure 2.1**.

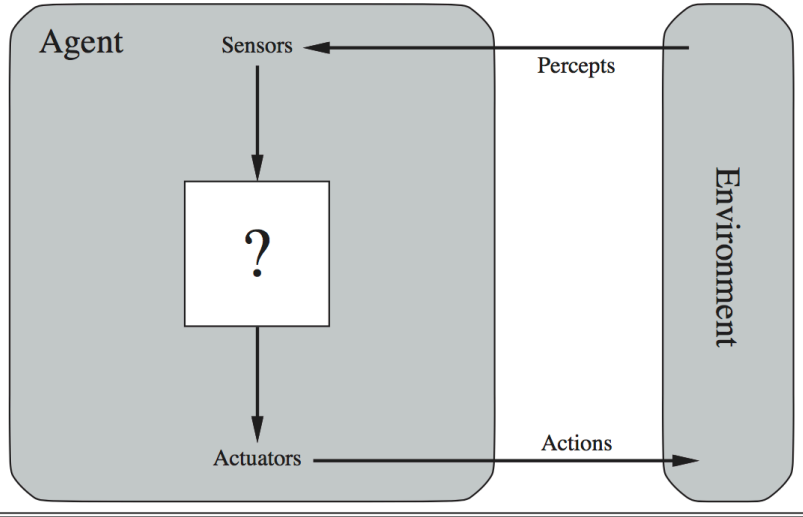

Figure 2.1 Agents interact with environments through sensors and actuators.

Agents and Environments

- A **human agent** has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators.
- A **robotic agent** might have cameras and infrared range finders for sensors and various motors for actuators.
- A **software agent** receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

Agents and Environments

- We use the term **percept** to refer to the agent's perceptual inputs at any given instant.
- An agent's **percept sequence** is the complete history of everything the agent has ever perceived.
- In general, *an agent's choice of action at any given instant can depend on the entire percept sequence observed to date, but not on anything it hasn't perceived*.

Agents and Environments

- By specifying the agent's choice of action for every possible percept sequence, we have said more or less everything there is to say about the agent.
- Mathematically speaking, we say that an agent's behavior is described by the **agent function** that maps any given *percept sequence* to an *action*.

Agents and Environments

- We can imagine tabulating the agent function that describes any given agent; for most agents, this would be a very large table—infinite, in fact, unless we place a bound on the length of percept sequences we want to consider.
- Given an agent to experiment with, we can, in principle, construct this table by trying out all possible percept sequences and recording which actions the agent does in response.

Agents and Environments

- The table is, of course, an external characterization of the agent.
- Internally, the agent function for an artificial agent will be implemented by an **agent program**.
- *It is important to keep these two ideas distinct.*
  - The *agent function* is an abstract mathematical description;
  - The *agent program* is a concrete implementation, running within some physical system.

Agents and Environments

- To illustrate these ideas, we use a very simple example—the vacuum-cleaner world shown in Figure 2.2.
- This world is so simple that we can describe everything that happens; it's also a made-up world, so we can invent many variations.

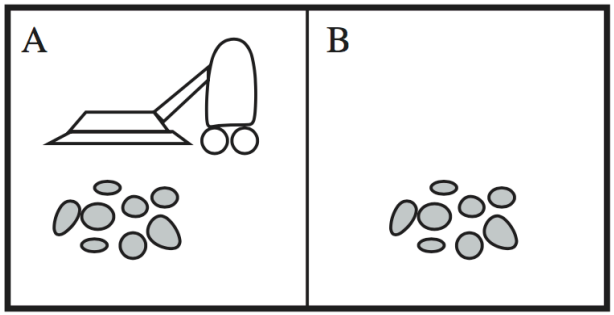

Figure 2.2 A vacuum-cleaner world with just two locations.

Agents and Environments

- This particular world has just two locations: squares A and B.
- The vacuum agent perceives which square it is in and whether there is dirt in the square.
- It can choose to move left, move right, suck up the dirt, or do nothing.

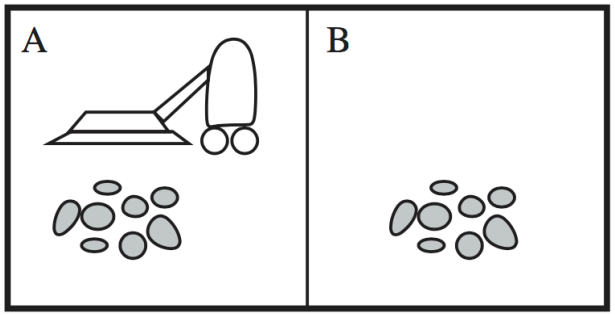

Figure 2.2 A vacuum-cleaner world with just two locations.

Agents and Environments

- One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square.
- A partial tabulation of this agent function is shown in Figure 2.3 and an agent program that implements it appears in Figure 2.8 (page 48).

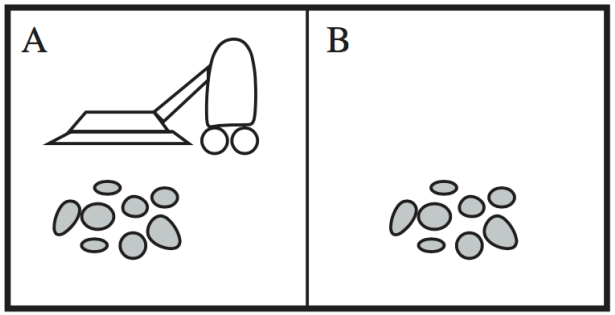

Figure 2.2 A vacuum-cleaner world with just two locations.

Agents and Environments

- Looking at Figure 2.3, we see that various vacuum-world agents can be defined simply by filling in the right-hand column in various ways.

Percept sequence	Action
$[A, Clean]$	*Right*
$[A, Dirty]$	*Suck*
$[B, Clean]$	*Left*
$[B, Dirty]$	*Suck*
$[A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Dirty]$	*Suck*
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	*Right*
$[A, Clean], [A, Clean], [A, Dirty]$	*Suck*
⋮	⋮

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

Agents and Environments

- Before closing this section, we should emphasize that the notion of an *agent* is meant to be a tool for analyzing systems, not an absolute characterization that divides the world into agents and non-agents.
- One could view a hand-held calculator as an agent that chooses the action of displaying "4" when given the percept sequence "2 + 2 =," but such an analysis would hardly aid our understanding of the calculator.

Agents and Environments

- In a sense, all areas of engineering can be seen as designing artifacts that interact with the world;
- AI operates at (what the authors consider to be) the most interesting end of the spectrum, where
  - the *artifacts* have significant computational resources and
  - the *task environment* requires nontrivial decision making.

1.2 Good Behavior: The Concept Of Rationality

Good Behavior

- A rational agent is one that does the right thing—conceptually speaking, every entry in the table for the agent function is filled out correctly.
- Obviously, doing the right thing is better than doing the wrong thing,
  - but what does it mean to do the "right thing"?

Good Behavior

- We answer this age-old question in an age-old way: by considering the *consequences* of the agent's behavior.
- When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives.
- This sequence of actions causes the environment to go through a sequence of states.
- If the sequence is desirable, then the agent has performed well.
- This notion of desirability is captured by a **performance measure** that evaluates any given sequence of environment states.

Good Behavior

- Notice that we said environment states, not agent states.
- If we define success in terms of agent's opinion of its own performance, an agent could achieve perfect rationality simply by deluding itself that its performance was perfect.
- Human agents in particular are notorious for "sour grapes"—believing they did not really want something (e.g., a Nobel Prize, Induction into Rock and Roll Hall of Fame) after not getting it.

Good Behavior

- Obviously, there is not one fixed performance measure for all tasks and agents; typically, a designer will devise one appropriate to the circumstances.
  - This is not as easy as it sounds.
- Consider, for example, the vacuum-cleaner agent from the preceding section.
- We might propose to measure performance by the amount of dirt cleaned up in a single eight-hour shift.
- With a rational agent, of course, what you ask for is what you get.

Good Behavior

- A rational agent can maximize this performance measure by cleaning up the dirt, then dumping it all on the floor, then cleaning it up again, and so on.
- A more suitable performance measure would reward the agent for having a clean floor.
- For example, one point could be awarded for each clean square at each time step (perhaps with a penalty for electricity consumed and noise generated).

Good Behavior

- As a general rule, it is better to design performance measures according to
  - what one actually wants in the environment, rather than
  - according to how one thinks the agent should behave.
- Even when the obvious pitfalls are avoided, there remain some knotty issues to untangle. For example, the notion of "clean floor" in the preceding scenario is based on *average cleanliness over time*.

Good Behavior

- Yet the same average cleanliness can be achieved by two different agents, one of which does a mediocre job all the time while the other cleans energetically but takes long breaks.
- Which is preferable might seem to be a fine point of janitorial science, but in fact it is a deep philosophical question with far-reaching implications.

Good Behavior

- Which is better— a reckless life of highs and lows, or a safe but humdrum existence?
- Which is better—an economy where everyone lives in moderate poverty, or one in which some live in plenty while others are very poor?
- We leave these questions as an exercise for the diligent reader.
  - i.e. the philosophy department

2.2.1 Rationality

Rationality

- What is rational at any given time depends on four things:
  - The **performance measure** that defines the criterion of success.
  - The agent's **prior knowledge** of the environment.
  - The **actions** that the agent can perform.
  - The agent's **percept sequence** to date.
- This leads to a definition of a rational agent:

  For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Rationality

- Consider the simple vacuum-cleaner agent that cleans a square if it is dirty and moves to the other square if not; this is the agent function tabulated in Figure 2.3.
- Is this a rational agent?
  - That depends!
- First, we need to say what the performance measure is, what is known about the environment, and what sensors and actuators the agent has.

Rationality

- Let us assume the following:
  - The performance measure awards one point for each clean square at each time step, over a "lifetime" of 1000 time steps.
  - The "geography" of the environment is known a priori but the dirt distribution and the initial location of the agent are not. Clean squares stay clean and sucking cleans the current square. The Left and Right actions move the agent left and right except when this would take the agent outside the environment, in which case the agent remains where it is.
  - The only available actions are Left, Right, and Suck.
  - The agent correctly perceives its location and whether that location contains dirt.

Rationality

- One can see easily that the same agent would be irrational under different circumstances.
- For example, once all the dirt is cleaned up, the agent will oscillate needlessly back and forth; if the performance measure includes a penalty of one point for each movement left or right, the agent will fare poorly.

Rationality

- A better agent for this case would do nothing once it is sure that all the squares are clean.
- If clean squares can become dirty again, the agent should occasionally check and re-clean them if needed.
- If the geography of the environment is unknown, the agent will need to explore it rather than stick to squares A and B.

2.2.2 Omniscience, Learning, and Autonomy

Omniscience, Learning, and Autonomy

- We need to be careful to distinguish between rationality and **omniscience**.
- An omniscient agent knows the *actual* outcome of its actions and can act accordingly.
  - An agent can be omniscient in a small, finite, and closed system (e.g. a simple game, like tic-tac-toe)
  - But omniscience is impossible in the real world.

Omniscience, Learning, and Autonomy

- Consider the following example:
  - I am walking along the Champs Elysées one day and I see an old friend across the street. There is no traffic nearby and I'm not otherwise engaged, so, being rational, I start to cross the street.
  - Meanwhile, at 33,000 feet, a cargo door falls off a passing airliner, and before I make it to the other side of the street I am flattened.
  - Was I irrational to cross the street? It is unlikely that my obituary would read "Idiot attempts to cross street."

Omniscience, Learning, and Autonomy

- This example shows that rationality is not the same as perfection.
- Rationality maximizes *expected performance*, while perfection maximizes *actual performance*.
- Retreating from a requirement of perfection is not just a question of being fair to agents.
- The point is that if we expect an agent to do what turns out to be the best action after the fact, it will be impossible to design an agent to fulfill this specification—unless we improve the performance of crystal balls or time machines.

Omniscience, Learning, and Autonomy

- Our definition of rationality does not require omniscience, then, because the rational choice depends only on the percept sequence *to date*. We must also ensure that we haven't inadvertently allowed the agent to engage in decidedly underintelligent activities.
- Performing actions in order to modify future percepts—sometimes called **information gathering**—is an important part of rationality and is covered in depth in Chapter 16.
- A second example of information gathering is provided by the **exploration** that must be undertaken by a vacuum-cleaning agent in an initially unknown environment.

Omniscience, Learning, and Autonomy

- Our definition requires a rational agent not only to gather information but also to **learn** as much as possible from what it perceives.
- The agent's initial configuration could reflect some prior knowledge of the environment, but as the agent gains experience this may be modified and augmented.
- To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent lacks **autonomy**.

1.3 The Nature of Environments

Specifying the Task Environment

- Now that we have a definition of rationality, we are almost ready to think about building rational agents.
- First, however, we must think about **task environments**, which are essentially the "problems" to which rational agents are the "solutions."
- We begin by showing how to specify a task environment, illustrating the process with a number of examples.
- We then show that task environments come in a variety of flavors. The flavor of the task environment directly affects the appropriate design for the agent program.

Specifying the Task Environment

- In our discussion of the rationality of the simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent's actuators and sensors.
- We group all these under the heading of the **task environment**.
- For the acronymically minded, we call this the **PEAS** (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors) description.
- In designing an agent, the first step must always be to specify the task environment as fully as possible.

PEAS: Automated Taxi Driver

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Figure 2.4 PEAS description of the task environment for an automated taxi.

Properties of Task Environments

- The range of task environments that might arise in AI is obviously vast.
- We can, however, identify a fairly small number of dimensions along which task environments can be categorized.
- These dimensions determine, to a large extent, the appropriate agent design and the applicability of each of the principal families of techniques for agent implementation.

PEAS: Various Examples

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Figure 2.5 Examples of agent types and their PEAS descriptions.

Properties of Task Environments

- Fully Observable vs. Partially Observable
- Single agent vs. Multiagent
- Deterministic vs. Stochastic
- Episodic vs. Sequential
- Static vs. Dynamic
- Discrete vs. Continuous
- Known vs. Unknown

Fully Observable vs. Partially Observable

- If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is *fully observable*.
- A task environment is effectively fully observable if the sensors detect all aspects that are relevant to the choice of action; relevance, in turn, depends on the performance measure.
- Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world.

Fully Observable vs. Partially Observable

- An environment might be ***partially observable*** because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data
- For example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares, and an automated taxi cannot see what other drivers are thinking.

Fully Observable vs. Partially Observable

- If the agent has no sensors at all then the environment is ***unobservable***.
- One might think that in such cases the agent's plight is hopeless, but, as we discuss in Chapter 4, the agent's goals may still be achievable, sometimes with certainty.

Single agent vs. Multiagent

- The distinction between single-agent and multiagent environments may seem simple enough.
- For example, an agent solving a crossword puzzle by itself is clearly in a *single-agent* environment, whereas an agent playing chess is in a *two-agent* environment.

Single agent vs. Multiagent

- There are, however, some subtle issues.
  - First, we have described how an entity *may* be viewed as an agent, but we have not explained which entities *must* be viewed as agents.
  - Does an agent A (the taxi driver for example) have to treat an object B (another vehicle) as an agent, or can it be treated merely as an object behaving according to the laws of physics, analogous to waves at the beach or leaves blowing in the wind?
  - The key distinction is whether B's behavior is best described as maximizing a performance measure whose value depends on agent A's behavior.

Deterministic vs. Stochastic

- If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is **deterministic**.
- Otherwise, it is **stochastic**.
- We say an environment is **uncertain** if it is not fully observable or not deterministic.
- A **nondeterministic** environment is one in which actions are characterized by their *possible* outcomes, but no probabilities are attached to them.

Episodic vs. Sequential

- In an **episodic** task environment, the agent's experience is divided into atomic episodes.
  - In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes.
  - An assembly line QA agent is episodic.
- In **sequential** environments, on the other hand, the current decision could affect all future decisions.
  - Chess and taxi driving are sequential.

Static vs. Dynamic

- If the environment can change while an agent is deliberating, then we say the environment is **dynamic** for that agent,
- Otherwise, it is **static**.
- If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is **semidynamic**.
  - Taxi driving?
  - Crossword puzzle solving?
  - Chess?
    - When played with a clock, it's semidynamic

Discrete vs. Continuous

- The discrete vs. continuous distinction applies to
  - the *state* of the environment,
  - the way *time* is handled, and
  - the *percepts* and *actions* of the agent.
- For example, the chess environment has a finite number of distinct states (excluding the clock). Chess also has a discrete set of percepts and actions.
- Taxi driving is a *continuous-state* and *continuous-time* problem,

Known vs. Unknown

- Strictly speaking, this distinction refers not to the environment itself but to the agent's (or designer's) state of knowledge about the "laws of physics" of the environment.
- In a **known** environment, the outcomes (or outcome probabilities if the environment is stochastic) for all actions are given.
- If the environment is **unknown**, the agent will have to learn how it works in order to make good decisions.

Known vs. Unknown

- Note that the distinction between known and unknown environments is not the same as the one between fully and partially observable environments. It is quite possible for a known environment to be partially observable.
- For example, in solitaire card games, we know the rules but are still unable to see the cards that have not yet been turned over.
- Conversely, an unknown environment can be fully observable—in a new video game, the screen may show the entire game state but I still don't know what the buttons do until I try them.

Task Environment Assessment

- As one might expect, the hardest case is:
  - partially observable
  - multiagent
  - stochastic
  - sequential
  - dynamic
  - continuous, and
  - unknown

Task Environment Assessment

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Figure 2.6 Examples of task environments and their characteristics.

1.4 The Structure of Agents

Agent Programs

- In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the actuators as they are generated.
- **Most of this course is about designing agent programs.**
- The agent programs that we design in this course all have the same skeleton: they take the current percept as input from the sensors and return an action to the actuators.

Agent Programs

- Notice the difference between the **agent program**, which takes the only current percept as input, and the **agent function**, which takes the entire percept history.
- The agent program takes just the current percept as input because nothing more is available from the environment
- If the agent's actions need to depend on some of all of the percept sequence, then the agent will have to *remember the percepts*.

Table-driven Agent

function TABLE-DRIVEN-AGENT(*percept*) **returns** an action
  persistent: *percepts*, a sequence, initially empty
    table, a table of actions, indexed by percept sequences, initially fully specified

  append *percept* to the end of *percepts*
  action ← LOOKUP(*percepts*, *table*)
  return *action*

Figure 2.7 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

Reflex Vacuum Agent

```
function REFLEX-VACUUM-AGENT([location,status]) returns an action

    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

Figure 2.8 The agent program for a simple reflex agent in the two-state vacuum environment. This program implements the agent function tabulated in Figure 2.3.

Agent Programs

- In the remainder of this section, we outline four basic kinds of agent programs that embody the principles underlying almost all intelligent systems:
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
  - Utility-based agents
- Each kind of agent program combines particular components in particular ways to generate actions.

Simple Reflex Agents

- The simplest kind of agent is the **simple reflex agent**. These agents select actions on the basis of the current percept, ignoring the rest of the percept history.
- Notice that the vacuum agent program is very small indeed (Fig. 2.10) compared to the corresponding table (Fig. 2.3).
- The most obvious reduction comes from ignoring the percept history, which cuts down the number of possibilities from 4^T to just 4.
- A further, small reduction comes from the fact that when the current square is dirty, the action does not depend on the location.

Simple Reflex Agents

- Simple reflex behaviors can be useful even in more complex environments.
- Imagine yourself as the *driver agent* of the automated taxi. If the car in front brakes and its brake lights come on, then you should notice this and initiate braking.
- We call such a connection a **condition–action rule**[1], written as:
  - **if** *car-in-front-is-braking* **then** *initiate-braking*.

1. Also called **situation–action rules**, **productions**, or **if–then rules**.

Simple Reflex Agents

- The program in Figure 2.8 is specific to one particular vacuum environment.
- A more general and flexible approach is
  - First build a general-purpose interpreter for *condition–action rules*
  - Then to create *rule sets* for specific task environments.
- Figure 2.9 (on the next slide) gives the structure of this general program in schematic form, showing how the condition–action rules allow the agent to make the connection from percept to action.

Simple Reflex Agents

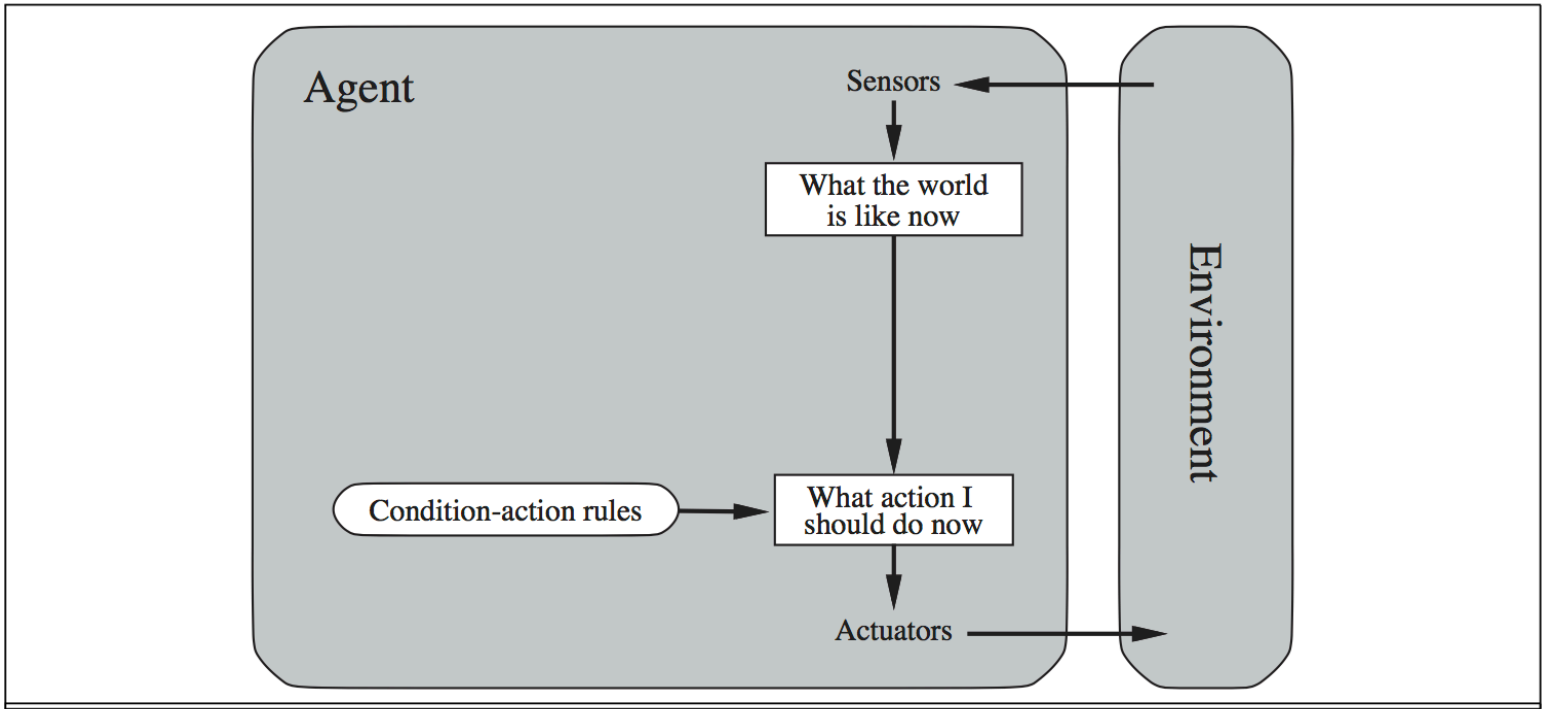

Figure 2.9 Schematic diagram of a simple reflex agent.

Simple Reflex Agents

function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *rules*, a set of condition–action rules

  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

Model-based Reflex Agents

- The most effective way to handle *partial observability* is for the agent to keep track of the part of the world it can't see now.
- That is, the agent should maintain some sort of internal state that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.
- For the braking problem, the internal state is not too extensive—just the previous frame from the camera, allowing the agent to detect when two red lights at the edge of the vehicle go on or off simultaneously.

Model-based Reflex Agents

- For other driving tasks such as changing lanes, the agent needs to keep track of where the other cars are if it can't see them all at once.
- However, the real world is even *far more* complex…
- For example, for any driving to be possible at all, the agent needs to keep track of where its keys are.
- Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program.

Model-based Reflex Agents

- ***First***, we need some information about how the world evolves *independently* of the agent.
  - For example, that an overtaking car generally will be closer behind than it was a moment ago.
- ***Second***, we need some information about how the agent's *own actions* affect the world.
  - For example, that when the agent turns the steering wheel clockwise, the car turns to the right, or that after driving for five minutes northbound on the freeway, one is usually about five miles north of where one was five minutes ago.

Model-based Reflex Agents

- This knowledge about "how the world works"—whether implemented in simple Boolean circuits or in complete scientific theories—is called a **model** of the world.
- An agent that uses such a model is called a **model-based agent**.

Model-based Reflex Agents

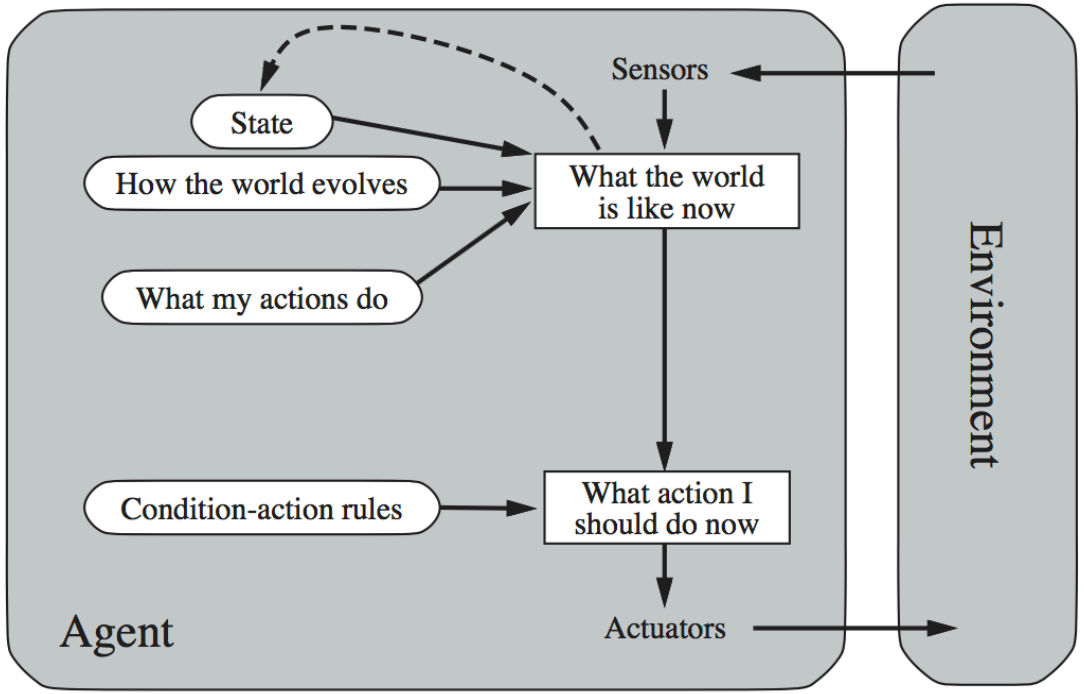

Figure 2.11 A model-based reflex agent.

Model-based Reflex Agents

function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *state*, the agent's current conception of the world state
    model, a description of how the next state depends on current state and action
    rules, a set of condition–action rules
    action, the most recent action, initially none

  state ← UPDATE-STATE(*state*, *action*, *percept*, *model*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

Model-based Reflex Agents

- Regardless of the kind of representation used, it is seldom possible for the agent to determine the current state of a partially observable environment exactly.
- Instead, the box labeled "what the world is like now" (Figure 2.11) represents the agent's "best guess" (or sometimes best guesses).
  - For example, an automated taxi may not be able to see around the large truck that has stopped in front of it and can only guess about what may be causing the hold-up.
  - Thus, uncertainty about the current state may be unavoidable, but the agent still has to make a decision.

Model-based Reflex Agents

- Regardless of the kind of representation used, it is seldom possible for the agent to determine the current state of a partially observable environment exactly.
- Instead, the box labeled "what the world is like now" (Figure 2.11) represents the agent's "best guess" (or sometimes best guesses).
  - For example, an automated taxi may not be able to see around the large truck that has stopped in front of it and can only guess about what may be causing the hold-up.
  - Thus, uncertainty about the current state may be unavoidable, but the agent still has to make a decision.

Goal-based Agents

- Knowing something about the current state of the environment is not always enough to decide what to do.
  - For example, at a road junction, the taxi can turn left, turn right, or go straight on.
  - The correct decision depends on *where the taxi is trying to get to*.
  - In other words, as well as a current state description, the agent needs some sort of **goal** information that describes situations that are desirable —for example, being at the passenger's destination.

Goal-based Agents

- Knowing something about the current state of the environment is not always enough to decide what to do.
  - For example, at a road junction, the taxi can turn left, turn right, or go straight on.
  - The correct decision depends on *where the taxi is trying to get to*.
  - In other words, as well as a current state description, the agent needs some sort of **goal** information that describes situations that are desirable —for example, being at the passenger's destination.

Goal-based Agents

- The agent program can combine this with the model—*the same information as was used in the* ***model-based reflex agent***—to choose actions that achieve the goal.
- Figure 2.13 shows the goal-based agent's structure.

Goal-based Agents

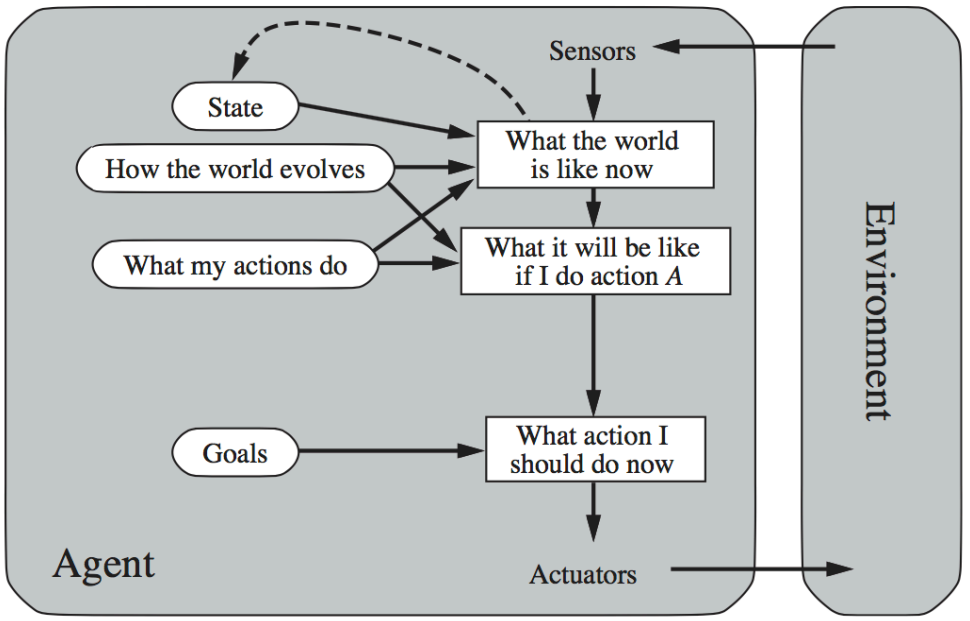

Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

Goal-based Agents

- Sometimes goal-based action selection is straightforward—for example, when goal satisfaction results immediately from a single action.
- Sometimes it will be more tricky—for example, when the agent has to consider long sequences of twists and turns in order to find a way to achieve the goal.
- **Search** (Chapters 3 to 5) and **planning** (Chapters 10 and 11) are the subfields of AI devoted to finding *action sequences* that achieve the agent's goals.

Goal-based Agents

- Notice that decision making of this kind is fundamentally different from the condition—action rules described earlier, in that it involves consideration of the future—both "What will happen if I do such-and-such?" and "Will that make me happy?"
- In the **reflex agent** designs, this information is not explicitly represented, because the built-in rules map directly from percepts to actions.

Goal-based Agents

- The **reflex agent** brakes when it sees brake lights.
- A **goal-based agent**, in principle, could reason that if the car in front has its brake lights on, it will slow down.
- Given the way the world usually evolves, the only action that will achieve the goal of not hitting other cars is to brake.

Utility-based Agents

- Goals alone are not enough to generate high-quality behavior in most environments.
- For example, many action sequences will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others.
- Goals just provide a crude binary distinction between “happy” and “unhappy” states.

Utility-based Agents

- A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent.
- Because "happy" does not sound very scientific, economists and computer scientists use the term **utility** instead.

Utility-based Agents

- We have already seen that a performance measure assigns a score to any given sequence of environment states, so it can easily distinguish between more and less desirable ways of getting to the taxi's destination.
- An agent's **utility function** is essentially an internalization of the performance measure.
- If the internal utility function and the external performance measure are in agreement, then an agent that chooses actions to maximize its utility will be rational according to the external performance measure.

Utility-based Agents

- Let us emphasize again that this is not the only way to be rational
  - We have already seen a rational agent program for the vacuum world (Figure 2.8) that has no idea what its utility function is.
- But, like goal-based agents, a **utility-based agent** has many advantages in terms of flexibility and learning.
- Furthermore, in two kinds of cases, goals are inadequate but a utility-based agent can still make rational decisions.

Utility-based Agents

- ***First***, when there are conflicting goals, only some of which can be achieved (for example, speed and safety), the utility function specifies the appropriate tradeoff.
- ***Second***, when there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed against the importance of the goals.

Utility-based Agents

- Partial observability and stochasticity are ubiquitous in the real world, and so, therefore, is decision making under uncertainty.
- Technically speaking, a rational utility-based agent chooses the action that maximizes the **expected utility** of the action outcomes
  - That is, the utility the agent expects to derive, on average, given the probabilities and utilities of each outcome,

Utility-based Agents

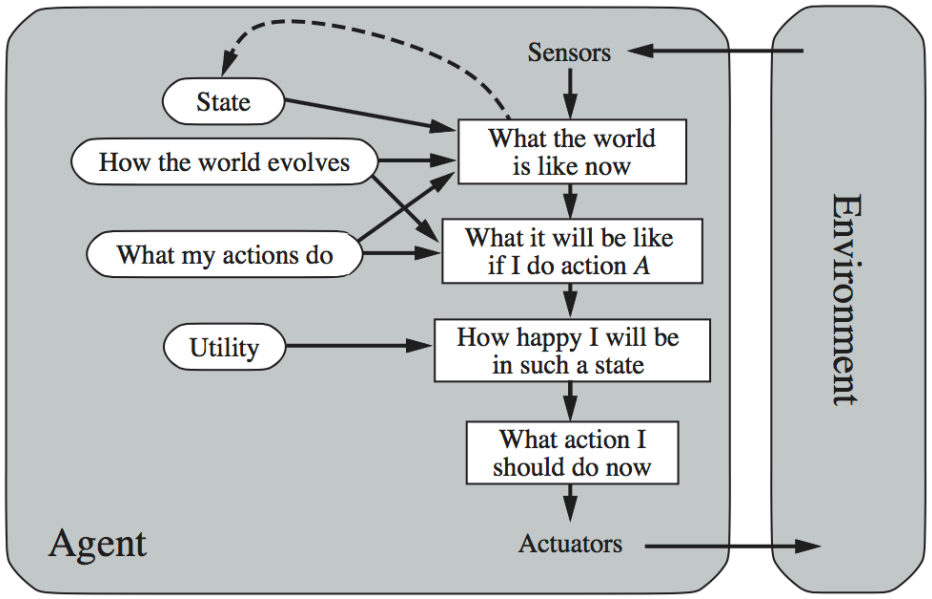

Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

Utility-based Agents

- At this point, the reader may be wondering, “Is it that simple? We just build agents that maximize expected utility, and we’re done?”
- It’s true that such agents would be intelligent, but it’s not simple.
- A utility-based agent has to model and keep track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning.

Utility-based Agents

- The results of this research fill many of the chapters of this book. Choosing the utility-maximizing course of action is also a difficult task, requiring ingenious algorithms that fill several more chapters.
  - Even with these algorithms, perfect rationality is usually unachievable in practice because of computational complexity, as we noted in Chapter 1.

Learning Agents

- We have described agent programs with various methods for selecting actions.
- We have not, so far, explained how the agent programs come into being.
- In his famous early paper, Turing (1950) considers the idea of actually programming his intelligent machines by hand.
- He estimates how much work this might take and concludes "Some more expeditious method seems desirable."

Learning Agents

- The method he proposes is to build learning machines and then to teach them. In many areas of AI, this is now the preferred method for creating state-of-the-art systems.
- Learning has another advantage, as we noted earlier: it allows the agent to operate in initially unknown environments and to become more competent than its initial knowledge alone might allow.

Learning Agents

- In this section, we briefly introduce the main ideas of learning agents.
- Throughout the book, we comment on opportunities and methods for learning in particular kinds of agents.
  - Part V goes into much more depth on the learning algorithms themselves.

Learning Agents

- A learning agent can be divided into four conceptual components, as shown in Figure 2.15.
- The most important distinction is between the learning element, which is responsible for making improvements, and the performance element, which is responsible for selecting external actions.
- The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.

Learning Agents

- The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.
- The design of the learning element depends very much on the design of the performance element.
- When trying to design an agent that learns a certain capability, the first question is ***not*** "How am I going to get it to learn this?"
  - …but "What kind of performance element will my agent need to do this once it has learned how?"

Learning Agents

- The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.
- The design of the learning element depends very much on the design of the performance element.
- When trying to design an agent that learns a certain capability, the first question is ***not*** "How am I going to get it to learn this?"
  - …but "What kind of performance element will my agent need to do this once it has learned how?"

Learning Agents

- The critic tells the learning element how well the agent is doing with respect to a fixed performance standard.
  - The critic is necessary because the *percepts* themselves provide no indication of the agent's success.
  - For example, a chess program could receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so.

Learning Agents

- It is important that the performance standard be fixed.
- Conceptually, one should think of it as being outside the agent altogether because the agent must not modify it to fit its own behavior.

Learning Agents

- The last component of the learning agent is the **problem generator**. It is responsible for suggesting actions that will lead to new and informative experiences.
- The point is that if the performance element had its way, it would keep doing the actions that are best, given what it knows.
- But if the agent is willing to explore a little and do some perhaps suboptimal actions in the short run, it might discover much better actions for the long run.

Learning Agents

- The problem generator's job is to suggest these exploratory actions.
- This is what scientists do when they carry out experiments.

Learning Agents

- To make the overall design more concrete, let us return to the automated taxi example.
- The performance element consists of whatever collection of knowledge and procedures the taxi has for selecting its driving actions.
- The taxi goes out on the road and drives, using this performance element. The critic observes the world and passes information along to the learning element.

Learning Agents

- For example, after the taxi makes a quick left turn across three lanes of traffic, the critic observes the shocking language used by other drivers.
  - From this experience, the learning element is able to formulate a rule saying this was a bad action, and the performance element is *modified by installation of the new rule*.
  - The problem generator might identify certain areas of behavior in need of improvement and suggest experiments, such as trying out the brakes on different road surfaces under different conditions.

How components of agent programs work

- We have described agent programs (in very high-level terms) as consisting of various components, whose function it is to answer questions such as:
  - "What is the world like now?"
  - "What action should I do now?"
  - "What do my actions do?"
- The next question for a student of AI is, "How on earth do these components work?"

How components of agent programs work

- It takes about a thousand pages to begin to answer that question properly, but here we want to draw the reader's attention to some basic distinctions among the various ways that the components can represent the environment that the agent inhabits.

How components of agent programs work

- Roughly speaking, we can place the representations along an axis of increasing complexity and expressive power—**atomic**, **factored**, and **structured**.
- To illustrate these ideas, it helps to consider a particular agent component, such as the one that deals with "What my actions do."
- This component describes the changes that might occur in the environment as the result of taking an action, and Figure 2.16 provides schematic depictions of how those transitions might be represented.

How components of agent programs work

- In an **atomic representation** each state of the world is indivisible —it has no internal structure.
- Consider the problem of finding a driving route from one end of a country to the other via some sequence of cities (we address this problem in Figure 3.2 on page 68).
- For the purposes of solving this problem, it may suffice to reduce the state of world to just the name of the city we are in—a single atom of knowledge; a "black box" whose only discernible property is that of being identical to or different from another black box.

How components of agent programs work

- The algorithms underlying
  - **search** and **game-playing** (Chapters 3–5),
  - **Hidden Markov models** (Chapter 15), and
  - **Markov decision processes** (Chapter 17)
- all work with atomic representations—or, at least, they treat representations as if they were atomic.

How components of agent programs work

- Now consider a higher-fidelity description for the same problem, where we need to be concerned with more than just atomic location in one city or another; we might need to pay attention to how much gas is in the tank, our current GPS coordinates, whether or not the oil warning light is working, how much spare change we have for toll crossings, what station is on the radio, and so on.

How components of agent programs work

- A **factored representation** splits up each state into a fixed set of **variables** or **attributes**, each of which can have a **value**.
- While two different atomic states have nothing in common—they are just different black boxes—two different factored states can share some attributes (such as being at some particular GPS location) and not others (such as having lots of gas or having no gas); this makes it much easier to work out how to turn one state into another.

Representing States

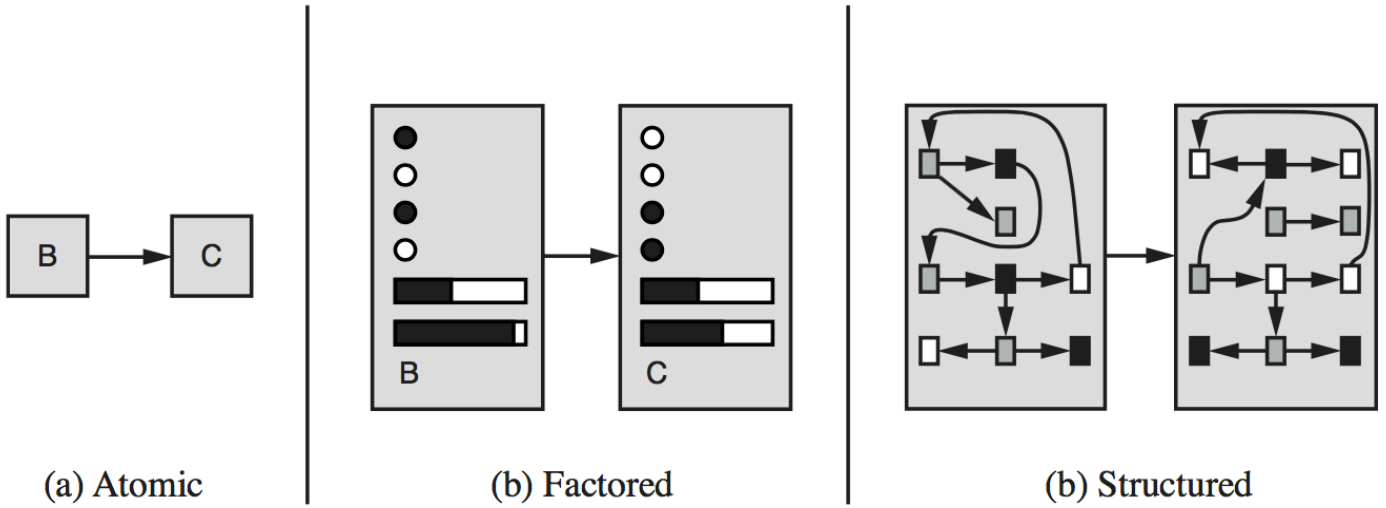

Figure 2.16 Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

How components of agent programs work

- With factored representations, we can also represent uncertainty—for example, ignorance about the amount of gas in the tank can be represented by leaving that attribute blank.
- Many important areas of AI are based on factored representations, including
  - **constraint satisfaction** algorithms (Chapter 6),
  - **propositional logic** (Chapter 7),
  - **planning** (Chapters 10 and 11),
  - **Bayesian networks** (Chapters 13–16), and
  - the **machine learning** algorithms in Chapters 18, 20, and 21.

How components of agent programs work

- For many purposes, we need to understand the world as having *things* in it that are *related* to each other, not just variables with values.
- For example, we might notice that a large truck ahead of us is reversing into the driveway of a dairy farm but a cow has got loose and is blocking the truck's path.

How components of agent programs work

- A factored representation is unlikely to be pre-equipped with the attribute *TruckAheadBackingIntoDairyFarmDrivewayBlockedByLooseCow* with value true or false.
- Instead, we would need a **structured representation**, in which objects such as cows and trucks and their various and varying relationships can be described explicitly.

How components of agent programs work

- Structured representations underlie
  - **relational databases** and **first-order logic** (Chapters 8, 9, and 12),
  - **first-order probability models** (Chapter 14),
  - **knowledge-based learning** (Chapter 19) and
  - much of **natural language understanding** (Chapters 22 and 23).
- In fact, almost everything that humans express in natural language concerns objects and their relationships.