real thing. Compare the simple state description we have chosen, *In(Arad)*, to an actual cross-country trip, where the state of the world includes so many things: the traveling companions, the current radio program, the scenery out of the window, the proximity of law enforcement officers, the distance to the next rest stop, the condition of the road, the weather, and so on. All these considerations are left out of our state descriptions because they are irrelevant to the problem of finding a route to Bucharest. The process of removing detail from a representation is called **abstraction**.

ABSTRACTION

In addition to abstracting the state description, we must abstract the actions themselves. A driving action has many effects. Besides changing the location of the vehicle and its occupants, it takes up time, consumes fuel, generates pollution, and changes the agent (as they say, travel is broadening). Our formulation takes into account only the change in location. Also, there are many actions that we omit altogether: turning on the radio, looking out of the window, slowing down for law enforcement officers, and so on. And of course, we don't specify actions at the level of "turn steering wheel to the left by one degree."

Can we be more precise about defining the appropriate level of abstraction? Think of the abstract states and actions we have chosen as corresponding to large sets of detailed world states and detailed action sequences. Now consider a solution to the abstract problem: for example, the path from Arad to Sibiu to Rimnicu Vilcea to Pitesti to Bucharest. This abstract solution corresponds to a large number of more detailed paths. For example, we could drive with the radio on between Sibiu and Rimnicu Vilcea, and then switch it off for the rest of the trip. The abstraction is *valid* if we can expand any abstract solution into a solution in the more detailed world; a sufficient condition is that for every detailed state that is "in Arad," there is a detailed path to some state that is "in Sibiu," and so on.[5] The abstraction is *useful* if carrying out each of the actions in the solution is easier than the original problem; in this case they are easy enough that they can be carried out without further search or planning by an average driving agent. The choice of a good abstraction thus involves removing as much detail as possible while retaining validity and ensuring that the abstract actions are easy to carry out. Were it not for the ability to construct useful abstractions, intelligent agents would be completely swamped by the real world.

3.2 EXAMPLE PROBLEMS

The problem-solving approach has been applied to a vast array of task environments. We list some of the best known here, distinguishing between *toy* and *real-world* problems. A **toy problem** is intended to illustrate or exercise various problem-solving methods. It can be given a concise, exact description and hence is usable by different researchers to compare the performance of algorithms. A **real-world problem** is one whose solutions people actually care about. Such problems tend not to have a single agreed-upon description, but we can give the general flavor of their formulations.

TOY PROBLEM

REAL-WORLD PROBLEM

[5] See Section 11.2 for a more complete set of definitions and algorithms.

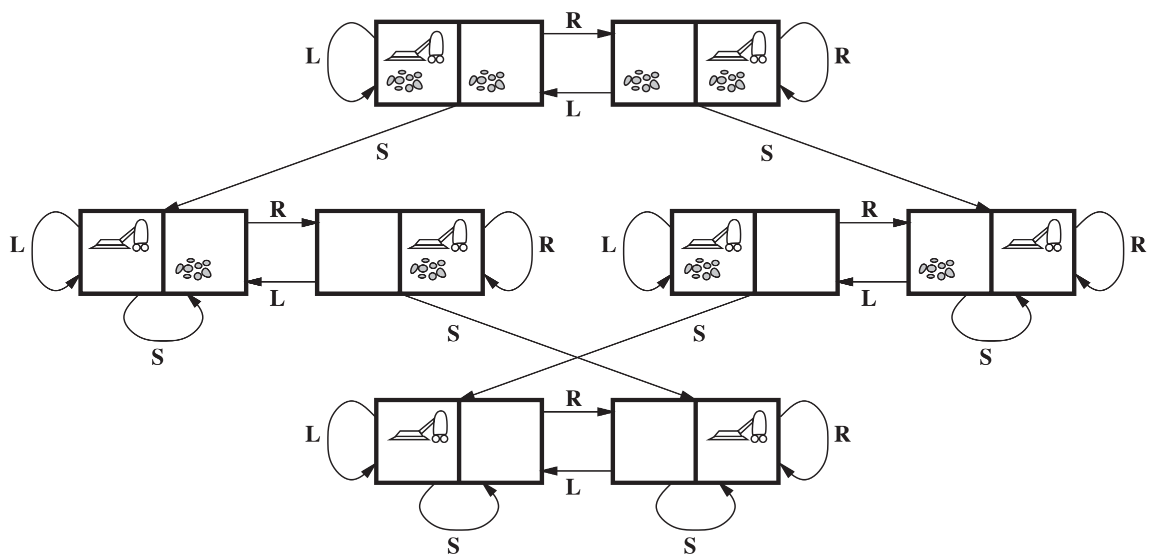

Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

3.2.1 Toy problems

The first example we examine is the **vacuum world** first introduced in Chapter 2. (See Figure 2.2.) This can be formulated as a problem as follows:

- **States**: The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus, there are $2 \times 2^2 = 8$ possible world states. A larger environment with n locations has $n \cdot 2^n$ states.
- **Initial state**: Any state can be designated as the initial state.
- **Actions**: In this simple environment, each state has just three actions: *Left*, *Right*, and *Suck*. Larger environments might also include *Up* and *Down*.
- **Transition model**: The actions have their expected effects, except that moving *Left* in the leftmost square, moving *Right* in the rightmost square, and *Suck*ing in a clean square have no effect. The complete state space is shown in Figure 3.3.
- **Goal test**: This checks whether all the squares are clean.
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

Compared with the real world, this toy problem has discrete locations, discrete dirt, reliable cleaning, and it never gets any dirtier. Chapter 4 relaxes some of these assumptions.

8-PUZZLE

The **8-puzzle**, an instance of which is shown in Figure 3.4, consists of a 3×3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The object is to reach a specified goal state, such as the one shown on the right of the figure. The standard formulation is as follows:

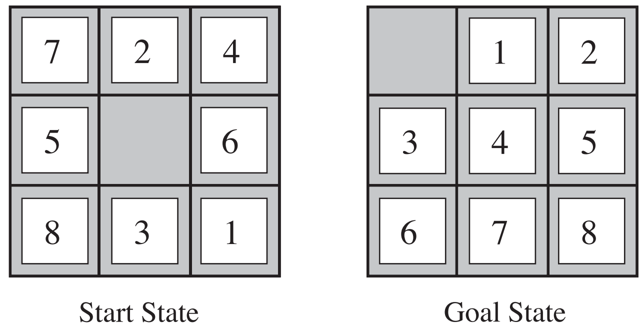

Figure 3.4 A typical instance of the 8-puzzle.

- **States**: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- **Initial state**: Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states (Exercise 3.4).
- **Actions**: The simplest formulation defines the actions as movements of the blank space *Left*, *Right*, *Up*, or *Down*. Different subsets of these are possible depending on where the blank is.
- **Transition model**: Given a state and action, this returns the resulting state; for example, if we apply *Left* to the start state in Figure 3.4, the resulting state has the 5 and the blank switched.
- **Goal test**: This checks whether the state matches the goal configuration shown in Figure 3.4. (Other goal configurations are possible.)
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

What abstractions have we included here? The actions are abstracted to their beginning and final states, ignoring the intermediate locations where the block is sliding. We have abstracted away actions such as shaking the board when pieces get stuck and ruled out extracting the pieces with a knife and putting them back again. We are left with a description of the rules of the puzzle, avoiding all the details of physical manipulations.

SLIDING-BLOCK PUZZLES

The 8-puzzle belongs to the family of **sliding-block puzzles**, which are often used as test problems for new search algorithms in AI. This family is known to be NP-complete, so one does not expect to find methods significantly better in the worst case than the search algorithms described in this chapter and the next. The 8-puzzle has $9!/2 = 181,440$ reachable states and is easily solved. The 15-puzzle (on a 4×4 board) has around 1.3 trillion states, and random instances can be solved optimally in a few milliseconds by the best search algorithms. The 24-puzzle (on a 5×5 board) has around 10^{25} states, and random instances take several hours to solve optimally.

8-QUEENS PROBLEM

The goal of the **8-queens problem** is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.) Figure 3.5 shows an attempted solution that fails: the queen in the rightmost column is attacked by the queen at the top left.

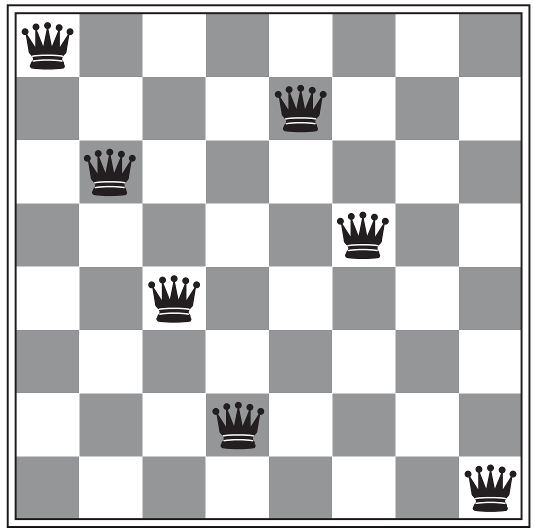

Figure 3.5 Almost a solution to the 8-queens problem. (Solution is left as an exercise.)

INCREMENTAL FORMULATION

COMPLETE-STATE FORMULATION

Although efficient special-purpose algorithms exist for this problem and for the whole n-queens family, it remains a useful test problem for search algorithms. There are two main kinds of formulation. An **incremental formulation** involves operators that *augment* the state description, starting with an empty state; for the 8-queens problem, this means that each action adds a queen to the state. A **complete-state formulation** starts with all 8 queens on the board and moves them around. In either case, the path cost is of no interest because only the final state counts. The first incremental formulation one might try is the following:

- **States**: Any arrangement of 0 to 8 queens on the board is a state.
- **Initial state**: No queens on the board.
- **Actions**: Add a queen to any empty square.
- **Transition model**: Returns the board with a queen added to the specified square.
- **Goal test**: 8 queens are on the board, none attacked.

In this formulation, we have $64 \cdot 63 \cdots 57 \approx 1.8 \times 10^{14}$ possible sequences to investigate. A better formulation would prohibit placing a queen in any square that is already attacked:

- **States**: All possible arrangements of n queens $(0 \leq n \leq 8)$, one per column in the leftmost n columns, with no queen attacking another.
- **Actions**: Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.

This formulation reduces the 8-queens state space from 1.8×10^{14} to just 2,057, and solutions are easy to find. On the other hand, for 100 queens the reduction is from roughly 10^{400} states to about 10^{52} states (Exercise 3.5)—a big improvement, but not enough to make the problem tractable. Section 4.1 describes the complete-state formulation, and Chapter 6 gives a simple algorithm that solves even the million-queens problem with ease.

Our final toy problem was devised by Donald Knuth (1964) and illustrates how infinite state spaces can arise. Knuth conjectured that, starting with the number 4, a sequence of factorial, square root, and floor operations will reach any desired positive integer. For example, we can reach 5 from 4 as follows:

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5 \,.$$

The problem definition is very simple:

- **States**: Positive numbers.
- **Initial state**: 4.
- **Actions**: Apply factorial, square root, or floor operation (factorial for integers only).
- **Transition model**: As given by the mathematical definitions of the operations.
- **Goal test**: State is the desired positive integer.

To our knowledge there is no bound on how large a number might be constructed in the process of reaching a given target—for example, the number 620,448,401,733,239,439,360,000 is generated in the expression for 5—so the state space for this problem is infinite. Such state spaces arise frequently in tasks involving the generation of mathematical expressions, circuits, proofs, programs, and other recursively defined objects.

3.2.2 Real-world problems

ROUTE-FINDING PROBLEM

We have already seen how the **route-finding problem** is defined in terms of specified locations and transitions along links between them. Route-finding algorithms are used in a variety of applications. Some, such as Web sites and in-car systems that provide driving directions, are relatively straightforward extensions of the Romania example. Others, such as routing video streams in computer networks, military operations planning, and airline travel-planning systems, involve much more complex specifications. Consider the airline travel problems that must be solved by a travel-planning Web site:

- **States**: Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and their status as domestic or international, the state must record extra information about these "historical" aspects.
- **Initial state**: This is specified by the user's query.
- **Actions**: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
- **Transition model**: The state resulting from taking a flight will have the flight's destination as the current location and the flight's arrival time as the current time.
- **Goal test**: Are we at the final destination specified by the user?
- **Path cost**: This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.