Instructor's Manual:
Exercise Solutions
for

Artificial Intelligence

A Modern Approach

Third Edition (International Version)

Stuart J. Russell and Peter Norvig

with contributions from
Ernest Davis, Nicholas J. Hay, and Mehran Sahami

Prentice Hall
Upper Saddle River Boston Columbus San Francisco New York
Indianapolis London Toronto Sydney Singapore Tokyo Montreal
Dubai Madrid Hong Kong Mexico City Munich Paris Amsterdam Cape Town

Editor-in-Chief: Michael Hirsch
Executive Editor: Tracy Dunkelberger
Assistant Editor: Melinda Haggerty
Editorial Assistant: Allison Michael
Vice President, Production: Vince O'Brien
Senior Managing Editor: Scott Disanno
Production Editor: Jane Bonnell
Interior Designers: Stuart Russell and Peter Norvig

Library of Congress Cataloging-in-Publication Data on File

Prentice Hall
is an imprint of

www.pearsonhighered.com

10 9 8 7 6 5 4 3 2 1
ISBN-13: 978-0-13-606738-2
ISBN-10: 0-13-606738-7

Preface

This Instructor's Solution Manual provides solutions (or at least solution sketches) for almost all of the 400 exercises in *Artificial Intelligence: A Modern Approach (Third Edition)*. We only give actual code for a few of the programming exercises; writing a lot of code would not be that helpful, if only because we don't know what language you prefer.

In many cases, we give ideas for discussion and follow-up questions, and we try to explain *why* we designed each exercise.

There is more supplementary material that we want to offer to the instructor, but we have decided to do it through the medium of the World Wide Web rather than through a CD or printed Instructor's Manual. The idea is that this solution manual contains the material that must be kept secret from students, but the Web site contains material that can be updated and added to in a more timely fashion. The address for the web site is:

`http://aima.cs.berkeley.edu`

and the address for the online Instructor's Guide is:

`http://aima.cs.berkeley.edu/instructors.html`

There you will find:

- Instructions on how to join the **aima-instructors** discussion list. We strongly recommend that you join so that you can receive updates, corrections, notification of new versions of this Solutions Manual, additional exercises and exam questions, etc., in a timely manner.
- Source code for programs from the text. We offer code in Lisp, Python, and Java, and point to code developed by others in C++ and Prolog.
- Programming resources and supplemental texts.
- Figures from the text, for making your own slides.
- Terminology from the index of the book.
- Other courses using the book that have home pages on the Web. You can see example syllabi and assignments here. Please *do not* put solution sets for AIMA exercises on public web pages!
- AI Education information on teaching introductory AI courses.
- Other sites on the Web with information on AI. Organized by chapter in the book; check this for supplemental material.

We welcome suggestions for new exercises, new environments and agents, etc. The book belongs to you, the instructor, as much as us. We hope that you enjoy teaching from it, that these supplemental materials help, and that you will share your supplements and experiences with other instructors.

Solutions for Chapter 1
Introduction

1.1

a. Dictionary definitions of **intelligence** talk about "the capacity to acquire and apply knowledge" or "the faculty of thought and reason" or "the ability to comprehend and profit from experience." These are all reasonable answers, but if we want something quantifiable we would use something like "the ability to apply knowledge in order to perform better in an environment."

b. We define **artificial intelligence** as the study and construction of agent programs that perform well in a given environment, for a given agent architecture.

c. We define an **agent** as an entity that takes action in response to percepts from an environment.

d. We define **rationality** as the property of a system which does the "right thing" given what it knows. See Section 2.2 for a more complete discussion. Both describe perfect rationality, however; see Section 27.3.

e. We define **logical reasoning** as the a process of deriving new sentences from old, such that the new sentences are necessarily true if the old ones are true. (Notice that does not refer to any specific syntax oor formal language, but it does require a well-defined notion of truth.)

1.2 See the solution for exercise 26.1 for some discussion of potential objections.

The probability of fooling an interrogator depends on just how unskilled the interrogator is. One entrant in the 2002 Loebner prize competition (which is not quite a real Turing Test) did fool one judge, although if you look at the transcript, it is hard to imagine what that judge was thinking. There certainly have been examples of a chatbot or other online agent fooling humans. For example, see See Lenny Foner's account of the Julia chatbot at foner.www.media.mit.edu/people/foner/Julia/. We'd say the chance today is something like 10%, with the variation depending more on the skill of the interrogator rather than the program. In 50 years, we expect that the entertainment industry (movies, video games, commercials) will have made sufficient investments in artificial actors to create very credible impersonators.

1.3 Yes, they are rational, because slower, deliberative actions would tend to result in more damage to the hand. If "intelligent" means "applying knowledge" or "using thought and reasoning" then it does not require intelligence to make a reflex action.

1.4 No. IQ test scores correlate well with certain other measures, such as success in college, ability to make good decisions in complex, real-world situations, ability to learn new skills and subjects quickly, and so on, but *only* if they're measuring fairly normal humans. The IQ test doesn't measure everything. A program that is specialized only for IQ tests (and specialized further only for the analogy part) would very likely perform poorly on other measures of intelligence. Consider the following analogy: if a human runs the 100m in 10 seconds, we might describe him or her as *very athletic* and expect competent performance in other areas such as walking, jumping, hurdling, and perhaps throwing balls; but we would not desscribe a Boeing 747 as *very athletic* because it can cover 100m in 0.4 seconds, nor would we expect it to be good at hurdling and throwing balls.

Even for humans, IQ tests are controversial because of their theoretical presuppositions about innate ability (distinct from training effects) adn the generalizability of results. See *The Mismeasure of Man* by Stephen Jay Gould, Norton, 1981 or *Multiple intelligences: the theory in practice* by Howard Gardner, Basic Books, 1993 for more on IQ tests, what they measure, and what other aspects there are to "intelligence."

1.5 In order of magnitude figures, the computational power of the computer is 100 times larger.

1.6 Just as you are unaware of all the steps that go into making your heart beat, you are also unaware of most of what happens in your thoughts. You do have a conscious awareness of some of your thought processes, but the majority remains opaque to your consciousness. The field of psychoanalysis is based on the idea that one needs trained professional help to analyze one's own thoughts.

1.7

- Although bar code scanning is in a sense computer vision, these are not AI systems. The problem of reading a bar code is an extremely limited and artificial form of visual interpretation, and it has been carefully designed to be as simple as possible, given the hardware.
- In many respects. The problem of determining the relevance of a web page to a query is a problem in natural language understanding, and the techniques are related to those we will discuss in Chapters 22 and 23. Search engines like Ask.com, which group the retrieved pages into categories, use clustering techniques analogous to those we discuss in Chapter 20. Likewise, other functionalities provided by a search engines use intelligent techniques; for instance, the spelling corrector uses a form of data mining based on observing users' corrections of their own spelling errors. On the other hand, the problem of indexing billions of web pages in a way that allows retrieval in seconds is a problem in database design, not in artificial intelligence.
- To a limited extent. Such menus tends to use vocabularies which are very limited – e.g. the digits, "Yes", and "No" — and within the designers' control, which greatly simplifies the problem. On the other hand, the programs must deal with an uncontrolled space of all kinds of voices and accents.

The voice activated directory assistance programs used by telephone companies, which must deal with a large and changing vocabulary are certainly AI programs.

- This is borderline. There is something to be said for viewing these as intelligent agents working in cyberspace. The task is sophisticated, the information available is partial, the techniques are heuristic (not guaranteed optimal), and the state of the world is dynamic. All of these are characteristic of intelligent activities. On the other hand, the task is very far from those normally carried out in human cognition.

1.8 Presumably the brain has evolved so as to carry out this operations on visual images, but the mechanism is only accessible for one particular purpose in this particular cognitive task of image processing. Until about two centuries ago there was no advantage in people (or animals) being able to compute the convolution of a Gaussian for any other purpose.

The really interesting question here is what we mean by saying that the "actual person" can do something. The person can see, but he cannot compute the convolution of a Gaussian; but computing that convolution is *part* of seeing. This is beyond the scope of this solution manual.

1.9 Evolution tends to perpetuate organisms (and combinations and mutations of organisms) that are successful enough to reproduce. That is, evolution favors organisms that can optimize their performance measure to at least survive to the age of sexual maturity, and then be able to win a mate. Rationality just means optimizing performance measure, so this is in line with evolution.

1.10 This question is intended to be about the essential nature of the AI problem and what is required to solve it, but could also be interpreted as a sociological question about the current practice of AI research.

A *science* is a field of study that leads to the acquisition of empirical knowledge by the scientific method, which involves falsifiable hypotheses about what is. A pure *engineering* field can be thought of as taking a fixed base of empirical knowledge and using it to solve problems of interest to society. Of course, engineers do bits of science—e.g., they measure the properties of building materials—and scientists do bits of engineering to create new devices and so on.

As described in Section 1.1, the "human" side of AI is clearly an empirical science—called cognitive science these days—because it involves psychological experiments designed out to find out how human cognition actually works. What about the the "rational" side? If we view it as studying the abstract relationship among an arbitrary task environment, a computing device, and the program for that computing device that yields the best performance in the task environment, then the rational side of AI is really mathematics and engineering; it does not require any empirical knowledge about the *actual* world—and the *actual* task environment—that we inhabit; that a given program will do well in a given environment is a *theorem*. (The same is true of pure decision theory.) In practice, however, we are interested in task environments that do approximate the actual world, so even the rational side of AI involves finding out what the actual world is like. For example, in studying rational agents that communicate, we are interested in task environments that contain humans, so we have

to find out what human language is like. In studying perception, we tend to focus on sensors such as cameras that extract useful information from the actual world. (In a world without light, cameras wouldn't be much use.) Moreover, to design vision algorithms that are good at extracting information from camera images, we need to understand the actual world that generates those images. Obtaining the required understanding of scene characteristics, object types, surface markings, and so on is a quite different kind of science from ordinary physics, chemistry, biology, and so on, but it is still science.

In summary, AI is definitely engineering but it would not be especially useful to us if it were not also an empirical science concerned with those aspects of the real world that affect the design of intelligent systems for that world.

1.11 This depends on your definition of "intelligent" and "tell." In one sense computers only do what the programmers command them to do, but in another sense what the programmers consciously tells the computer to do often has very little to do with what the computer actually does. Anyone who has written a program with an ornery bug knows this, as does anyone who has written a successful machine learning program. So in one sense Samuel "told" the computer "learn to play checkers better than I do, and then play that way," but in another sense he told the computer "follow this learning algorithm" and it learned to play. So we're left in the situation where you may or may not consider learning to play checkers to be s sign of intelligence (or you may think that learning to play in the right way requires intelligence, but not in this way), and you may think the intelligence resides in the programmer or in the computer.

1.12 The point of this exercise is to notice the parallel with the previous one. Whatever you decided about whether computers could be intelligent in 1.11, you are committed to making the same conclusion about animals (including humans), *unless* your reasons for deciding whether something is intelligent take into account the mechanism (programming via genes versus programming via a human programmer). Note that Searle makes this appeal to mechanism in his Chinese Room argument (see Chapter 26).

1.13 Again, the choice you make in 1.11 drives your answer to this question.

1.14

a. (ping-pong) A reasonable level of proficiency was achieved by Andersson's robot (Andersson, 1988).

b. (driving in Cairo) No. Although there has been a lot of progress in automated driving, all such systems currently rely on certain relatively constant clues: that the road has shoulders and a center line, that the car ahead will travel a predictable course, that cars will keep to their side of the road, and so on. Some lane changes and turns can be made on clearly marked roads in light to moderate traffic. Driving in downtown Cairo is too unpredictable for any of these to work.

c. (driving in Victorville, California) Yes, to some extent, as demonstrated in DARPA's Urban Challenge. Some of the vehicles managed to negotiate streets, intersections, well-behaved traffic, and well-behaved pedestrians in good visual conditions.

d. (shopping at the market) No. No robot can currently put together the tasks of moving in a crowded environment, using vision to identify a wide variety of objects, and grasping the objects (including squishable vegetables) without damaging them. The component pieces are nearly able to handle the individual tasks, but it would take a major integration effort to put it all together.

e. (shopping on the web) Yes. Software robots are capable of handling such tasks, particularly if the design of the web grocery shopping site does not change radically over time.

f. (bridge) Yes. Programs such as GIB now play at a solid level.

g. (theorem proving) Yes. For example, the proof of Robbins algebra described on page 360.

h. (funny story) No. While some computer-generated prose and poetry is hysterically funny, this is invariably unintentional, except in the case of programs that echo back prose that they have memorized.

i. (legal advice) Yes, in some cases. AI has a long history of research into applications of automated legal reasoning. Two outstanding examples are the Prolog-based expert systems used in the UK to guide members of the public in dealing with the intricacies of the social security and nationality laws. The social security system is said to have saved the UK government approximately $150 million in its first year of operation. However, extension into more complex areas such as contract law awaits a satisfactory encoding of the vast web of common-sense knowledge pertaining to commercial transactions and agreement and business practices.

j. (translation) Yes. In a limited way, this is already being done. See Kay, Gawron and Norvig (1994) and Wahlster (2000) for an overview of the field of speech translation, and some limitations on the current state of the art.

k. (surgery) Yes. Robots are increasingly being used for surgery, although always under the command of a doctor. Robotic skills demonstrated at superhuman levels include drilling holes in bone to insert artificial joints, suturing, and knot-tying. They are not yet capable of planning and carrying out a complex operation autonomously from start to finish.

1.15

The progress made in this contests is a matter of fact, but the impact of that progress is a matter of opinion.

- **DARPA Grand Challenge for Robotic Cars** In 2004 the Grand Challenge was a 240 km race through the Mojave Desert. It clearly stressed the state of the art of autonomous driving, and in fact no competitor finished the race. The best team, CMU, completed only 12 of the 240 km. In 2005 the race featured a 212km course with fewer curves and wider roads than the 2004 race. Five teams finished, with Stanford finishing first, edging out two CMU entries. This was hailed as a great achievement for robotics and for the Challenge format. In 2007 the Urban Challenge put cars in a city setting, where they had to obey traffic laws and avoid other cars. This time CMU edged out Stanford.

The competition appears to have been a good testing ground to put theory into practice, something that the failures of 2004 showed was needed. But it is important that the competition was done at just the right time, when there was theoretical work to consolidate, as demonstrated by the earlier work by Dickmanns (whose VaMP car drove autonomously for 158km in 1995) and by Pomerleau (whose Navlab car drove 5000km across the USA, also in 1995, with the steering controlled autonomously for 98% of the trip, although the brakes and accelerator were controlled by a human driver).

- **International Planning Competition** In 1998, five planners competed: Blackbox, HSP, IPP, SGP, and STAN. The result page (`ftp://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`) stated "all of these planners performed very well, compared to the state of the art a few years ago." Most plans found were 30 or 40 steps, with some over 100 steps. In 2008, the competition had expanded quite a bit: there were more tracks (satisficing vs. optimizing; sequential vs. temporal; static vs. learning). There were about 25 planners, including submissions from the 1998 groups (or their descendants) and new groups. Solutions found were much longer than in 1998. In sum, the field has progressed quite a bit in participation, in breadth, and in power of the planners. In the 1990s it was possible to publish a Planning paper that discussed only a theoretical approach; now it is necessary to show quantitative evidence of the efficacy of an approach. The field is stronger and more mature now, and it seems that the planning competition deserves some of the credit. However, some researchers feel that too much emphasis is placed on the particular classes of problems that appear in the competitions, and not enough on real-world applications.
- **Robocup Robotics Soccer** This competition has proved extremely popular, attracting 407 teams from 43 countries in 2009 (up from 38 teams from 11 countries in 1997). The robotic platform has advanced to a more capable humanoid form, and the strategy and tactics have advanced as well. Although the competition has spurred innovations in distributed control, the winning teams in recent years have relied more on individual ball-handling skills than on advanced teamwork. The competition has served to increase interest and participation in robotics, although it is not clear how well they are advancing towards the goal of defeating a human team by 2050.
- **TREC Information Retrieval Conference** This is one of the oldest competitions, started in 1992. The competitions have served to bring together a community of researchers, have led to a large literature of publications, and have seen progress in participation and in quality of results over the years. In the early years, TREC served its purpose as a place to do evaluations of retrieval algorithms on text collections that were large for the time. However, starting around 2000 TREC became less relevant as the advent of the World Wide Web created a corpus that was available to anyone and was much larger than anything TREC had created, and the development of commercial search engines surpassed academic research.
- **NIST Open Machine Translation Evaluation** This series of evaluations (explicitly not labelled a "competition") has existed since 2001. Since then we have seen great advances in Machine Translation quality as well as in the number of languages covered.

The dominant approach has switched from one based on grammatical rules to one that relies primarily on statistics. The NIST evaluations seem to track these changes well, but don't appear to be driving the changes.

Overall, we see that whatever you measure is bound to increase over time. For most of these competitions, the measurement was a useful one, and the state of the art has progressed. In the case of ICAPS, some planning researchers worry that too much attention has been lavished on the competition itself. In some cases, progress has left the competition behind, as in TREC, where the resources available to commercial search engines outpaced those available to academic researchers. In this case the TREC competition was useful—it helped train many of the people who ended up in commercial search engines—and in no way drew energy away from new ideas.

Solutions for Chapter 2
Intelligent Agents

2.1 This question tests the student's understanding of environments, rational actions, and performance measures. Any sequential environment in which rewards may take time to arrive will work, because then we can arrange for the reward to be "over the horizon." Suppose that in any state there are two action choices, a and b, and consider two cases: the agent is in state s at time T or at time $T-1$. In state s, action a reaches state s' with reward 0, while action b reaches state s again with reward 1; in s' either action gains reward 10. At time $T-1$, it's rational to do a in s, with expected total reward 10 before time is up; but at time T, it's rational to do b with total expected reward 1 because the reward of 10 cannot be obtained before time is up.

Students may also provide common-sense examples from real life: investments whose payoff occurs after the end of life, exams where it doesn't make sense to start the high-value question with too little time left to get the answer, and so on.

The environment state can include a clock, of course; this doesn't change the gist of the answer—now the action will depend on the clock as well as on the non-clock part of the state—but it does mean that the agent can never be in the same state twice.

2.2 Notice that for our simple environmental assumptions we need not worry about quantitative uncertainty.

a. It suffices to show that for all possible actual environments (i.e., all dirt distributions and initial locations), this agent cleans the squares at least as fast as any other agent. This is trivially true when there is no dirt. When there is dirt in the initial location and none in the other location, the world is clean after one step; no agent can do better. When there is no dirt in the initial location but dirt in the other, the world is clean after two steps; no agent can do better. When there is dirt in both locations, the world is clean after three steps; no agent can do better. (Note: in general, the condition stated in the first sentence of this answer is much stricter than necessary for an agent to be rational.)

b. The agent in (a) keeps moving backwards and forwards even after the world is clean. It is better to do $NoOp$ once the world is clean (the chapter says this). Now, since the agent's percept doesn't say whether the other square is clean, it would seem that the agent must have some memory to say whether the other square has already been cleaned. To make this argument rigorous is more difficult—for example, could the agent arrange things so that it would only be in a clean left square when the right square

EXTERNAL MEMORY

was already clean? As a general strategy, an agent *can* use the environment itself as a form of **external memory**—a common technique for humans who use things like appointment calendars and knots in handkerchiefs. In this particular case, however, that is not possible. Consider the reflex actions for $[A, Clean]$ and $[B, Clean]$. If either of these is $NoOp$, then the agent will fail in the case where that is the initial percept but the other square is dirty; hence, neither can be $NoOp$ and therefore the simple reflex agent is doomed to keep moving. In general, the problem with reflex agents is that they have to do the same thing in situations that look the same, even when the situations are actually quite different. In the vacuum world this is a big liability, because every interior square (except home) looks either like a square with dirt or a square without dirt.

c. If we consider asymptotically long lifetimes, then it is clear that learning a map (in some form) confers an advantage because it means that the agent can avoid bumping into walls. It can also learn where dirt is most likely to accumulate and can devise an optimal inspection strategy. The precise details of the exploration method needed to construct a complete map appear in Chapter 4; methods for deriving an optimal inspection/cleanup strategy are in Chapter 21.

2.3

a. *An agent that senses only partial information about the state cannot be perfectly rational.*
False. Perfect rationality refers to the ability to make good decisions given the sensor information received.

b. *There exist task environments in which no pure reflex agent can behave rationally.*
True. A pure reflex agent ignores previous percepts, so cannot obtain an optimal state estimate in a partially observable environment. For example, correspondence chess is played by sending moves; if the other player's move is the current percept, a reflex agent could not keep track of the board state and would have to respond to, say, "a4" in the same way regardless of the position in which it was played.

c. *There exists a task environment in which every agent is rational.*
True. For example, in an environment with a single state, such that all actions have the same reward, it doesn't matter which action is taken. More generally, any environment that is reward-invariant under permutation of the actions will satisfy this property.

d. *The input to an agent program is the same as the input to the agent function.*
False. The agent function, notionally speaking, takes as input the entire percept sequence up to that point, whereas the agent program takes the current percept only.

e. *Every agent function is implementable by some program/machine combination.*
False. For example, the environment may contain Turing machines and input tapes and the agent's job is to solve the halting problem; there is an agent *function* that specifies the right answers, but no agent program can implement it. Another example would be an agent function that requires solving intractable problem instances of arbitrary size in constant time.

f. *Suppose an agent selects its action uniformly at random from the set of possible actions. There exists a deterministic task environment in which this agent is rational.*
True. This is a special case of (c); if it doesn't matter which action you take, selecting randomly is rational.

g. *It is possible for a given agent to be perfectly rational in two distinct task environments.*
True. For example, we can arbitrarily modify the parts of the environment that are unreachable by any optimal policy as long as they stay unreachable.

h. *Every agent is rational in an unobservable environment.*
False. Some actions are stupid—and the agent may know this if it has a model of the environment—even if one cannot perceive the environment state.

i. *A perfectly rational poker-playing agent never loses.*
False. Unless it draws the perfect hand, the agent can always lose if an opponent has better cards. This can happen for game after game. The correct statement is that the agent's expected winnings are nonnegative.

2.4 Many of these can actually be argued either way, depending on the level of detail and abstraction.

A. Partially observable, stochastic, sequential, dynamic, continuous, multi-agent.

B. Partially observable, stochastic, sequential, dynamic, continuous, single agent (unless there are alien life forms that are usefully modeled as agents).

C. Partially observable, deterministic, sequential, static, discrete, single agent. This can be multi-agent and dynamic if we buy books via auction, or dynamic if we purchase on a long enough scale that book offers change.

D. Fully observable, stochastic, episodic (every point is separate), dynamic, continuous, multi-agent.

E. Fully observable, stochastic, episodic, dynamic, continuous, single agent.

F. Fully observable, stochastic, sequential, static, continuous, single agent.

G. Fully observable, deterministic, sequential, static, continuous, single agent.

H. Fully observable, strategic, sequential, static, discrete, multi-agent.

2.5 The following are just some of the many possible definitions that can be written:

- *Agent*: an entity that perceives and acts; or, one that *can be viewed* as perceiving and acting. Essentially any object qualifies; the key point is the way the object implements an agent function. (Note: some authors restrict the term to *programs* that operate *on behalf of* a human, or to programs that can cause some or all of their code to run on other machines on a network, as in **mobile agents**.)

MOBILE AGENT

- *Agent function*: a function that specifies the agent's action in response to every possible percept sequence.
- *Agent program*: that program which, combined with a machine architecture, implements an agent function. In our simple designs, the program takes a new percept on each invocation and returns an action.

- *Rationality*: a property of agents that choose actions that maximize their expected utility, given the percepts to date.
- *Autonomy*: a property of agents whose behavior is determined by their own experience rather than solely by their initial programming.
- *Reflex agent*: an agent whose action depends only on the current percept.
- *Model-based agent*: an agent whose action is derived directly from an internal model of the current world state that is updated over time.
- *Goal-based agent*: an agent that selects actions that it believes will achieve explicitly represented goals.
- *Utility-based agent*: an agent that selects actions that it believes will maximize the expected utility of the outcome state.
- *Learning agent*: an agent whose behavior improves over time based on its experience.

2.6 Although these questions are very simple, they hint at some very fundamental issues. Our answers are for the simple agent designs for *static* environments where nothing happens while the agent is deliberating; the issues get even more interesting for dynamic environments.

a. Yes; take any agent program and insert null statements that do not affect the output.

b. Yes; the agent function might specify that the agent print $true$ when the percept is a Turing machine program that halts, and $false$ otherwise. (Note: in dynamic environments, for machines of less than infinite speed, the rational agent function may not be implementable; e.g., the agent function that always plays a winning move, if any, in a game of chess.)

c. Yes; the agent's behavior is fixed by the architecture and program.

d. There are 2^n agent programs, although many of these will not run at all. (Note: Any given program can devote at most n bits to storage, so its internal state can distinguish among only 2^n past histories. Because the agent function specifies actions based on percept histories, there will be many agent functions that cannot be implemented because of lack of memory in the machine.)

e. It depends on the program and the environment. If the environment is dynamic, speeding up the machine may mean choosing different (perhaps better) actions and/or acting sooner. If the environment is static and the program pays no attention to the passage of elapsed time, the agent function is unchanged.

2.7

The design of goal- and utility-based agents depends on the structure of the task environment. The simplest such agents, for example those in chapters 3 and 10, compute the agent's entire future sequence of actions in advance before acting at all. This strategy works for static and deterministic environments which are either fully-known or unobservable

For fully-observable and fully-known static environments a policy can be computed in advance which gives the action to by taken in any given state.

```
function GOAL-BASED-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
              goal, a description of the desired goal state
              plan, a sequence of actions to take, initially empty
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, model)
  if GOAL-ACHIEVED(state,goal) then return a null action
  if plan is empty then
plan ← PLAN(state,goal,model)
  action ← FIRST(plan)
  plan ← REST(plan)
  return action
```

Figure S2.1 A goal-based agent.

For partially-observable environments the agent can compute a conditional plan, which specifies the sequence of actions to take as a function of the agent's perception. In the extreme, a conditional plan gives the agent's response to every contingency, and so it is a representation of the entire agent function.

In all cases it may be either intractable or too expensive to compute everything out in advance. Instead of a conditional plan, it may be better to compute a single sequence of actions which is likely to reach the goal, then monitor the environment to check whether the plan is succeeding, repairing or replanning if it is not. It may be even better to compute only the start of this plan before taking the first action, continuing to plan at later time steps.

Pseudocode for simple goal-based agent is given in Figure S2.1. GOAL-ACHIEVED tests to see whether the current state satisfies the goal or not, doing nothing if it does. PLAN computes a sequence of actions to take to achieve the goal. This might return only a prefix of the full plan, the rest will be computed after the prefix is executed. This agent will act to maintain the goal: if at any point the goal is not satisfied it will (eventually) replan to achieve the goal again.

At this level of abstraction the utility-based agent is not much different than the goal-based agent, except that action may be continuously required (there is not necessarily a point where the utility function is "satisfied"). Pseudocode is given in Figure S2.2.

2.8 The file `"agents/environments/vacuum.lisp"` in the code repository implements the vacuum-cleaner environment. Students can easily extend it to generate different shaped rooms, obstacles, and so on.

2.9 A reflex agent program implementing the rational agent function described in the chapter is as follows:

```
(defun reflex-rational-vacuum-agent (percept)
  (destructuring-bind (location status) percept
```

function UTILITY-BASED-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
    $utility - function$, a description of the agent's utility function
    $plan$, a sequence of actions to take, initially empty
    $action$, the most recent action, initially none

$state \leftarrow$ UPDATE-STATE($state$, $action$, $percept$, $model$)
if $plan$ is empty **then**
$plan \leftarrow$ PLAN($state$,$utility - function$,$model$)
$action \leftarrow$ FIRST($plan$)
$plan \leftarrow$ REST($plan$)
return $action$

Figure S2.2 A utility-based agent.

```
(cond ((eq status 'Dirty) 'Suck)
      ((eq location 'A) 'Right)
      (t 'Left))))
```

For states 1, 3, 5, 7 in Figure 4.9, the performance measures are 1996, 1999, 1998, 2000 respectively.

2.10

a. No; see answer to 2.4(b).

b. See answer to 2.4(b).

c. In this case, a simple reflex agent can be perfectly rational. The agent can consist of a table with eight entries, indexed by percept, that specifies an action to take for each possible state. After the agent acts, the world is updated and the next percept will tell the agent what to do next. For larger environments, constructing a table is infeasible. Instead, the agent could run one of the optimal search algorithms in Chapters 3 and 4 and execute the first step of the solution sequence. Again, no internal state is *required*, but it would help to be able to store the solution sequence instead of recomputing it for each new percept.

2.11

a. Because the agent does not know the geography and perceives only location and local dirt, and cannot remember what just happened, it will get stuck forever against a wall when it tries to move in a direction that is blocked—that is, unless it randomizes.

b. One possible design cleans up dirt and otherwise moves randomly:

```
(defun randomized-reflex-vacuum-agent (percept)
  (destructuring-bind (location status) percept
    (cond ((eq status 'Dirty) 'Suck)
          (t (random-element '(Left Right Up Down))))))
```

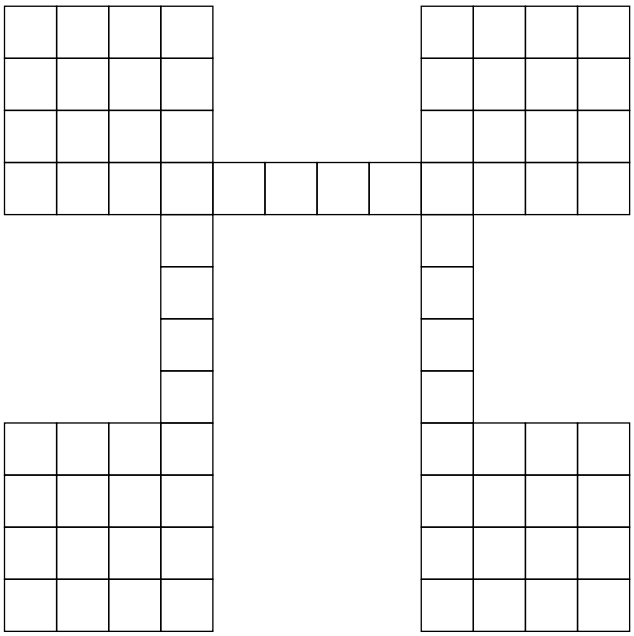


Figure S2.3 An environment in which random motion will take a long time to cover all the squares.

This is fairly close to what the Roomba^{TM} vacuum cleaner does (although the Roomba has a bump sensor and randomizes only when it hits an obstacle). It works reasonably well in nice, compact environments. In maze-like environments or environments with small connecting passages, it can take a very long time to cover all the squares.

c. An example is shown in Figure S2.3. Students may also wish to measure clean-up time for linear or square environments of different sizes, and compare those to the efficient online search algorithms described in Chapter 4.

d. A reflex agent with state can build a map (see Chapter 4 for details). An online depth-first exploration will reach every state in time linear in the size of the environment; therefore, the agent can do much better than the simple reflex agent.

The question of rational behavior in unknown environments is a complex one but it is worth encouraging students to think about it. We need to have some notion of the prior probability distribution over the class of environments; call this the initial **belief state**. Any action yields a new percept that can be used to update this distribution, moving the agent to a new belief state. Once the environment is completely explored, the belief state collapses to a single possible environment. Therefore, the problem of optimal exploration can be viewed as a search for an optimal strategy in the space of possible belief states. This is a well-defined, if horrendously intractable, problem. Chapter 21 discusses some cases where optimal exploration is possible. Another concrete example of exploration is the Minesweeper computer game (see Exercise 7.22). For very small Minesweeper environments, optimal exploration is feasible although the belief state

update is nontrivial to explain.

2.12 The problem appears at first to be very similar; the main difference is that instead of using the location percept to build the map, the agent has to "invent" its own locations (which, after all, are just nodes in a data structure representing the state space graph). When a bump is detected, the agent assumes it remains in the same location and can add a wall to its map. For grid environments, the agent can keep track of its (x, y) location and so can tell when it has returned to an old state. In the general case, however, there is no simple way to tell if a state is new or old.

2.13

a. For a reflex agent, this presents no *additional* challenge, because the agent will continue to $Suck$ as long as the current location remains dirty. For an agent that constructs a sequential plan, every $Suck$ action would need to be replaced by "$Suck$ until clean." If the dirt sensor can be wrong on each step, then the agent might want to wait for a few steps to get a more reliable measurement before deciding whether to $Suck$ or move on to a new square. Obviously, there is a trade-off because waiting too long means that dirt remains on the floor (incurring a penalty), but acting immediately risks either dirtying a clean square or ignoring a dirty square (if the sensor is wrong). A rational agent must also continue touring and checking the squares in case it missed one on a previous tour (because of bad sensor readings). it is not immediately obvious how the waiting time at each square should change with each new tour. These issues can be clarified by experimentation, which may suggest a general trend that can be verified mathematically. This problem is a partially observable Markov decision process—see Chapter 17. Such problems are hard in general, but some special cases may yield to careful analysis.

b. In this case, the agent must keep touring the squares indefinitely. The probability that a square is dirty increases monotonically with the time since it was last cleaned, so the rational strategy is, roughly speaking, to repeatedly execute the shortest possible tour of all squares. (We say "roughly speaking" because there are complications caused by the fact that the shortest tour may visit some squares twice, depending on the geography.) This problem is also a partially observable Markov decision process.

Solutions for Chapter 3
Solving Problems by Searching

3.1 In goal formulation, we decide which aspects of the world we are interested in, and which can be ignored or abstracted away. Then in problem formulation we decide how to manipulate the important aspects (and ignore the others). If we did problem formulation first we would not know what to include and what to leave out. That said, it can happen that there is a cycle of iterations between goal formulation, problem formulation, and problem solving until one arrives at a sufficiently useful and efficient solution.

3.2

a. We'll define the coordinate system so that the center of the maze is at $(0, 0)$, and the maze itself is a square from $(-1, -1)$ to $(1, 1)$.

Initial state: robot at coordinate $(0, 0)$, facing North.
Goal test: either $|x| > 1$ or $|y| > 1$ where (x, y) is the current location.
Successor function: move forwards any distance d; change direction robot it facing.
Cost function: total distance moved.

The state space is infinitely large, since the robot's position is continuous.

b. The state will record the intersection the robot is currently at, along with the direction it's facing. At the end of each corridor leaving the maze we will have an exit node. We'll assume some node corresponds to the center of the maze.

Initial state: at the center of the maze facing North.
Goal test: at an exit node.
Successor function: move to the next intersection in front of us, if there is one; turn to face a new direction.
Cost function: total distance moved.

There are $4n$ states, where n is the number of intersections.

c. Initial state: at the center of the maze.
Goal test: at an exit node.
Successor function: move to next intersection to the North, South, East, or West.
Cost function: total distance moved.

We no longer need to keep track of the robot's orientation since it is irrelevant to

predicting the outcome of our actions, and not part of the goal test. The motor system that executes this plan will need to keep track of the robot's current orientation, to know when to rotate the robot.

d. State abstractions:

(i) Ignoring the height of the robot off the ground, whether it is tilted off the vertical.
(ii) The robot can face in only four directions.
(iii) Other parts of the world ignored: possibility of other robots in the maze, the weather in the Caribbean.

Action abstractions:

(i) We assumed all positions we safely accessible: the robot couldn't get stuck or damaged.
(ii) The robot can move as far as it wants, without having to recharge its batteries.
(iii) Simplified movement system: moving forwards a certain distance, rather than controlled each individual motor and watching the sensors to detect collisions.

3.3

a. State space: States are all possible city pairs (i, j). The map is *not* the state space.

Successor function: The successors of (i, j) are all pairs (x, y) such that $Adjacent(x, i)$ and $Adjacent(y, j)$.

Goal: Be at (i, i) for some i.

Step cost function: The cost to go from (i, j) to (x, y) is $max(d(i, x), d(j, y))$.

b. In the best case, the friends head straight for each other in steps of equal size, reducing their separation by twice the time cost on each step. Hence (iii) is admissible.

c. Yes: e.g., a map with two nodes connected by one link. The two friends will swap places forever. The same will happen on any chain if they start an odd number of steps apart. (One can see this best on the graph that represents the state space, which has two disjoint sets of nodes.) The same even holds for a grid of any size or shape, because every move changes the Manhattan distance between the two friends by 0 or 2.

d. Yes: take any of the unsolvable maps from part (c) and add a self-loop to any one of the nodes. If the friends start an odd number of steps apart, a move in which one of the friends takes the self-loop changes the distance by 1, rendering the problem solvable. If the self-loop is not taken, the argument from (c) applies and no solution is possible.

3.4 From http://www.cut-the-knot.com/pythagoras/fifteen.shtml, this proof applies to the fifteen puzzle, but the same argument works for the eight puzzle:

Definition: The goal state has the numbers in a certain order, which we will measure as starting at the upper left corner, then proceeding left to right, and when we reach the end of a row, going down to the leftmost square in the row below. For any other configuration besides the goal, whenever a tile with a greater number on it precedes a tile with a smaller number, the two tiles are said to be **inverted**.

Proposition: For a given puzzle configuration, let N denote the sum of the total number of inversions and the row number of the empty square. Then $(N mod 2)$ is invariant under any

legal move. In other words, after a legal move an odd N remains odd whereas an even N remains even. Therefore the goal state in Figure 3.4, with no inversions and empty square in the first row, has $N = 1$, and can only be reached from starting states with odd N, not from starting states with even N.

Proof: First of all, sliding a tile horizontally changes neither the total number of inversions nor the row number of the empty square. Therefore let us consider sliding a tile vertically.

Let's assume, for example, that the tile A is located directly over the empty square. Sliding it down changes the parity of the row number of the empty square. Now consider the total number of inversions. The move only affects relative positions of tiles A, B, C, and D. If none of the B, C, D caused an inversion relative to A (i.e., all three are larger than A) then after sliding one gets three (an odd number) of additional inversions. If one of the three is smaller than A, then before the move B, C, and D contributed a single inversion (relative to A) whereas after the move they'll be contributing two inversions - a change of 1, also an odd number. Two additional cases obviously lead to the same result. Thus the change in the sum N is always even. This is precisely what we have set out to show.

So before we solve a puzzle, we should compute the N value of the start and goal state and make sure they have the same parity, otherwise no solution is possible.

3.5 The formulation puts one queen per column, with a new queen placed only in a square that is not attacked by any other queen. To simplify matters, we'll first consider the n–rooks problem. The first rook can be placed in any square in column 1 (n choices), the second in any square in column 2 except the same row that as the rook in column 1 ($n-1$ choices), and so on. This gives $n!$ elements of the search space.

For n queens, notice that a queen attacks at most three squares in any given column, so in column 2 there are at least $(n-3)$ choices, in column at least $(n-6)$ choices, and so on. Thus the state space size $S \geq n \cdot (n-3) \cdot (n-6) \cdots$. Hence we have

$$\begin{aligned} S^3 &\geq n \cdot n \cdot n \cdot (n-3) \cdot (n-3) \cdot (n-3) \cdot (n-6) \cdot (n-6) \cdot (n-6) \cdot \cdots \\ &\geq n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot (n-4) \cdot (n-5) \cdot (n-6) \cdot (n-7) \cdot (n-8) \cdot \cdots \\ &= n! \end{aligned}$$

or $S \geq \sqrt[3]{n!}$.

3.6

a. Initial state: No regions colored.
Goal test: All regions colored, and no two adjacent regions have the same color.
Successor function: Assign a color to a region.
Cost function: Number of assignments.

b. Initial state: As described in the text.
Goal test: Monkey has bananas.
Successor function: Hop on crate; Hop off crate; Push crate from one spot to another; Walk from one spot to another; grab bananas (if standing on crate).
Cost function: Number of actions.

c. Initial state: considering all input records.
Goal test: considering a single record, and it gives "illegal input" message.
Successor function: run again on the first half of the records; run again on the second half of the records.
Cost function: Number of runs.
Note: This is a **contingency problem**; you need to see whether a run gives an error message or not to decide what to do next.

d. Initial state: jugs have values $[0, 0, 0]$.
Successor function: given values $[x, y, z]$, generate $[12, y, z]$, $[x, 8, z]$, $[x, y, 3]$ (by filling); $[0, y, z]$, $[x, 0, z]$, $[x, y, 0]$ (by emptying); or for any two jugs with current values x and y, pour y into x; this changes the jug with x to the minimum of $x + y$ and the capacity of the jug, and decrements the jug with y by by the amount gained by the first jug.
Cost function: Number of actions.

3.7

a. If we consider all (x, y) points, then there are an infinite number of states, and of paths.

b. (For this problem, we consider the start and goal points to be vertices.) The shortest distance between two points is a straight line, and if it is not possible to travel in a straight line because some obstacle is in the way, then the next shortest distance is a sequence of line segments, end-to-end, that deviate from the straight line by as little as possible. So the first segment of this sequence must go from the start point to a tangent point on an obstacle – any path that gave the obstacle a wider girth would be longer. Because the obstacles are polygonal, the tangent points must be at vertices of the obstacles, and hence the entire path must go from vertex to vertex. So now the state space is the set of vertices, of which there are 35 in Figure 3.31.

c. Code not shown.

d. Implementations and analysis not shown.

3.8

a. Any path, no matter how bad it appears, might lead to an arbitrarily large reward (negative cost). Therefore, one would need to exhaust all possible paths to be sure of finding the best one.

b. Suppose the greatest possible reward is c. Then if we also know the maximum depth of the state space (e.g. when the state space is a tree), then any path with d levels remaining can be improved by at most cd, so any paths worse than cd less than the best path can be pruned. For state spaces with loops, this guarantee doesn't help, because it is possible to go around a loop any number of times, picking up c reward each time.

c. The agent should plan to go around this loop forever (unless it can find another loop with even better reward).

d. The value of a scenic loop is lessened each time one revisits it; a novel scenic sight is a great reward, but seeing the same one for the tenth time in an hour is tedious, not

rewarding. To accommodate this, we would have to expand the state space to include a memory—a state is now represented not just by the current location, but by a current location and a bag of already-visited locations. The reward for visiting a new location is now a (diminishing) function of the number of times it has been seen before.

e. Real domains with looping behavior include eating junk food and going to class.

3.9

a. Here is one possible representation: A state is a six-tuple of integers listing the number of missionaries, cannibals, and boats on the first side, and then the second side of the river. The goal is a state with 3 missionaries and 3 cannibals on the second side. The cost function is one per action, and the successors of a state are all the states that move 1 or 2 people and 1 boat from one side to another.

b. The search space is small, so any optimal algorithm works. For an example, see the file `"search/domains/cannibals.lisp"`. It suffices to eliminate moves that circle back to the state just visited. From all but the first and last states, there is only one other choice.

c. It is not obvious that almost all moves are either illegal or revert to the previous state. There is a feeling of a large branching factor, and no clear way to proceed.

3.10 A **state** is a situation that an agent can find itself in. We distinguish two types of states: world states (the actual concrete situations in the real world) and representational states (the abstract descriptions of the real world that are used by the agent in deliberating about what to do).

A **state space** is a graph whose nodes are the set of all states, and whose links are actions that transform one state into another.

A **search tree** is a tree (a graph with no undirected loops) in which the root node is the start state and the set of children for each node consists of the states reachable by taking any action.

A **search node** is a node in the search tree.

A **goal** is a state that the agent is trying to reach.

An **action** is something that the agent can choose to do.

A **successor function** described the agent's options: given a state, it returns a set of (action, state) pairs, where each state is the state reachable by taking the action.

The **branching factor** in a search tree is the number of actions available to the agent.

3.11 A world state is how reality is or could be. In one world state we're in Arad, in another we're in Bucharest. The world state also includes which street we're on, what's currently on the radio, and the price of tea in China. A state description is an agent's internal description of a world state. Examples are $In(Arad)$ and $In(Bucharest)$. These descriptions are necessarily approximate, recording only some aspect of the state.

We need to distinguish between world states and state descriptions because state description are lossy abstractions of the world state, because the agent could be mistaken about

how the world is, because the agent might want to imagine things that aren't true but it could make true, and because the agent cares about the world not its internal representation of it.

Search nodes are generated during search, representing a state the search process knows how to reach. They contain additional information aside from the state description, such as the sequence of actions used to reach this state. This distinction is useful because we may generate different search nodes which have the same state, and because search nodes contain more information than a state representation.

3.12 The state space is a tree of depth one, with all states successors of the initial state. There is no distinction between depth-first search and breadth-first search on such a tree. If the sequence length is unbounded the root node will have infinitely many successors, so only algorithms which test for goal nodes as we generate successors can work.

What happens next depends on how the composite actions are sorted. If there is no particular ordering, then a random but systematic search of potential solutions occurs. If they are sorted by dictionary order, then this implements depth-first search. If they are sorted by length first, then dictionary ordering, this implements breadth-first search.

A significant disadvantage of collapsing the search space like this is if we discover that a plan starting with the action "unplug your battery" can't be a solution, there is no easy way to ignore all other composite actions that start with this action. This is a problem in particular for informed search algorithms.

Discarding sequence structure is not a particularly practical approach to search.

3.13

The graph separation property states that "every path from the initial state to an unexplored state has to pass through a state in the frontier."

At the start of the search, the frontier holds the initial state; hence, trivially, every path from the initial state to an unexplored state includes a node in the frontier (the initial state itself).

Now, we assume that the property holds at the beginning of an arbitrary iteration of the GRAPH-SEARCH algorithm in Figure 3.7. We assume that the iteration completes, i.e., the frontier is not empty and the selected leaf node n is not a goal state. At the end of the iteration, n has been removed from the frontier and its successors (if not already explored or in the frontier) placed in the frontier. Consider any path from the initial state to an unexplored state; by the induction hypothesis such a path (at the beginning of the iteration) includes at least one frontier node; except when n is the only such node, the separation property automatically holds. Hence, we focus on paths passing through n (and no other frontier node). By definition, the next node n' along the path from n must be a successor of n that (by the preceding sentence) is already not in the frontier. Furthermore, n' cannot be in the explored set, since by assumption there is a path from n' to an unexplored node not passing through the frontier, which would violate the separation property as every explored node is connected to the initial state by explored nodes (see lemma below for proof this is always possible). Hence, n' is not in the explored set, hence it will be added to the frontier; then the path will include a frontier node and the separation property is restored.

The property is violated by algorithms that move nodes from the frontier into the ex-

plored set before all of their successors have been generated, as well as by those that fail to add some of the successors to the frontier. Note that it is not necessary to generate *all* successors of a node at once before expanding another node, as long as partially expanded nodes remain in the frontier.

Lemma: Every explored node is connected to the initial state by a path of explored nodes.

Proof: This is true initially, since the initial state is connected to itself. Since we never remove nodes from the explored region, we only need to check new nodes we add to the explored list on an expansion. Let n be such a new explored node. This is previously on the frontier, so it is a neighbor of a node n' previously explored (i.e., its parent). n' is, by hypothesis is connected to the initial state by a path of explored nodes. This path with n appended is a path of explored nodes connecting n' to the initial state.

3.14

a. *False*: a lucky DFS might expand exactly d nodes to reach the goal. A* largely dominates any graph-search algorithm that is *guaranteed to find optimal solutions*.

b. *True*: $h(n) = 0$ is always an admissible heuristic, since costs are nonnegative.

c. *True*: A* search is often used in robotics; the space can be discretized or skeletonized.

d. *True*: depth of the solution matters for breadth-first search, not cost.

e. *False*: a rook can move across the board in move one, although the Manhattan distance from start to finish is 8.

3.15

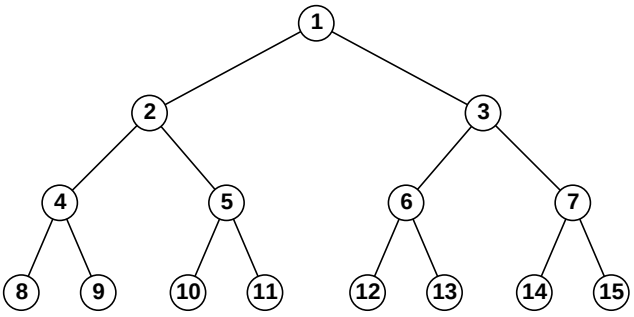


Figure S3.1 The state space for the problem defined in Ex. 3.15.

a. See Figure S3.1.

b. Breadth-first: 1 2 3 4 5 6 7 8 9 10 11
Depth-limited: 1 2 4 8 9 5 10 11
Iterative deepening: 1; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11

c. Bidirectional search is very useful, because the only successor of n in the reverse direction is $\lfloor (n/2) \rfloor$. This helps focus the search. The branching factor is 2 in the forward direction; 1 in the reverse direction.

d. Yes; start at the goal, and apply the single reverse successor action until you reach 1.

e. The solution can be read off the binary numeral for the goal number. Write the goal number in binary. Since we can only reach positive integers, this binary expansion beings with a 1. From most- to least- significant bit, skipping the initial 1, go Left to the node $2n$ if this bit is 0 and go Right to node $2n + 1$ if it is 1. For example, suppose the goal is 11, which is 1011 in binary. The solution is therefore Left, Right, Right.

3.16

a. **Initial state**: one arbitrarily selected piece (say a straight piece).
Successor function: for any open peg, add any piece type from remaining types. (You can add to open holes as well, but that isn't necessary as all complete tracks can be made by adding to pegs.) For a curved piece, add *in either orientation*; for a fork, add *in either orientation* and (if there are two holes) connecting *at either hole*. It's a good idea to disallow any overlapping configuration, as this terminates hopeless configurations early. (Note: there is no need to consider open holes, because in any solution these will be filled by pieces added to open pegs.)
Goal test: all pieces used in a single connected track, no open pegs or holes, no overlapping tracks.
Step cost: one per piece (actually, doesn't really matter).

b. All solutions are at the same depth, so depth-first search would be appropriate. (One could also use depth-limited search with limit $n - 1$, but strictly speaking it's not necessary to do the work of checking the limit because states at depth $n - 1$ have no successors.) The space is very large, so uniform-cost and breadth-first would fail, and iterative deepening simply does unnecessary extra work. There are many repeated states, so it might be good to use a closed list.

c. A solution has no open pegs or holes, so every peg is in a hole, so there must be equal numbers of pegs and holes. Removing a fork violates this property. There are two other "proofs" that are acceptable: 1) a similar argument to the effect that there must be an even number of "ends"; 2) each fork creates two tracks, and only a fork can rejoin those tracks into one, so if a fork is missing it won't work. The argument using pegs and holes is actually more general, because it also applies to the case of a three-way fork that has one hole and three pegs or one peg and three holes. The "ends" argument fails here, as does the fork/rejoin argument (which is a bit handwavy anyway).

d. The maximum possible number of open pegs is 3 (starts at 1, adding a two-peg fork increases it by one). Pretending each piece is unique, any piece can be added to a peg, giving at most $12 + (2 \cdot 16) + (2 \cdot 2) + (2 \cdot 2 \cdot 2) = 56$ choices per peg. The total depth is 32 (there are 32 pieces), so an upper bound is $168^{32}/(12! \cdot 16! \cdot 2! \cdot 2!)$ where the factorials deal with permutations of identical pieces. One could do a more refined analysis to handle the fact that the branching factor shrinks as we go down the tree, but it is not pretty.

3.17 **a.** The algorithm expands nodes in order of increasing path cost; therefore the first goal it encounters will be the goal with the cheapest cost.

b. It will be the same as iterative deepening, d iterations, in which $O(b^d)$ nodes are generated.

c. d/ϵ

d. Implementation not shown.

3.18 Consider a domain in which every state has a single successor, and there is a single goal at depth n. Then depth-first search will find the goal in n steps, whereas iterative deepening search will take $1 + 2 + 3 + \cdots + n = O(n^2)$ steps.

3.19 As an ordinary person (or agent) browsing the web, we can only generate the successors of a page by visiting it. We can then do breadth-first search, or perhaps best-search search where the heuristic is some function of the number of words in common between the start and goal pages; this may help keep the links on target. Search engines keep the complete graph of the web, and may provide the user access to all (or at least some) of the pages that link to a page; this would allow us to do bidirectional search.

3.20 Code not shown, but a good start is in the code repository. Clearly, graph search must be used—this is a classic grid world with many alternate paths to each state. Students will quickly find that computing the optimal solution sequence is prohibitively expensive for moderately large worlds, because the state space for an $n \times n$ world has $n^2 \cdot 2^n$ states. The completion time of the random agent grows less than exponentially in n, so for any reasonable exchange rate between search cost ad path cost the random agent will eventually win.

3.21

a. When all step costs are equal, $g(n) \propto depth(n)$, so uniform-cost search reproduces breadth-first search.

b. Breadth-first search is best-first search with $f(n) = depth(n)$; depth-first search is best-first search with $f(n) = -depth(n)$; uniform-cost search is best-first search with $f(n) = g(n)$.

c. Uniform-cost search is A* search with $h(n) = 0$.

3.22 The student should find that on the 8-puzzle, RBFS expands more nodes (because it does not detect repeated states) but has lower cost per node because it does not need to maintain a queue. The number of RBFS node re-expansions is not too high because the presence of many tied values means that the best path changes seldom. When the heuristic is slightly perturbed, this advantage disappears and RBFS's performance is much worse.

For TSP, the state space is a tree, so repeated states are not an issue. On the other hand, the heuristic is real-valued and there are essentially no tied values, so RBFS incurs a heavy penalty for frequent re-expansions.

3.23 The sequence of queues is as follows:

L[0+244=244]

M[70+241=311], T[111+329=440]

L[140+244=384], D[145+242=387], T[111+329=440]

D[145+242=387], T[111+329=440], M[210+241=451], T[251+329=580]

C[265+160=425], T[111+329=440], M[210+241=451], M[220+241=461], T[251+329=580]
T[111+329=440], M[210+241=451], M[220+241=461], P[403+100=503], T[251+329=580], R[411+193=604], D[385+242=627]
M[210+241=451], M[220+241=461], L[222+244=466], P[403+100=503], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627]
M[220+241=461], L[222+244=466], P[403+100=503], L[280+244=524], D[285+242=527], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627]
L[222+244=466], P[403+100=503], L[280+244=524], D[285+242=527], L[290+244=534], D[295+242=537], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627]
P[403+100=503], L[280+244=524], D[285+242=527], M[292+241=533], L[290+244=534], D[295+242=537], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627], T[333+329=662]
B[504+0=504], L[280+244=524], D[285+242=527], M[292+241=533], L[290+244=534], D[295+242=537], T[251+329=580], A[229+366=595], R[411+193=604], D[385+242=627], T[333+329=662], R[500+193=693], C[541+160=701]

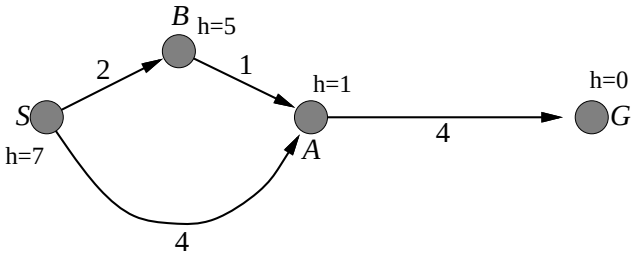


Figure S3.2 A graph with an inconsistent heuristic on which GRAPH-SEARCH fails to return the optimal solution. The successors of S are A with $f = 5$ and B with $f = 7$. A is expanded first, so the path via B will be discarded because A will already be in the closed list.

3.24 See Figure S3.2.

3.25 It is complete whenever $0 \leq w < 2$. $w = 0$ gives $f(n) = 2g(n)$. This behaves exactly like uniform-cost search—the factor of two makes no difference in the *ordering* of the nodes. $w = 1$ gives A* search. $w = 2$ gives $f(n) = 2h(n)$, i.e., greedy best-first search. We also have

$$f(n) = (2 - w)[g(n) + \frac{w}{2 - w}h(n)]$$

which behaves exactly like A* search with a heuristic $\frac{w}{2-w}h(n)$. For $w \leq 1$, this is always less than $h(n)$ and hence admissible, provided $h(n)$ is itself admissible.

3.26

a. The branching factor is 4 (number of neighbors of each location).

b. The states at depth k form a square rotated at 45 degrees to the grid. Obviously there are a linear number of states along the boundary of the square, so the answer is $4k$.

c. Without repeated state checking, BFS expends exponentially many nodes: counting precisely, we get $((4^{x+y+1} - 1)/3) - 1$.

d. There are quadratically many states within the square for depth $x + y$, so the answer is $2(x + y)(x + y + 1) - 1$.

e. True; this is the Manhattan distance metric.

f. False; all nodes in the rectangle defined by $(0, 0)$ and (x, y) are candidates for the optimal path, and there are quadratically many of them, all of which may be expended in the worst case.

g. True; removing links may induce detours, which require more steps, so h is an underestimate.

h. False; nonlocal links can reduce the actual path length below the Manhattan distance.

3.27

a. n^{2n}. There are n vehicles in n^2 locations, so roughly (ignoring the one-per-square constraint) $(n^2)^n = n^{2n}$ states.

b. 5^n.

c. Manhattan distance, i.e., $|(n - i + 1) - x_i| + |n - y_i|$. This is exact for a lone vehicle.

d. Only (iii) $\min\{h_1, \ldots, h_n\}$. The explanation is nontrivial as it requires two observations. First, let the *work* W in a given solution be the total *distance* moved by all vehicles over their joint trajectories; that is, for each vehicle, add the lengths of all the steps taken. We have $W \geq \sum_i h_i \geq\geq n \cdot min\{h_1, ..., h_n\}$. Second, the total work we can get done per step is $\leq n$. (Note that for every car that jumps 2, another car has to stay put (move 0), so the total work per step is bounded by n.) Hence, completing all the work requires at least $n \cdot min\{h_1, ..., h_n\}/n = min\{h_1, ..., h_n\}$ steps.

3.28 The heuristic $h = h_1 + h_2$ (adding misplaced tiles and Manhattan distance) sometimes overestimates. Now, suppose $h(n) \leq h^*(n) + c$ (as given) and let G_2 be a goal that is suboptimal by more than c, i.e., $g(G_2) > C^* + c$. Now consider any node n on a path to an optimal goal. We have

$$\begin{aligned} f(n) &= g(n) + h(n) \\ &\leq g(n) + h^*(n) + c \\ &\leq C^* + c \\ &\leq g(G_2) \end{aligned}$$

so G_2 will never be expanded before an optimal goal is expanded.

3.29 A heuristic is consistent iff, for every node n and every successor n' of n generated by any action a,

$$h(n) \leq c(n, a, n') + h(n')$$

One simple proof is by induction on the number k of nodes on the shortest path to any goal from n. For $k = 1$, let n' be the goal node; then $h(n) \leq c(n, a, n')$. For the inductive

case, assume n' is on the shortest path k steps from the goal and that $h(n')$ is admissible by hypothesis; then

$$h(n) \leq c(n, a, n') + h(n') \leq c(n, a, n') + h^*(n') = h^*(n)$$

so $h(n)$ at $k + 1$ steps from the goal is also admissible.

3.30 This exercise reiterates a small portion of the classic work of Held and Karp (1970).

a. The TSP problem is to find a minimal (total length) path through the cities that forms a closed loop. MST is a relaxed version of that because it asks for a minimal (total length) graph that need not be a closed loop—it can be any fully-connected graph. As a heuristic, MST is admissible—it is always shorter than or equal to a closed loop.

b. The straight-line distance back to the start city is a rather weak heuristic—it vastly underestimates when there are many cities. In the later stage of a search when there are only a few cities left it is not so bad. To say that MST dominates straight-line distance is to say that MST always gives a higher value. This is obviously true because a MST that includes the goal node and the current node must either be the straight line between them, or it must include two or more lines that add up to more. (This all assumes the triangle inequality.)

c. See `"search/domains/tsp.lisp"` for a start at this. The file includes a heuristic based on connecting each unvisited city to its nearest neighbor, a close relative to the MST approach.

d. See (Cormen *et al.*, 1990, p.505) for an algorithm that runs in $O(E \log E)$ time, where E is the number of edges. The code repository currently contains a somewhat less efficient algorithm.

3.31 The misplaced-tiles heuristic is exact for the problem where a tile can move from square A to square B. As this is a relaxation of the condition that a tile can move from square A to square B if B is blank, Gaschnig's heuristic cannot be less than the misplaced-tiles heuristic. As it is also admissible (being exact for a relaxation of the original problem), Gaschnig's heuristic is therefore more accurate.

If we permute two adjacent tiles in the goal state, we have a state where misplaced-tiles and Manhattan both return 2, but Gaschnig's heuristic returns 3.

To compute Gaschnig's heuristic, repeat the following until the goal state is reached: let B be the current location of the blank; if B is occupied by tile X (not the blank) in the goal state, move X to B; otherwise, move any misplaced tile to B. Students could be asked to prove that this is the optimal solution to the relaxed problem.

3.32 Students should provide results in the form of graphs and/or tables showing both run-time and number of nodes generated. (Different heuristics have different computation costs.) Runtimes may be very small for 8-puzzles, so you may want to assign the 15-puzzle or 24-puzzle instead. The use of pattern databases is also worth exploring experimentally.

Solutions for Chapter 4
Beyond Classical Search

4.1

a. Local beam search with $k = 1$ is hill-climbing search.

b. Local beam search with one initial state and no limit on the number of states retained, resembles breadth-first search in that it adds one complete layer of nodes before adding the next layer. Starting from one state, the algorithm would be essentially identical to breadth-first search except that each layer is generated all at once.

c. Simulated annealing with $T = 0$ at all times: ignoring the fact that the termination step would be triggered immediately, the search would be identical to first-choice hill climbing because every downward successor would be rejected with probability 1. (Exercise may be modified in future printings.)

d. Simulated annealing with $T = \infty$ at all times is a random-walk search: it always accepts a new state.

e. Genetic algorithm with population size $N = 1$: if the population size is 1, then the two selected parents will be the same individual; crossover yields an exact copy of the individual; then there is a small chance of mutation. Thus, the algorithm executes a random walk in the space of individuals.

4.2 Despite its humble origins, this question raises many of the same issues as the scientifically important problem of protein design. There is a discrete assembly space in which pieces are chosen to be added to the track and a continuous configuration space determined by the "joint angles" at every place where two pieces are linked. Thus we can define a state as a set of oriented, linked pieces and the associated joint angles in the range $[-10, 10]$, plus a set of unlinked pieces. The linkage and joint angles exactly determine the physical layout of the track; we can allow for (and penalize) layouts in which tracks lie on top of one another, or we can disallow them. The evaluation function would include terms for how many pieces are used, how many loose ends there are, and (if allowed) the degree of overlap. We might include a penalty for the amount of deviation from 0-degree joint angles. (We could also include terms for "interestingness" and "traversability"—for example, it is nice to be able to drive a train starting from any track segment to any other, ending up in either direction without having to lift up the train.) The tricky part is the set of allowed moves. Obviously we can unlink any piece or link an unlinked piece to an open peg with either orientation at any allowed angle (possibly excluding moves that create overlap). More problematic are moves to join a peg

and hole on already-linked pieces and moves to change the angle of a joint. Changing one angle may force changes in others, and the changes will vary depending on whether the other pieces are at their joint-angle limit. In general there will be no unique "minimal" solution for a given angle change in terms of the consequent changes to other angles, and some changes may be impossible.

4.3 Here is one simple hill-climbing algorithm:

- Connect all the cities into an arbitrary path.
- Pick two points along the path at random.
- Split the path at those points, producing three pieces.
- Try all six possible ways to connect the three pieces.
- Keep the best one, and reconnect the path accordingly.
- Iterate the steps above until no improvement is observed for a while.

4.4 Code not shown.

4.5 See Figure S4.1 for the adapted algorithm. For states that OR-SEARCH finds a solution for it records the solution found. If it later visits that state again it immediately returns that solution.

When OR-SEARCH fails to find a solution it has to be careful. Whether a state can be solved depends on the path taken to that solution, as we do not allow cycles. So on failure OR-SEARCH records the value of $path$. If a state is which has previously failed when $path$ contained any subset of its present value, OR-SEARCH returns failure.

To avoid repeating sub-solutions we can label all new solutions found, record these labels, then return the label if these states are visited again. Post-processing can prune off unused labels. Alternatively, we can output a direct acyclic graph structure rather than a tree.

See (Bertoli *et al.*, 2001) for further details.

4.6

The question statement describes the required changes in detail, see Figure S4.2 for the modified algorithm. When OR-SEARCH cycles back to a state on $path$ it returns a token $loop$ which means to loop back to the most recent time this state was reached along the path to it. Since $path$ is implicitly stored in the returned plan, there is sufficient information for later processing, or a modified implementation, to replace these with labels.

The plan representation is implicitly augmented to keep track of whether the plan is cyclic (i.e., contains a $loop$) so that OR-SEARCH can prefer acyclic solutions.

AND-SEARCH returns failure if all branches lead directly to a $loop$, as in this case the plan will always loop forever. This is the only case it needs to check as if all branches in a finite plan loop there must be some And-node whose children all immediately loop.

4.7 A sequence of actions is a solution to a belief state problem if it takes every initial physical state to a goal state. We can relax this problem by requiring it take only *some* initial physical state to a goal state. To make this well defined, we'll require that it finds a solution

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
  OR-SEARCH(problem.INITIAL-STATE, problem, [])

function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
  if problem.GOAL-TEST(state) then return the empty plan
  if state has previously been solved then return RECALL-SUCCESS(state)
  if state has previously failed for a subset of path then return failure
  if state is on path then
      RECORD-FAILURE(state, path)
      return failure
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
      if plan ≠ failure then
          RECORD-SUCCESS(state, [action | plan])
          return [action | plan]
  return failure

function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
  for each s_i in states do
      plan_i ← OR-SEARCH(s_i, problem, path)
      if plan_i = failure then return failure
  return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n-1} then plan_{n-1} else plan_n]
```

Figure S4.1 AND-OR search with repeated state checking.

for the physical state with the most costly solution. If $h^*(s)$ is the optimal cost of solution starting from the physical state s, then

$$h(S) = \max_{s \in S} h^*(s)$$

is the heuristic estimate given by this relaxed problem. This heuristic assumes any solution to the most difficult state the agent things possible will solve all states.

On the sensorless vacuum cleaner problem in Figure 4.14, h correctly determines the optimal cost for all states except the central three states (those reached by $[suck]$, $[suck, left]$ and $[suck, right]$) and the root, for which h estimates to be 1 unit cheaper than they really are. This means A* will expand these three central nodes, before marching towards the solution.

4.8

a. An action sequence is a solution for belief state b if performing it starting in any state $s \in b$ reaches a goal state. Since any state in a subset of b is in b, the result is immediate.

Any action sequence which is *n*ot a solution for belief state b is also not a solution for any superset; this is the contrapositive of what we've just proved. One cannot, in general, say anything about arbitrary supersets, as the action sequence need not lead to a goal on the states outside of b. One can say, for example, that if an action sequence

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
  OR-SEARCH(problem.INITIAL-STATE, problem, [])

function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
  if problem.GOAL-TEST(state) then return the empty plan
  if state is on path then return loop
  cyclic − plan ← None
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
      if plan ≠ failure then
          if plan is acyclic then return [action | plan]
          cyclic − plan ← [action | plan]
  if cyclic − plan ≠ None then return cyclic − plan
  return failure

function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
  loopy ← True
  for each s_i in states do
      plan_i ← OR-SEARCH(s_i, problem, path)
      if plan_i = failure then return failure
      if plan_i ≠ loop then loopy ← False
  if not loopy then
     return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n−1} then plan_{n−1} else plan_n]
  return failure
```

Figure S4.2 AND-OR search with repeated state checking.

solves a belief state b and a belief state b' then it solves the union belief state $b \cup b'$.

b. On expansion of a node, do not add to the frontier any child belief state which is a superset of a previously explored belief state.

c. If you keep a record of previously solved belief states, add a check to the start of OR-search to check whether the belief state passed in is a subset of a previously solved belief state, returning the previous solution in case it is.

4.9

Consider a very simple example: an initial belief state $\{S_1, S_2\}$, actions a and b both leading to goal state G from either initial state, and

$$c(S_1, a, G) = 3\,; \qquad c(S_2, a, G) = 5\,;$$
$$c(S_1, b, G) = 2\,; \qquad c(S_2, b, G) = 6\,.$$

In this case, the solution $[a]$ costs 3 or 5, the solution $[b]$ costs 2 or 6. Neither is "optimal" in any obvious sense.

In some cases, there *will* be an optimal solution. Let us consider just the deterministic case. For this case, we can think of the cost of a plan as a mapping from each initial physical state to the actual cost of executing the plan. In the example above, the cost for $[a]$ is

$\{S_1{:}3, S_2{:}5\}$ and the cost for $[b]$ is $\{S_1{:}2, S_2{:}6\}$. We can say that plan p_1 *weakly dominates* p_2 if, for each initial state, the cost for p_1 is no higher than the cost for p_2. (Moreover, p_1 *dominates* p_2 if it weakly dominates it *and* has a lower cost for some state.) If a plan p weakly dominates all others, it is optimal. Notice that this definition reduces to ordinary optimality in the observable case where every belief state is a singleton. As the preceding example shows, however, a problem may have no optimal solution in this sense. A perhaps acceptable version of A* would be one that returns any solution that is not dominated by another.

PRINCIPLE OF OPTIMALITY

To understand whether it is possible to apply A* at all, it helps to understand its dependence on Bellman's (1957) **principle of optimality**: *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.* It is important to understand that this is a restriction on performance measures designed to facilitate efficient algorithms, not a general definition of what it means to be optimal.

In particular, if we define the cost of a plan in belief-state space as the minimum cost of any physical realization, we violate Bellman's principle. Modifying and extending the previous example, suppose that a and b reach S_3 from S_1 and S_4 from S_2, and then reach G from there:

$$c(S_1, a, S_3) = 6\,; \qquad c(S_2, a, S_4) = 2\,;$$
$$c(S_1, b, S_3) = 6\,; \qquad c(S_2, b, S_4) = 1\,. c(S_3, a, G) = 2\,; \qquad c(S_4, a, G) = 2\,;$$
$$c(S_3, b, G) = 1\,; \qquad c(S_4, b, G) = 9\,.$$

In the belief state $\{S_3, S_4\}$, the minimum cost of $[a]$ is $\min\{2, 2\} = 2$ and the minimum cost of $[b]$ is $\min\{1, 9\} = 1$, so the optimal plan is $[b]$. In the initial belief state $\{S_1, S_2\}$, the four possible plans have the following costs:

$$[a, a] : \min\{8, 4\} = 4\,; [a, b] : \min\{7, 11\} = 7\,; [b, a] : \min\{8, 3\} = 3\,; [b, b] : \min\{7, 10\} = 7\,.$$

Hence, the optimal plan in $\{S_1, S_2\}$ is $[b, a]$, which does *not* choose b in $\{S_3, S_4\}$ even though that is the optimal plan at that point. This counterintuitive behavior is a direct consequence of choosing the minimum of the possible path costs as the performance measure.

This example gives just a small taste of what might happen with nonadditive performance measures. Details of how to modify and analyze A* for general path-dependent cost functions are give by Dechter and Pearl (1985). Many aspects of A* carry over; for example, we can still derive lower bounds on the cost of a path through a given node. For a belief state b, the minimum value of $g(s) + h(s)$ for each state s in b is a lower bound on the minimum cost of a plan that goes through b.

4.10 The belief state space is shown in Figure S4.3. No solution is possible because no path leads to a belief state all of whose elements satisfy the goal. If the problem is fully observable, the agent reaches a goal state by executing a sequence such that $Suck$ is performed only in a dirty square. This ensures deterministic behavior and every state is obviously solvable.

4.11

The student needs to make several design choices in answering this question. First, how will the vertices of objects be represented? The problem states the percept is a list of vertex positions, but that is not precise enough. Here is one good choice: The agent has an

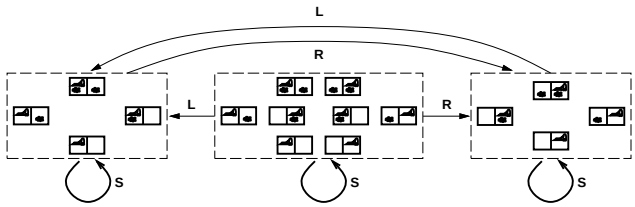


Figure S4.3 The belief state space for the sensorless vacuum world under Murphy's law.

orientation (a heading in degrees). The visible vertexes are listed in clockwise order, starting straight ahead of the agent. Each vertex has a relative angle (0 to 360 degrees) and a distance. We also want to know if a vertex represents the left edge of an obstacle, the right edge, or an interior point. We can use the symbols L, R, or I to indicate this.

The student will need to do some basic computational geometry calculations: intersection of a path and a set of line segments to see if the agent will bump into an obstacle, and visibility calculations to determine the percept. There are efficient algorithms for doing this on a set of line segments, but don't worry about efficiency; an exhaustive algorithm is ok. If this seems too much, the instructor can provide an environment simulator and ask the student only to program the agent.

To answer (c), the student will need some exchange rate for trading off search time with movement time. It is probably too complex to make the simulation asynchronous real-time; easier to impose a penalty in points for computation.

For (d), the agent will need to maintain a set of possible positions. Each time the agent moves, it may be able to eliminate some of the possibilities. The agent may consider moves that serve to reduce uncertainty rather than just get to the goal.

4.12 This question is slightly ambiguous as to what the percept is—either the percept is just the location, or it gives exactly the set of unblocked directions (i.e., blocked directions are illegal actions). We will assume the latter. (Exercise may be modified in future printings.) There are 12 possible locations for internal walls, so there are $2^{12} = 4096$ possible environment configurations. A belief state designates a *subset* of these as possible configurations; for example, before seeing any percepts all 4096 configurations are possible—this is a single belief state.

a. Online search is equivalent to offline search in belief-state space where each action in a belief-state can have multiple successor belief-states: one for each percept the agent could observe after the action. A successor belief-state is constructed by taking the previous belief-state, itself a set of states, replacing each state in this belief-state by the successor state under the action, and removing all successor states which are inconsistent with the percept. This is exactly the construction in Section 4.4.2. AND-OR search can be used to solve this search problem. The initial belief state has $2^1 0 = 1024$ states in it, as we know whether two edges have walls or not (the upper and right edges have no walls) but nothing more. There are $2^{2^{12}}$ possible belief states, one for each set of environment configurations.

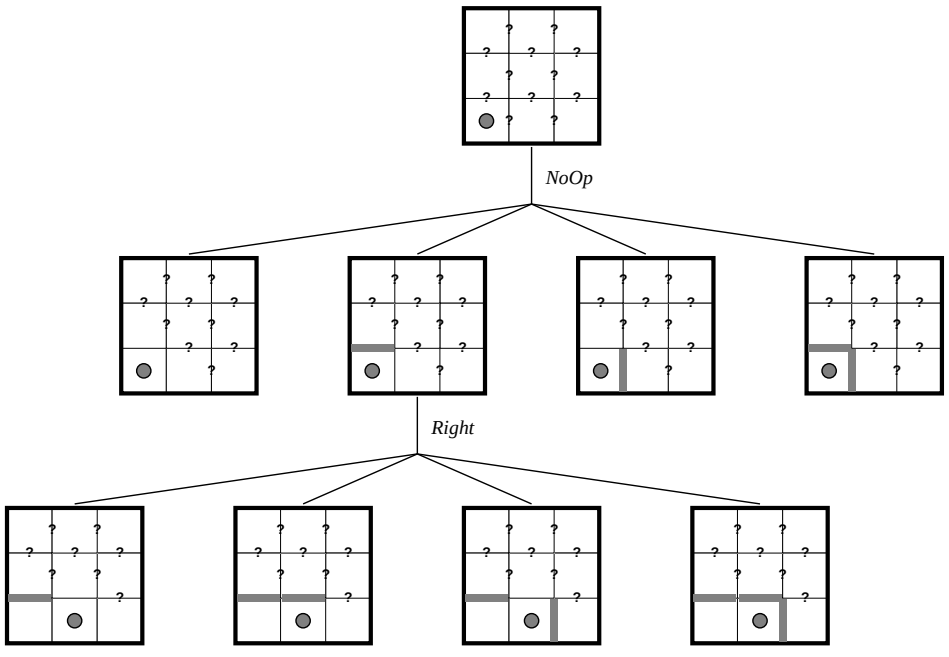


Figure S4.4 The 3×3 maze exploration problem: the initial state, first percept, and one selected action with its perceptual outcomes.

We can view this as a contingency problem in belief state space. After each action and percept, the agent learns whether or not an internal wall exists between the current square and each neighboring square. Hence, each reachable belief state can be represented exactly by a list of status values (present, absent, unknown) for each wall separately. That is, the belief state is completely decomposable and there are exactly 3^{12} reachable belief states. The maximum number of possible wall-percepts in each state is 16 (2^4), so each belief state has four actions, each with up to 16 nondeterministic successors.

b. Assuming the external walls are known, there are two internal walls and hence $2^2 = 4$ possible percepts.

c. The initial null action leads to four possible belief states, as shown in Figure S4.4. From each belief state, the agent chooses a single action which can lead to up to 8 belief states (on entering the middle square). Given the possibility of having to retrace its steps at a dead end, the agent can explore the entire maze in no more than 18 steps, so the complete plan (expressed as a tree) has no more than 8^{18} nodes. On the other hand, there are just 3^{12} reachable belief states, so the plan could be expressed more concisely as a table of actions indexed by belief state (a **policy** in the terminology of Chapter 17).

4.13 Hillclimbing is surprisingly effective at finding reasonable if not optimal paths for very little computational cost, and seldom fails in two dimensions.

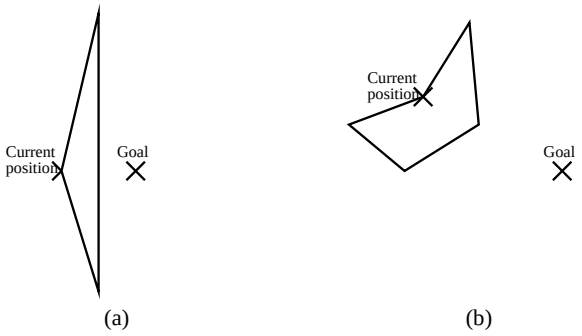


Figure S4.5 (a) Getting stuck with a convex obstacle. (b) Getting stuck with a nonconvex obstacle.

a. It is possible (see Figure S4.5(a)) but very unlikely—the obstacle has to have an unusual shape and be positioned correctly with respect to the goal.

b. With nonconvex obstacles, getting stuck is much more likely to be a problem (see Figure S4.5(b)).

c. Notice that this is just depth-limited search, where you choose a step along the best path even if it is not a solution.

d. Set k to the maximum number of sides of any polygon and you can always escape.

e. LRTA* always makes a move, but may move back if the old state looks better than the new state. But then the old state is penalized for the cost of the trip, so eventually the local minimum fills up and the agent escapes.

4.14

Since we can observe successor states, we always know how to backtrack from to a previous state. This means we can adapt iterative deepening search to solve this problem. The only difference is backtracking must be explicit, following the action which the agent can see leads to the previous state.

The algorithm expands the following nodes:

Depth 1: $(0,0)$, $(1,0)$, $(0,0)$, $(-1,0)$, $(0,0)$

Depth 2: $(0,1)$, $(0,0)$, $(0,-1)$, $(0,0)$, $(1,0)$, $(2,0)$, $(1,0)$, $(0,0)$, $(1,0)$, $(1,1)$, $(1,0)$, $(1,-1)$

Solutions for Chapter 5
Adversarial Search

5.1 The translation uses the model of the opponent $OM(s)$ to fill in the opponent's actions, leaving our actions to be determined by the search algorithm. Let $P(s)$ be the state predicted to occur after the opponent has made all their moves according to OM. Note that the opponent may take multiple moves in a row before we get a move, so we need to define this recursively. We have $P(s) = s$ if PLAYERs is us or TERMINAL-TESTs is true, otherwise $P(s) = P(\text{RESULT}(s, OM(s))$.

The search problem is then given by:

a. Initial state: $P(S_0)$ where S_0 is the initial game state. We apply P as the opponent may play first

b. Actions: defined as in the game by ACTIONSs.

c. Successor function: $\text{RESULT}'(s, a) = P(\text{RESULT}(s, a))$

d. Goal test: goals are terminal states

e. Step cost: the cost of an action is zero unless the resulting state s' is terminal, in which case its cost is $M - \text{UTILITY}(s')$ where $M = \max_s \text{UTILITY}(s)$. Notice that all costs are non-negative.

Notice that the state space of the search problem consists of game state where we are to play and terminal states. States where the opponent is to play have been compiled out. One might alternatively leave those states in but just have a single possible action.

Any of the search algorithms of Chapter 3 can be applied. For example, depth-first search can be used to solve this problem, if all games eventually end. This is equivalent to using the minimax algorithm on the original game if $OM(s)$ always returns the minimax move in s.

5.2

a. Initial state: two arbitrary 8-puzzle states. Successor function: one move on an unsolved puzzle. (You could also have actions that change both puzzles at the same time; this is OK but technically you have to say what happens when one is solved but not the other.) Goal test: both puzzles in goal state. Path cost: 1 per move.

b. Each puzzle has $9!/2$ reachable states (remember that half the states are unreachable). The joint state space has $(9!)^2/4$ states.

c. This is like backgammon; expectiminimax works.

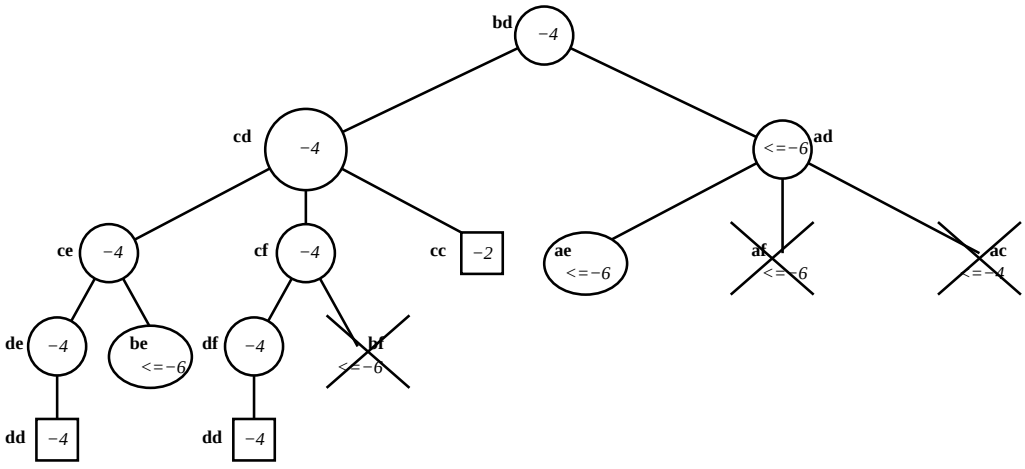


Figure S5.1 Pursuit-evasion solution tree.

d. Actually the statement in the question is not true (it applies to a previous version of part (c) in which the opponent is just trying to prevent you from winning—in that case, the coin tosses will eventually allow you to solve one puzzle without interruptions). For the game described in (c), consider a state in which the coin has come up heads, say, and you get to work on a puzzle that is 2 steps from the goal. Should you move one step closer? If you do, your opponent wins if he tosses heads; or if he tosses tails, you toss tails, and he tosses heads; or any sequence where both toss tails n times and then he tosses heads. So his probability of winning is *at least* $1/2+1/8+1/32+\cdots = 2/3$. So it seems you're better off moving *away* from the goal. (There's no way to stay the same distance from the goal.) This problem unintentionally seems to have the same kind of solution as suicide tictactoe with passing.

5.3

a. See Figure S5.1; the values are just (minus) the number of steps along the path from the root.

b. See Figure S5.1; note that there is both an upper bound and a lower bound for the left child of the root.

c. See figure.

d. The shortest-path length between the two players is a lower bound on the total capture time (here the players take turns, so no need to divide by two), so the "?" leaves have a capture time greater than or equal to the sum of the cost from the root and the shortest-path length. Notice that this bound is derived when the Evader plays very badly. The true value of a node comes from best play by both players, so we can get better bounds by assuming better play. For example, we can get a better bound from the cost when the Evader simply moves backwards and forwards rather than moving towards the Pursuer.

e. See figure (we have used the simple bounds). Notice that once the right child is known

to have a value below –6, the remaining successors need not be considered.

f. The pursuer always wins if the tree is finite. To prove this, let the tree be rooted as the pursuer's current node. (I.e., pick up the tree by that node and dangle all the other branches down.) The evader must either be at the root, in which case the pursuer has won, or in some subtree. The pursuer takes the branch leading to that subtree. This process repeats at most d times, where d is the maximum depth of the original subtree, until the pursuer either catches the evader or reaches a leaf node. Since the leaf has no subtrees, the evader must be at that node.

5.4 The basic physical state of these games is fairly easy to describe. One important thing to remember for Scrabble and bridge is that the physical state is not accessible to all players and so cannot be provided directly to each player by the environment simulator. Particularly in bridge, each player needs to maintain some best guess (or multiple hypotheses) as to the actual state of the world. We expect to be putting some of the game implementations online as they become available.

5.5 Code not shown.

5.6 The most obvious change is that the space of actions is now continuous. For example, in pool, the cueing direction, angle of elevation, speed, and point of contact with the cue ball are all continuous quantities.

The simplest solution is just to discretize the action space and then apply standard methods. This might work for tennis (modelled crudely as alternating shots with speed and direction), but for games such as pool and croquet it is likely to fail miserably because small changes in direction have large effects on action outcome. Instead, one must analyze the game to identify a discrete set of meaningful local goals, such as "potting the 4-ball" in pool or "laying up for the next hoop" in croquet. Then, in the current context, a local optimization routine can work out the best way to achieve each local goal, resulting in a discrete set of possible choices. Typically, these games are stochastic, so the backgammon model is appropriate provided that we use sampled outcomes instead of summing over all outcomes.

Whereas pool and croquet are modelled correctly as turn-taking games, tennis is not. While one player is moving to the ball, the other player is moving to anticipate the opponent's return. This makes tennis more like the simultaneous-action games studied in Chapter 17. In particular, it may be reasonable to derive *randomized* strategies so that the opponent cannot anticipate where the ball will go.

5.7 Consider a MIN node whose children are terminal nodes. If MIN plays suboptimally, then the value of the node is greater than or equal to the value it would have if MIN played optimally. Hence, the value of the MAX node that is the MIN node's parent can only be increased. This argument can be extended by a simple induction all the way to the root. *If the suboptimal play by* MIN *is predictable*, then one can do better than a minimax strategy. For example, if MIN always falls for a certain kind of trap and loses, then setting the trap guarantees a win even if there is actually a devastating response for MIN. This is shown in Figure S5.2.

5.8

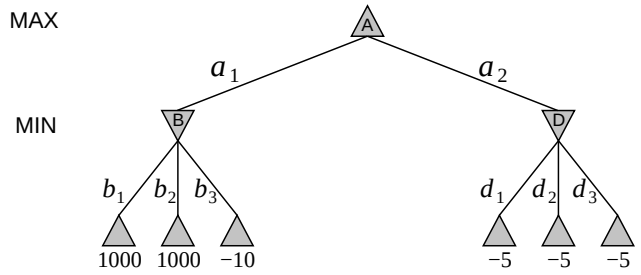


Figure S5.2 A simple game tree showing that setting a trap for MIN by playing a_i is a win if MIN falls for it, but may also be disastrous. The minimax move is of course a_2, with value -5.

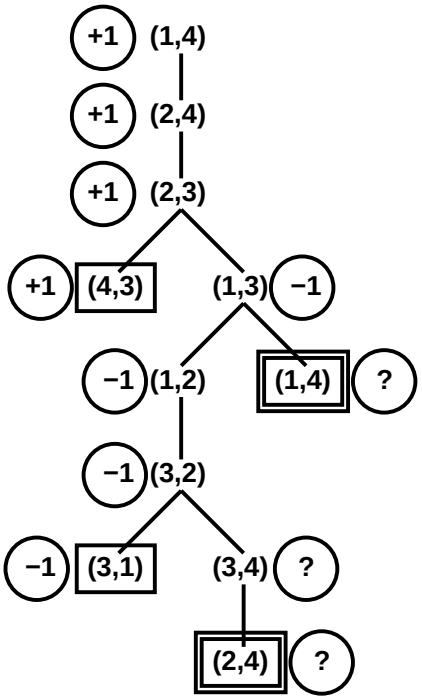


Figure S5.3 The game tree for the four-square game in Exercise 5.8. Terminal states are in single boxes, loop states in double boxes. Each state is annotated with its minimax value in a circle.

a. (5) The game tree, complete with annotations of all minimax values, is shown in Figure S5.3.

b. (5) The "?" values are handled by assuming that an agent with a choice between winning the game and entering a "?" state will always choose the win. That is, min(–1,?) is –1 and max(+1,?) is +1. If all successors are "?", the backed-up value is "?".

c. (5) Standard minimax is depth-first and would go into an infinite loop. It can be fixed

by comparing the current state against the stack; and if the state is repeated, then return a "?" value. Propagation of "?" values is handled as above. Although it works in this case, it does not *always* work because it is not clear how to compare "?" with a drawn position; nor is it clear how to handle the comparison when there are wins of different degrees (as in backgammon). Finally, in games with chance nodes, it is unclear how to compute the average of a number and a "?". Note that it is *not* correct to treat repeated states automatically as drawn positions; in this example, both (1,4) and (2,4) repeat in the tree but they are won positions.

What is really happening is that each state has a well-defined but initially unknown value. These unknown values are related by the minimax equation at the bottom of 164. If the game tree is acyclic, then the minimax algorithm solves these equations by propagating from the leaves. If the game tree has cycles, then a dynamic programming method must be used, as explained in Chapter 17. (Exercise 17.7 studies this problem in particular.) These algorithms can determine whether each node has a well-determined value (as in this example) or is really an infinite loop in that both players prefer to stay in the loop (or have no choice). In such a case, the rules of the game will need to define the value (otherwise the game will never end). In chess, for example, a state that occurs 3 times (and hence is assumed to be desirable for both players) is a draw.

d. This question is a little tricky. One approach is a proof by induction on the size of the game. Clearly, the base case $n = 3$ is a loss for A and the base case $n = 4$ is a win for A. For any $n > 4$, the initial moves are the same: A and B both move one step towards each other. Now, we can see that they are engaged in a subgame of size $n - 2$ on the squares $[2, \ldots, n - 1]$, *except* that there is an extra choice of moves on squares 2 and $n - 1$. Ignoring this for a moment, it is clear that if the "$n - 2$" is won for A, then A gets to the square $n - 1$ before B gets to square 2 (by the definition of winning) and therefore gets to n before B gets to 1, hence the "n" game is won for A. By the same line of reasoning, if "$n - 2$" is won for B then "n" is won for B. Now, the presence of the extra moves complicates the issue, but not too much. First, the player who is slated to win the subgame $[2, \ldots, n - 1]$ never moves back to his home square. If the player slated to lose the subgame does so, then it is easy to show that he is bound to lose the game itself—the other player simply moves forward and a subgame of size $n - 2k$ is played one step closer to the loser's home square.

5.9 For **a**, there are at most 9! games. (This is the number of move sequences that fill up the board, but many wins and losses end before the board is full.) For **b–e**, Figure S5.4 shows the game tree, with the evaluation function values below the terminal nodes and the backed-up values to the right of the non-terminal nodes. The values imply that the best starting move for X is to take the center. The terminal nodes with a bold outline are the ones that do not need to be evaluated, assuming the optimal ordering.

5.10

a. An upper bound on the number of terminal nodes is $N!$, one for each ordering of squares, so an upper bound on the total number of nodes is $\sum_{i=1}^{N} i!$. This is not much

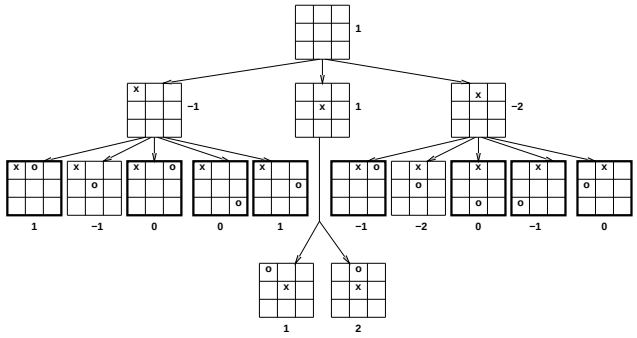


Figure S5.4 Part of the game tree for tic-tac-toe, for Exercise 5.9.

bigger than $N!$ itself as the factorial function grows superexponentially. This is an overestimate because some games will end early when a winning position is filled.

This count doesn't take into account transpositions. An upper bound on the number of distinct game states is 3^N, as each square is either empty or filled by one of the two players. Note that we can determine who is to play just from looking at the board.

b. In this case no games terminate early, and there are $N!$ different games ending in a draw. So ignoring repeated states, we have exactly $\sum_{i=1}^{N} i!$ nodes.

At the end of the game the squares are divided between the two players: $\lceil N/2 \rceil$ to the first player and $\lfloor N/2 \rfloor$ to the second. Thus, a good lower bound on the number of distinct states is $\binom{N}{\lceil N/2 \rceil}$, the number of distinct terminal states.

c. For a state s, let $X(s)$ be the number of winning positions containing no O's and $O(s)$ the number of winning positions containing no X's. One evaluation function is then $Eval(s) = X(s) - O(S)$. Notice that empty winning positions cancel out in the evaluation function.

Alternatively, we might weight potential winning positions by how close they are to completion.

d. Using the upper bound of $N!$ from (a), and observing that it takes $100NN!$ instructions. At 2GHz we have 2 billion instructions per second (roughly speaking), so solve for the largest N using at most this many instructions. For one second we get $N = 9$, for one minute $N = 11$, and for one hour $N = 12$.

5.11 See `"search/algorithms/games.lisp"` for definitions of games, game-playing agents, and game-playing environments. `"search/algorithms/minimax.lisp"` contains the minimax and alpha-beta algorithms. Notice that the game-playing environment is essentially a generic environment with the update function defined by the rules of the game. Turn-taking is achieved by having agents do nothing until it is their turn to move.

See `"search/domains/cognac.lisp"` for the basic definitions of a simple game (slightly more challenging than Tic-Tac-Toe). The code for this contains only a trivial evaluation function. Students can use minimax and alpha-beta to solve small versions of the game to termination (probably up to 4×3); they should notice that alpha-beta is far faster

than minimax, but still cannot scale up without an evaluation function and truncated horizon. Providing an evaluation function is an interesting exercise. From the point of view of data structure design, it is also interesting to look at how to speed up the legal move generator by precomputing the descriptions of rows, columns, and diagonals.

Very few students will have heard of kalah, so it is a fair assignment, but the game is boring—depth 6 lookahead and a purely material-based evaluation function are enough to beat most humans. Othello is interesting and about the right level of difficulty for most students. Chess and checkers are sometimes unfair because usually a small subset of the class will be experts while the rest are beginners.

5.12 The minimax algorithm for non-zero-sum games works exactly as for multiplayer games, described on p.165–6; that is, the evaluation function is a vector of values, one for each player, and the backup step selects whichever vector has the highest value for the player whose turn it is to move. The example at the end of Section 5.2.2 (p.165) shows that alpha-beta pruning is not possible in general non-zero-sum games, because an unexamined leaf node might be optimal for both players.

5.13 This question is not as hard as it looks. The derivation below leads directly to a definition of α and β values. The notation n_i refers to (the value of) the node at depth i on the path from the root to the leaf node n_j. Nodes $n_{i1} \ldots n_{i_{b_i}}$ are the siblings of node i.

a. We can write $n_2 = \max(n_3, n_{31}, \ldots, n_{3b_3})$, giving

$$n_1 = \min(\max(n_3, n_{31}, \ldots, n_{3b_3}), n_{21}, \ldots, n_{2b_2})$$

Then n_3 can be similarly replaced, until we have an expression containing n_j itself.

b. In terms of the l and r values, we have

$$n_1 = \min(l_2, \max(l_3, n_3, r_3), r_2)$$

Again, n_3 can be expanded out down to n_j. The most deeply nested term will be $\min(l_j, n_j, r_j)$.

c. If n_j is a max node, then the lower bound on its value only increases as its successors are evaluated. Clearly, if it exceeds l_j it will have no further effect on n_1. By extension, if it exceeds $\min(l_2, l_4, \ldots, l_j)$ it will have no effect. Thus, by keeping track of this value we can decide when to prune n_j. This is exactly what α-β does.

d. The corresponding bound for min nodes n_k is $\max(l_3, l_5, \ldots, l_k)$.

5.14 The result is given in Section 6 of Knuth (1975). The exact statement (Corollary 1 of Theorem 1) is that the algorithms examines $b^{\lfloor m/2 \rfloor} + b^{\lceil m/2 \rceil} - 1$ nodes at level m. These are exactly the nodes reached when Min plays only optimal moves and/or Max plays only optimal moves. The proof is by induction on m.

5.15 With 32 pieces, each needing 6 bits to specify its position on one of 64 squares, we need 24 bytes (6 32-bit words) to store a position, so we can store roughly 80 million positions in the table (ignoring pointers for hash table bucket lists). This is about 1/22 of the 1800 million positions generated during a three-minute search.

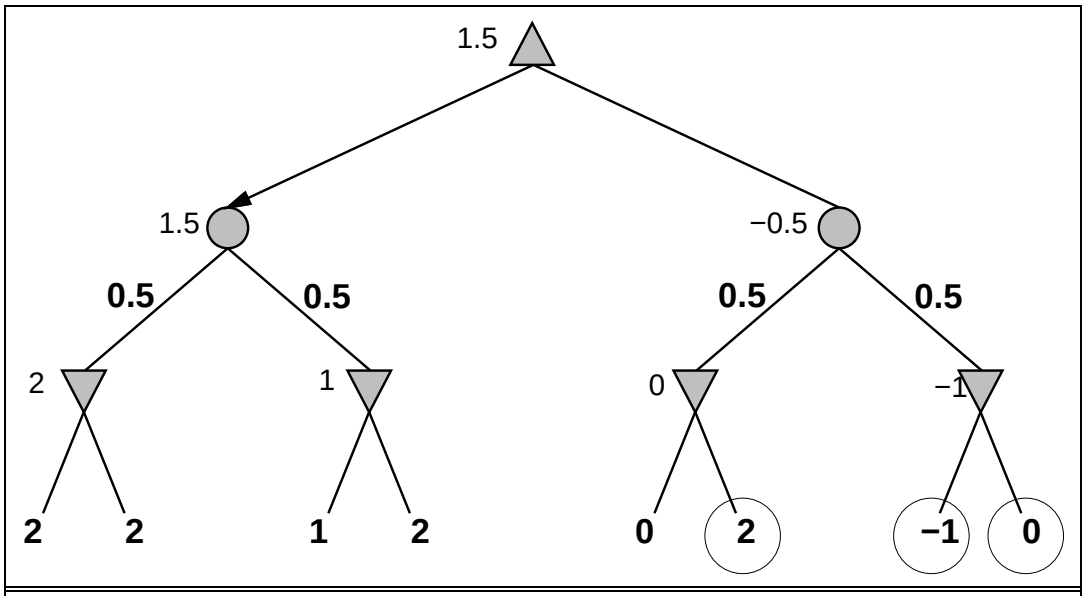


Figure S5.5 Pruning with chance nodes solution.

Generating the hash key directly from an array-based representation of the position might be quite expensive. Modern programs (see, e.g., Heinz, 2000) carry along the hash key and modify it as each new position is generated. Suppose this takes on the order of 20 operations; then on a 2GHz machine where an evaluation takes 2000 operations we can do roughly 100 lookups per evaluation. Using a rough figure of one millisecond for a disk seek, we could do 1000 evaluations per lookup. Clearly, using a disk-resident table is of dubious value, even if we can get some locality of reference to reduce the number of disk reads.

5.16

a. See Figure S5.5.

b. Given nodes 1–6, we would need to look at 7 and 8: if they were both $+\infty$ then the values of the min node and chance node above would also be $+\infty$ and the best move would change. Given nodes 1–7, we do not need to look at 8. Even if it is $+\infty$, the min node cannot be worth more than -1, so the chance node above cannot be worth more than -0.5, so the best move won't change.

c. The worst case is if either of the third and fourth leaves is -2, in which case the chance node above is 0. The best case is where they are both 2, then the chance node has value 2. So it must lie between 0 and 2.

d. See figure.

5.18 The general strategy is to reduce a general game tree to a one-ply tree by induction on the depth of the tree. The inductive step must be done for min, max, and chance nodes, and simply involves showing that the transformation is carried though the node. Suppose that the values of the descendants of a node are $x_1 \ldots x_n$, and that the transformation is $ax + b$, where

a is positive. We have

$$\begin{aligned}
\min(ax_1 + b, ax_2 + b, \ldots, ax_n + b) &= a\min(x_1, x_2, \ldots, x_n) + b \\
\max(ax_1 + b, ax_2 + b, \ldots, ax_n + b) &= a\min(x_1, x_2, \ldots, x_n) + b \\
p_1(ax_1 + b) + p_2(ax_2 + b) + \cdots + p_n(ax_n + b) &= a(p_1x_1 + p_2x_2 + \cdots p_nx_n) + b
\end{aligned}$$

Hence the problem reduces to a one-ply tree where the leaves have the values from the original tree multiplied by the linear transformation. Since $x > y \Rightarrow ax + b > ay + b$ if $a > 0$, the best choice at the root will be the same as the best choice in the original tree.

5.19 This procedure will give incorrect results. Mathematically, the procedure amounts to assuming that averaging commutes with min and max, which it does not. Intuitively, the choices made by each player in the deterministic trees are based on full knowledge of future dice rolls, and bear no necessary relationship to the moves made without such knowledge. (Notice the connection to the discussion of card games in Section 5.6.2 and to the general problem of fully and partially observable Markov decision problems in Chapter 17.) In practice, the method works reasonably well, and it might be a good exercise to have students compare it to the alternative of using expectiminimax with sampling (rather than summing over) dice rolls.

5.20

a. No pruning. In a max tree, the value of the root is the value of the best leaf. Any unseen leaf might be the best, so we have to see them all.

b. No pruning. An unseen leaf might have a value arbitrarily higher or lower than any other leaf, which (assuming non-zero outcome probabilities) means that there is no bound on the value of any incompletely expanded chance or max node.

c. No pruning. Same argument as in (a).

d. No pruning. Nonnegative values allow *lower* bounds on the values of chance nodes, but a lower bound does not allow any pruning.

e. Yes. If the first successor has value 1, the root has value 1 and all remaining successors can be pruned.

f. Yes. Suppose the first action at the root has value 0.6, and the first outcome of the second action has probability 0.5 and value 0; then all other outcomes of the second action can be pruned.

g. (ii) Highest probability first. This gives the strongest bound on the value of the node, all other things being equal.

5.21

a. *In a fully observable, turn-taking, zero-sum game between two perfectly rational players, it does not help the first player to know what strategy the second player is using—that is, what move the second player will make, given the first player's move.*
True. The second player will play optimally, and so is perfectly predictable up to ties. Knowing which of two equally good moves the opponent will make does not change the value of the game to the first player.

b. *In a partially observable, turn-taking, zero-sum game between two perfectly rational players, it does not help the first player to know what move the second player will make, given the first player's move.*
False. In a partially observable game, knowing the second player's move tells the first player additional information about the game state that would otherwise be available only to the second player. For example, in Kriegspiel, knowing the opponent's future move tells the first player where one of the opponent's pieces is; in a card game, it tells the first player one of the opponent's cards.

c. *A perfectly rational backgammon agent never loses.*
False. Backgammon is a game of chance, and the opponent may consistently roll much better dice. The correct statement is that the *expected* winnings are optimal. It is suspected, but not known, that when playing first the expected winnings are positive even against an optimal opponent.

5.22 One can think of chance events during a game, such as dice rolls, in the same way as hidden but preordained information (such as the order of the cards in a deck). The key distinctions are whether the players can influence what information is revealed and whether there is any asymmetry in the information available to each player.

a. Expectiminimax is appropriate only for backgammon and Monopoly. In bridge and Scrabble, each player knows the cards/tiles he or she possesses but not the opponents'. In Scrabble, the benefits of a fully rational, randomized strategy that includes reasoning about the opponents' state of knowledge are probably small, but in bridge the questions of knowledge and information disclosure are central to good play.

b. None, for the reasons described earlier.

c. Key issues include reasoning about the opponent's beliefs, the effect of various actions on those beliefs, and methods for representing them. Since belief states for rational agents are probability distributions over all possible states (including the belief states of others), this is nontrivial.

Solutions for Chapter 6
Constraint Satisfaction Problems

6.1 There are 18 solutions for coloring Australia with three colors. Start with *SA*, which can have any of three colors. Then moving clockwise, *WA* can have either of the other two colors, and everything else is strictly determined; that makes 6 possibilities for the mainland, times 3 for Tasmania yields 18.

6.2

a. Solution A: There is a variable corresponding to each of the n^2 positions on the board.
Solution B: There is a variable corresponding to each knight.

b. Solution A: Each variable can take one of two values, $\{$occupied,vacant$\}$
Solution B: Each variable's domain is the set of squares.

c. Solution A: every pair of squares separated by a knight's move is constrained, such that both cannot be occupied. Furthermore, the entire set of squares is constrained, such that the total number of occupied squares should be k.
Solution B: every pair of knights is constrained, such that no two knights can be on the same square or on squares separated by a knight's move. Solution B may be preferable because there is no global constraint, although Solution A has the smaller state space when k is large.

d. Any solution must describe a *complete-state* formulation because we are using a local search algorithm. For simulated annealing, the successor function must completely connect the space; for random-restart, the goal state must be reachable by hillclimbing from some initial state. Two basic classes of solutions are:
Solution C: ensure no attacks at any time. Actions are to remove any knight, add a knight in any unattacked square, or move a knight to any unattacked square.
Solution D: allow attacks but try to get rid of them. Actions are to remove any knight, add a knight in any square, or move a knight to any square.

6.3 **a.** Crossword puzzle construction can be solved many ways. One simple choice is depth-first search. Each successor fills in a word in the puzzle with one of the words in the dictionary. It is better to go one word at a time, to minimize the number of steps.

b. As a CSP, there are even more choices. You could have a variable for each box in the crossword puzzle; in this case the value of each variable is a letter, and the constraints are

that the letters must make words. This approach is feasible with a most-constraining value heuristic. Alternately, we could have each string of consecutive horizontal or vertical boxes be a single variable, and the domain of the variables be words in the dictionary of the right length. The constraints would say that two intersecting words must have the same letter in the intersecting box. Solving a problem in this formulation requires fewer steps, but the domains are larger (assuming a big dictionary) and there are fewer constraints. Both formulations are feasible.

6.4 **a.** For rectilinear floor-planning, one possibility is to have a variable for each of the small rectangles, with the value of each variable being a 4-tuple consisting of the x and y coordinates of the upper left and lower right corners of the place where the rectangle will be located. The domain of each variable is the set of 4-tuples that are the right size for the corresponding small rectangle and that fit within the large rectangle. Constraints say that no two rectangles can overlap; for example if the value of variable R_1 is $[0, 0, 5, 8]$, then no other variable can take on a value that overlaps with the $0, 0$ to $5, 8$ rectangle.

b. For class scheduling, one possibility is to have three variables for each class, one with times for values (e.g. MWF8:00, TuTh8:00, MWF9:00, ...), one with classrooms for values (e.g. Wheeler110, Evans330, ...) and one with instructors for values (e.g. Abelson, Bibel, Canny, ...). Constraints say that only one class can be in the same classroom at the same time, and an instructor can only teach one class at a time. There may be other constraints as well (e.g. an instructor should not have two consecutive classes).

c. For Hamiltonian tour, one possibility is to have one variable for each stop on the tour, with binary constraints requiring neighboring cities to be connected by roads, and an AllDiff constraint that all variables have a different value.

6.5 The exact steps depend on certain choices you are free to make; here are the ones I made:

a. Choose the X_3 variable. Its domain is $\{0, 1\}$.
b. Choose the value 1 for X_3. (We can't choose 0; it wouldn't survive forward checking, because it would force F to be 0, and the leading digit of the sum must be non-zero.)
c. Choose F, because it has only one remaining value.
d. Choose the value 1 for F.
e. Now X_2 and $\mathbf{X}_1$ are tied for minimum remaining values at 2; let's choose X_2.
f. Either value survives forward checking, let's choose 0 for X_2.
g. Now X_1 has the minimum remaining values.
h. Again, arbitrarily choose 0 for the value of X_1.
i. The variable O must be an even number (because it is the sum of $T + T$ less than 5 (because $O + O = R + 10 \times 0$). That makes it most constrained.
j. Arbitrarily choose 4 as the value of O.
k. R now has only 1 remaining value.
l. Choose the value 8 for R.
m. T now has only 1 remaining value.

n. Choose the value 7 for T.

o. U must be an even number less than 9; choose U.

p. The only value for U that survives forward checking is 6.

q. The only variable left is W.

r. The only value left for W is 3.

s. This is a solution.

This is a rather easy (under-constrained) puzzle, so it is not surprising that we arrive at a solution with no backtracking (given that we are allowed to use forward checking).

6.6 The problem statement sets out the solution fairly completely. To express the ternary constraint on A, B and C that $A + B = C$, we first introduce a new variable, AB. If the domain of A and B is the set of numbers N, then the domain of AB is the set of pairs of numbers from N, i.e. $N \times N$. Now there are three binary constraints, one between A and AB saying that the value of A must be equal to the first element of the pair-value of AB; one between B and AB saying that the value of B must equal the second element of the value of AB; and finally one that says that the sum of the pair of numbers that is the value of AB must equal the value of C. All other ternary constraints can be handled similarly.

Now that we can reduce a ternary constraint into binary constraints, we can reduce a 4-ary constraint on variables A, B, C, D by first reducing A, B, C to binary constraints as shown above, then adding back D in a ternary constraint with AB and C, and then reducing this ternary constraint to binary by introducing CD.

By induction, we can reduce any n-ary constraint to an $(n-1)$-ary constraint. We can stop at binary, because any unary constraint can be dropped, simply by moving the effects of the constraint into the domain of the variable.

6.7 The "Zebra Puzzle" can be represented as a CSP by introducing a variable for each color, pet, drink, country, and cigarette brand (a total of 25 variables). The value of each variable is a number from 1 to 5 indicating the house number. This is a good representation because it easy to represent all the constraints given in the problem definition this way. (We have done so in the Python implementation of the code, and at some point we may reimplement this in the other languages.) Besides ease of expressing a problem, the other reason to choose a representation is the efficiency of finding a solution. here we have mixed results—on some runs, min-conflicts local search finds a solution for this problem in seconds, while on other runs it fails to find a solution after minutes.

Another representation is to have five variables for each house, one with the domain of colors, one with pets, and so on.

6.8

a. $A_1 = R$.

b. $H = R$ conflicts with A_1.

c. $H = G$.

d. $A_4 = R$.

e. $F_1 = R$.

f. $A_2 = R$ conflicts with A_1, $A_2 = G$ conflicts with H, so $A_2 = B$.

g. $F_2 = R$.

h. $A_3 = R$ conflicts with A_4, $A_3 = G$ conflicts with H, $A_3 = B$ conflicts with A_2, so backtrack. Conflict set is $\{A_2, H, A_4\}$, so jump to A_2. Add $\{H, A_4\}$ to A_2's conflict set.

i. A_2 has no more values, so backtrack. Conflict set is $\{A_1, H, A_4\}$ so jump back to A_4. Add $\{A_1, H\}$ to A_4's conflict set.

j. $A_4 = G$ conflicts with H, so $A_4 = B$.

k. $F_1 = R$

l. $A_2 = R$ conflicts with A_1, $A_2 = G$ conflicts with H, so $A_2 = B$.

m. $F_2 = R$

n. $A_3 = R$.

o. $T = R$ conflicts with F_1 and F_2, $T = G$ conflicts with G, so $T = B$.

p. Success.

6.9 The most constrained variable makes sense because it chooses a variable that is (all other things being equal) likely to cause a failure, and it is more efficient to fail as early as possible (thereby pruning large parts of the search space). The least constraining value heuristic makes sense because it allows the most chances for future assignments to avoid conflict.

6.11 We'll trace through each iteration of the **while** loop in AC-3 (for one possible ordering of the arcs):

a. Remove $SA - WA$, delete G from SA.

b. Remove $SA - V$, delete R from SA, leaving only B.

c. Remove $NT - WA$, delete G from NT.

d. Remove $NT - SA$, delete B from NT, leaving only R.

e. Remove $NSW - SA$, delete B from NSW.

f. Remove $NSW - V$, delete R from NSW, leaving only G.

g. Remove $Q - NT$, delete R from Q.

h. Remove $Q - SA$, delete B from Q.

i. remove $Q - NSW$, delete G from Q, leaving no domain for Q.

6.12 On a tree-structured graph, no arc will be considered more than once, so the AC-3 algorithm is $O(ED)$, where E is the number of edges and D is the size of the largest domain.

6.13 The basic idea is to preprocess the constraints so that, for each value of X_i, we keep track of those variables X_k for which an arc from X_k to X_i is satisfied by that particular value of X_i. This data structure can be computed in time proportional to the size of the problem representation. Then, when a value of X_i is deleted, we reduce by 1 the count of allowable

values for each (X_k, X_i) arc recorded under that value. This is very similar to the forward chaining algorithm in Chapter 7. See Mohr and Henderson (1986) for detailed proofs.

6.14

We establish arc-consistency from the bottom up because we will then (after establishing consistency) solve the problem from the top down. It will always be possible to find a solution (if one exists at all) with no backtracking because of the definition of arc consistency: whatever choice we make for the value of the parent node, there will be a value for the child.

6.15

It is certainly possible to solve Sudoku problems in this fashion. However, it is not as effective as the partial-assignment approach, and not as effective as min-conflicts is on the N-queens problem. Perhaps that is because there are two different types of conflicts: a conflict with one of the numbers that defines the initial problem is one that must be corrected, but a conflict between two numbers that were placed elsewhere in the rid can be corrected by replacing either of the two. A version of min-conflicts that recognizes the difference between these two situations might do better than the naive min-conflicts algorithm.

6.16 A **constraint** is a restriction on the possible values of two or more variables. For example, a constraint might say that $A = a$ is not allowed in conjunction with $B = b$.

Backtracking search is a form of depth-first search in which there is a single representation of the state that gets updated for each successor, and then must be restored when a dead end is reached.

A directed arc from variable A to variable B in a CSP is **arc consistent** if, for every value in the current domain of A, there is some consistent value of B.

Backjumping is a way of making backtracking search more efficient, by jumping back more than one level when a dead end is reached.

Min-conflicts is a heuristic for use with local search on CSP problems. The heuristic says that, when given a variable to modify, choose the value that conflicts with the fewest number of other variables.

A **cycle cutset** is a set of variables which when removed from the constraint graph make it acyclic (i.e., a tree). When the variables of a cycle cutset are instantiated the remainder of the CSP can be solved in linear time.

6.17 A simple algorithm for finding a cutset of no more than k nodes is to enumerate all subsets of nodes of size $1, 2, \ldots, k$, and for each subset check whether the remaining nodes form a tree. This algorithm takes time $\binom{\sum_{i=1}^{k} n}{nk}$, which is $O(n^k)$.

Becker and Geiger (1994; http://citeseer.nj.nec.com/becker94approximation.html) give an algorithm called MGA (modified greedy algorithm) that finds a cutset that is no more than twice the size of the minimal cutset, using time $O(E + V \log(V))$, where E is the number of edges and V is the number of variables.

Whether the cycle cutset approach is practical depends more on the graph than on the cutset-finding algorithm. That is because, for a cutset of size c, we still have an exponential (d^c) factor before we can solve the CSP. So any graph with a large cutset will be intractable to solve by this method, even if we could find the cutset with no effort at all.

Solutions for Chapter 7
Logical Agents

7.1 To save space, we'll show the list of models as a table (Figure S7.1) rather than a collection of diagrams. There are eight possible combinations of pits in the three squares, and four possibilities for the wumpus location (including nowhere).

We can see that $KB \models \alpha_2$ because every line where KB is true also has α_2 true. Similarly for α_3.

7.2 As human reasoners, we can see from the first two statements, that if it is mythical, then it is immortal; otherwise it is a mammal. So it must be either immortal or a mammal, and thus horned. That means it is also magical. However, we can't deduce anything about whether it is mythical. To proide a formal answer, we can enumerate the possible worlds ($2^5 = 32$ of them with 5 proposition symbols), mark those in which all the assertions are true, and see which conclusions hold in all of those. Or, we can let the machine do the work—in this case, the Lisp code for propositional reasoning:

```
> (setf kb (make-prop-kb))
#S(PROP-KB SENTENCE (AND))
> (tell kb "Mythical => Immortal")
T
> (tell kb "~Mythical => ~Immortal ^ Mammal")
T
> (tell kb "Immortal | Mammal => Horned")
T
> (tell kb "Horned => Magical")
T
> (ask kb "Mythical")
NIL
> (ask kb "~Mythical")
NIL
> (ask kb "Magical")
T
> (ask kb "Horned")
T
```

7.3

a. See Figure S7.2. We assume the language has built-in Boolean operators **not**, **and**, **or**, **iff**.

Model	KB	α_2	α_3
		$true$	
$P_{1,3}$		$true$	
$P_{2,2}$			
$P_{3,1}$		$true$	
$P_{1,3}$, $P_{2,2}$			
$P_{2,2}$, $P_{3,1}$			
$P_{3,1}$, $P_{1,3}$		$true$	
$P_{1,3}$, $P_{3,1}$, $P_{2,2}$			
$W_{1,3}$		$true$	$true$
$W_{1,3}$, $P_{1,3}$		$true$	$true$
$W_{1,3}$, $P_{2,2}$			$true$
$W_{1,3}$, $P_{3,1}$	$true$	$true$	$true$
$W_{1,3}$, $P_{1,3}$, $P_{2,2}$			$true$
$W_{1,3}$, $P_{2,2}$, $P_{3,1}$			$true$
$W_{1,3}$, $P_{3,1}$, $P_{1,3}$		$true$	$true$
$W_{1,3}$, $P_{1,3}$, $P_{3,1}$, $P_{2,2}$			$true$
$W_{3,1}$,		$true$	
$W_{3,1}$, $P_{1,3}$		$true$	
$W_{3,1}$, $P_{2,2}$			
$W_{3,1}$, $P_{3,1}$		$true$	
$W_{3,1}$, $P_{1,3}$, $P_{2,2}$			
$W_{3,1}$, $P_{2,2}$, $P_{3,1}$			
$W_{3,1}$, $P_{3,1}$, $P_{1,3}$		$true$	
$W_{3,1}$, $P_{1,3}$, $P_{3,1}$, $P_{2,2}$			
$W_{2,2}$		$true$	
$W_{2,2}$, $P_{1,3}$		$true$	
$W_{2,2}$, $P_{2,2}$			
$W_{2,2}$, $P_{3,1}$		$true$	
$W_{2,2}$, $P_{1,3}$, $P_{2,2}$			
$W_{2,2}$, $P_{2,2}$, $P_{3,1}$			
$W_{2,2}$, $P_{3,1}$, $P_{1,3}$		$true$	
$W_{2,2}$, $P_{1,3}$, $P_{3,1}$, $P_{2,2}$			

Figure S7.1 A truth table constructed for Ex. 7.2. Propositions not listed as true on a given line are assumed false, and only $true$ entries are shown in the table.

b. The question is somewhat ambiguous: we can interpret "in a *partial* model" to mean in *all* such models or *some* such models. For the former interpretation, the sentences $False \wedge P$, $True \vee \neg P$, and $P \wedge \neg P$ can all be determined to be true or false in any partial model. For the latter interpretation, we can in addition have sentences such as $A \wedge P$ which is false in the partial model $\{A = false\}$.

c. A general algorithm for partial models must handle the empty partial model, with no assignments. In that case, the algorithm must determine validity and unsatisfiability,

```
function PL-TRUE?(s, m) returns true or false
  if s = True then return true
  else if s = False then return false
  else if SYMBOL?(s) then return LOOKUP(s, m)
  else branch on the operator of s
      ¬: return not PL-TRUE?(ARG1(s), m)
      ∨: return PL-TRUE?(ARG1(s), m) or PL-TRUE?(ARG2(s), m)
      ∧: return PL-TRUE?(ARG1(s), m) and PL-TRUE?(ARG2(s), m)
      ⇒: (not PL-TRUE?(ARG1(s), m)) or PL-TRUE?(ARG2(s), m)
      ⇔: PL-TRUE?(ARG1(s), m) iff PL-TRUE?(ARG2(s), m)
```

Figure S7.2 Pseudocode for evaluating the truth of a sentence wrt a model.

which are co-NP-complete and NP-complete respectively.

d. It helps if **and** and **or** evaluate their arguments in sequence, terminating on false or true arguments, respectively. In that case, the algorithm already has the desired properties: in the partial model where P is true and Q is unknown, $P \vee Q$ returns true, and $\neg P \wedge Q$ returns false. But the truth values of $Q \vee \neg Q$, $Q \vee True$, and $Q \wedge \neg Q$ are not detected.

e. Early termination in Boolean operators will provide a very substantial speedup. In most languages, the Boolean operators already have the desired property, so you would have to write special "dumb" versions and observe a slow-down.

7.4 In all cases, the question can be resolved easily by referring to the definition of entailment.

a. $False \models True$ is true because $False$ has no models and hence entails every sentence AND because $True$ is true in all models and hence is entailed by every sentence.

b. $True \models False$ is false.

c. $(A \wedge B) \models (A \Leftrightarrow B)$ is true because the left-hand side has exactly one model that is one of the two models of the right-hand side.

d. $A \Leftrightarrow B \models A \vee B$ is false because one of the models of $A \Leftrightarrow B$ has both A and B false, which does not satisfy $A \vee B$.

e. $A \Leftrightarrow B \models \neg A \vee B$ is true because the RHS is $A \Rightarrow B$, one of the conjuncts in the definition of $A \Leftrightarrow B$.

f. $(A \wedge B) \Rightarrow C \models (A \Rightarrow C) \vee (B \Rightarrow C)$ is true because the RHS is false only when both disjuncts are false, i.e., when A and B are true and C is false, in which case the LHS is also false. This may seem counterintuitive, and would not hold if $\Rightarrow$ is interpreted as "causes."

g. $(C \vee (\neg A \wedge \neg B)) \equiv ((A \Rightarrow C) \wedge (B \Rightarrow C))$ is true; proof by truth table enumeration, or by application of distributivity (Fig 7.11).

h. $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B)$ is true; removing a conjunct only allows more models.

i. $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B) \wedge (\neg D \vee E)$ is false; removing a disjunct allows fewer models.

j. $(A \vee B) \wedge \neg(A \Rightarrow B)$ *is* satisfiable; model has A and $\neg B$.

k. $(A \Leftrightarrow B) \wedge (\neg A \vee B)$ *is* satisfiable; RHS is entailed by LHS so models are those of $A \Leftrightarrow B$.

l. $(A \Leftrightarrow B) \Leftrightarrow C$ does have the same number of models as $(A \Leftrightarrow B)$; half the models of $(A \Leftrightarrow B)$ satisfy $(A \Leftrightarrow B) \Leftrightarrow C$, as do half the non-models, and there are the same numbers of models and non-models.

7.5 Remember, $\alpha \models \beta$ iff in every model in which α is true, β is also true. Therefore,

a. α is valid if and only if $True \models \alpha$.
Forward: If $alpha$ is *valid* it is true in all models, hence it is true in all models of $True$.
Backward: if $True \models \alpha$ then α must be true in all models of $True$, i.e., in all models, hence α must be valid.

b. For any α, $False \models \alpha$.
$False$ doesn't hold in any model, so α trivially holds in every model of $False$.

c. $\alpha \models \beta$ if and only if the sentence $(\alpha \Rightarrow \beta)$ is valid.
Both sides are equivalent to the assertion that there is no model in which α is true and β is false, i.e., no model in which $\alpha \Rightarrow \beta$ is false.

d. $\alpha \equiv \beta$ if and only if the sentence $(\alpha \Leftrightarrow \beta)$ is valid.
Both sides are equivalent to the assertion that α and β have the same truth value in every model.

e. $\alpha \models \beta$ if and only if the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable.
As in **c**, both sides are equivalent to the assertion that there is no model in which α is true and β is false.

7.6

a. If $\alpha \models \gamma$ or $\beta \models \gamma$ (or both) then $(\alpha \wedge \beta) \models \gamma$.
True. This follows from monotonicity.

b. If $\alpha \models (\beta \wedge \gamma)$ then $\alpha \models \beta$ and $\alpha \models \gamma$.
True. If $\beta \wedge \gamma$ is true in every model of α, then β and γ are true in every model of α, so $\alpha \models \beta$ and $\alpha \models \gamma$.

c. If $\alpha \models (\beta \vee \gamma)$ then $\alpha \models \beta$ or $\alpha \models \gamma$ (or both).
False. Consider $\beta \equiv A$, $\gamma \equiv \neg A$.

7.7 These can be computed by counting the rows in a truth table that come out true, but each has some simple property that allows a short-cut:

a. Sentence is false only if B and C are false, which occurs in 4 cases for A and D, leaving 12.

b. Sentence is false only if A, B, C, and D are false, which occurs in 1 case, leaving 15.

c. The last four conjuncts specify a model in which the first conjunct is false, so 0.

7.8 A binary logical connective is defined by a truth table with 4 rows. Each of the four rows may be true or false, so there are $2^4 = 16$ possible truth tables, and thus 16 possible connectives. Six of these are trivial ones that ignore one or both inputs; they correspond to $True$, $False$, P, Q, $\neg P$ and $\neg Q$. Four of them we have already studied: $\wedge, \vee, \Rightarrow, \Leftrightarrow$. The remaining six are potentially useful. One of them is reverse implication ($\Leftarrow$ instead of $\Rightarrow$), and the other five are the negations of $\wedge, \vee, \Leftrightarrow, \Rightarrow$ and $\Leftarrow$. The first three of these are sometimes called *nand*, *nor*, and *xor*.

7.9 We use the truth table code in Lisp in the directory `logic/prop.lisp` to show each sentence is valid. We substitute `P, Q, R` for α, β, γ because of the lack of Greek letters in ASCII. To save space in this manual, we only show the first four truth tables:

```
> (truth-table "P ^ Q <=> Q ^ P")
----------------------------------------
 P  Q  P ^ Q  Q ^ P  (P ^ Q) <=> (Q ^ P)
----------------------------------------
 F  F    F      F            \(true\)
 T  F    F      F             T
 F  T    F      F             T
 T  T    T      T             T
----------------------------------------
NIL

> (truth-table "P | Q <=> Q | P")
----------------------------------------
 P  Q  P | Q  Q | P  (P | Q) <=> (Q | P)
----------------------------------------
 F  F    F      F             T
 T  F    T      T             T
 F  T    T      T             T
 T  T    T      T             T
----------------------------------------
NIL

> (truth-table "P ^ (Q ^ R) <=> (P ^ Q) ^ R")
------------------------------------------------------------------------
 P  Q  R  Q ^ R  P ^ (Q ^ R)  P ^ Q ^ R  (P ^ (Q ^ R)) <=> (P ^ Q ^ R)
------------------------------------------------------------------------
 F  F  F    F         F           F                    T
 T  F  F    F         F           F                    T
 F  T  F    F         F           F                    T
 T  T  F    F         F           F                    T
 F  F  T    F         F           F                    T
 T  F  T    F         F           F                    T
 F  T  T    T         F           F                    T
 T  T  T    T         T           T                    T
------------------------------------------------------------------------
NIL

> (truth-table "P | (Q | R) <=> (P | Q) | R")
------------------------------------------------------------------------
 P  Q  R  Q | R  P | (Q | R)  P | Q | R  (P | (Q | R)) <=> (P | Q | R)
```

```
------------------------------------------------------------------------
 F  F  F    F         F           F                    T
 T  F  F    F         T           T                    T
 F  T  F    T         T           T                    T
 T  T  F    T         T           T                    T
 F  F  T    T         T           T                    T
 T  F  T    T         T           T                    T
 F  T  T    T         T           T                    T
 T  T  T    T         T           T                    T
------------------------------------------------------------------------
NIL
```

For the remaining sentences, we just show that they are valid according to the `validity` function:

```
> (validity "~~P <=> P")
VALID
> (validity "P => Q <=> ~Q => ~P")
VALID
> (validity "P => Q <=> ~P | Q")
VALID
> (validity "(P <=> Q) <=> (P => Q) ^ (Q => P)")
VALID
> (validity "~(P ^ Q) <=> ~P | ~Q")
VALID
> (validity "~(P | Q) <=> ~P ^ ~Q")
VALID
> (validity "P ^ (Q | R) <=> (P ^ Q) | (P ^ R)")
VALID
> (validity "P | (Q ^ R) <=> (P | Q) ^ (P | R)")
VALID
```

7.10

a. Valid.

b. Neither.

c. Neither.

d. Valid.

e. Valid.

f. Valid.

g. Valid.

7.11 Each possible world can be written as a conjunction of literals, e.g. $(A \wedge B \wedge \neg C)$. Asserting that a possible world is not the case can be written by negating that, e.g. $\neg(A \wedge B \wedge \neg C)$, which can be rewritten as $(\neg A \vee \neg B \vee C)$. This is the form of a clause; a conjunction of these clauses is a CNF sentence, and can list the negations of all the possible worlds that would make the sentence false.

7.12 To prove the conjunction, it suffices to prove each literal separately. To prove $\neg B$, add the negated goal S7: B.

- Resolve S7 with S5, giving S8: F.
- Resolve S7 with S6, giving S9: C.
- Resolve S8 with S3, giving S10: $(\neg C \vee \neg B)$.
- Resolve S9 with S10, giving S11: $\neg B$.
- Resolve S7 with S11 giving the empty clause.

To prove $\neg A$, add the negated goal S7: A.

- Resolve S7 with the first clause of S1, giving S8: $(B \vee E)$.
- Resolve S8 with S4, giving S9: B.
- Proceed as above to derive the empty clause.

7.13

a. $P \Rightarrow Q$ is equivalent to $\neg P \vee Q$ by implication elimination (Figure 7.11), and $\neg(P_1 \wedge \cdots \wedge P_m)$ is equivalent to $(\neg P_1 \vee \cdots \vee \neg P_m)$ by de Morgan's rule, so $(\neg P_1 \vee \cdots \vee \neg P_m \vee Q)$ is equivalent to $(P_1 \wedge \cdots \wedge P_m) \Rightarrow Q$.

b. A clause can have positive and negative literals; let the negative literals have the form $\neg P_1, \ldots, \neg P_m$ and let the positive literals have the form $Q_1, \ldots, Q_n$, where the P_is and Q_js are symbols. Then the clause can be written as $(\neg P_1 \vee \cdots \vee \neg P_m \vee Q_1 \vee \cdots \vee Q_n)$. By the previous argument, with $Q = Q_1 \vee \cdots \vee Q_n$, it is immediate that the clause is equivalent to

$$(P_1 \wedge \cdots \wedge P_m) \Rightarrow Q_1 \vee \cdots \vee Q_n \ .$$

c. For atoms p_i, q_i, r_i, s_i where $p_j = q_k$:

$$\frac{\begin{array}{c} p_1 \wedge \ldots \ p_j \ \ldots \wedge p_{n_1} \Rightarrow r_1 \vee \ldots r_{n_2} \\ s_1 \wedge \ldots \wedge s_{n_3} \Rightarrow q_1 \vee \ldots \ q_k \ \ldots \vee q_{n_4} \end{array}}{p_1 \wedge \ldots p_{j-1} \wedge p_{j+1} \wedge p_{n_1} \wedge s_1 \wedge \ldots s_{n_3} \Rightarrow r_1 \vee \ldots r_{n_2} \vee q_1 \vee \ldots \ q_{k-1} \vee q_{k+1} \vee \ldots \vee q_{n_4}}$$

7.14

a. Correct representations of "a person who is radical is electable if he/she is conservative, but otherwise is not electable":

(i) $(R \wedge E) \iff C$
No; this sentence asserts, among other things, that all conservatives are radical, which is not what was stated.

(ii) $R \Rightarrow (E \iff C)$
Yes, this says that if a person is a radical then they are electable if and only if they are conservative.

(iii) $R \Rightarrow ((C \Rightarrow E) \vee \neg E)$
No, this is equivalent to $\neg R \vee \neg C \vee E \vee \neg E$ which is a tautology, true under any assignment.

b. Horn form:

(i) Yes:

$$\begin{aligned}(R \wedge E) \iff C &\equiv ((R \wedge E) \Rightarrow C) \wedge (C \Rightarrow (R \wedge E)) \\ &\equiv ((R \wedge E) \Rightarrow C) \wedge (C \Rightarrow R) \wedge (C \Rightarrow E)\end{aligned}$$

(ii) Yes:

$$\begin{aligned}R \Rightarrow (E \iff C) &\equiv R \Rightarrow ((E \Rightarrow C) \wedge (C \Rightarrow E)) \\ &\equiv \neg R \vee ((\neg E \vee C) \wedge (\neg C \vee E)) \\ &\equiv (\neg R \vee \neg E \vee C) \wedge (\neg R \vee \neg C \vee E)\end{aligned}$$

(iii) Yes, e.g., $True \Rightarrow True$.

7.15

a. The graph is simply a connected chain of 5 nodes, one per variable.

b. $n + 1$ solutions. Once any X_i is true, all subsequent X_js must be true. Hence the solutions are i falses followed by $n - i$ trues, for $i = 0, \ldots, n$.

c. The complexity is $O(n^2)$. This is somewhat tricky. Consider what part of the complete binary tree is explored by the search. The algorithm must follow all solution sequences, which themselves cover a quadratic-sized portion of the tree. Failing branches are all those trying a $false$ after the preceding variable is assigned $true$. Such conflicts are detected immediately, so they do not change the quadratic cost.

d. These facts are not obviously connected. Horn-form logical inference problems need not have tree-structured constraint graphs; the linear complexity comes from the nature of the constraint (implication) not the structure of the problem.

7.16 A clause is a disjunction of literals, and its models are the *union* of the sets of models of each literal; and each literal satisfies half the possible models. (Note that $False$ is unsatisfiable, but it is really another name for the empty clause.) A 3-SAT clause with three distinct variables rules out exactly 1/8 of all possible models, so five clauses can rule out no more than 5/8 of the models. Eight clauses are needed to rule out all models. Suppose we have variables A, B, C. There are eight models, and we write one clause to rule out each model. For example, the model $\{A = false, B = false, C = false\}$ is ruled out by the clause $(\neg A \vee \neg B \vee \neg C)$.

7.17

a. The negated goal is $\neg G$. Resolve witht he last two clauses to produce $\neg C$ and $\neg D$. Resolve with the second and third clauses to produce $\neg A$ and $\neg B$. Resolve these successively against the first clause to produce the empty clause.

b. This can be answered with or without $True$ and $False$ symbols; we'll omit them for simplicity. First, each 2-CNF clause has two places to put literals. There are $2n$ distinct literals, so there are $(2n)^2$ syntactically distinct clauses. Now, many of these clauses are semantically identical. Let us handle them in groups. There are $C(2n, 2) = (2n)(2n - 1)/2 = 2n^2 - n$ clauses with two different literals, if we ignore ordering. All these

clauses are semantically distinct except those that are equivalent to $True$ (e.g., $(A \vee \neg A)$), of which there are n, so that makes $2n^2 - 2n + 1$ clauses with distinct literals. There are $2n$ clauses with repeated literals, all distinct. So there are $2n^2 + 1$ distinct clauses in all.

c. Resolving two 2-CNF clauses cannot increase the clause size; therefore, resolution can generate only $O(n^2)$ distinct clauses before it must terminate.

d. First, note that the number of 3-CNF clauses is $O(n^3)$, so we cannot argue for nonpolynomial complexity on the basis of the number of different clauses! The key observation is that resolving two 3-CNF clauses can *increase* the clause size to 4, and so on, so clause size can grow to $O(n)$, giving $O(2^n)$ possible clauses.

7.18

a. A simple truth table has eight rows, and shows that the sentence is true for all models and hence valid.

b. For the left-hand side we have:

$$(Food \Rightarrow Party) \vee (Drinks \Rightarrow Party)$$
$$(\neg Food \vee Party) \vee (\neg Drinks \vee Party)$$
$$(\neg Food \vee Party \vee \neg Drinks \vee Party)$$
$$(\neg Food \vee \neg Drinks \vee Party)$$

and for the right-hand side we have

$$(Food \wedge Drinks) \Rightarrow Party$$
$$\neg(Food \wedge Drinks) \vee Party$$
$$(\neg Food \vee \neg Drinks) \vee Party$$
$$(\neg Food \vee \neg Drinks \vee Party)$$

The two sides are identical in CNF, and hence the original sentence is of the form $P \Rightarrow P$, which is valid for any P.

c. To prove that a sentence is valid, prove that its negation is unsatisfiable. I.e., negate it, convert to CNF, use resolution to prove a contradiction. We can use the above CNF result for the LHS.

$$\neg[[(Food \Rightarrow Party) \vee (Drinks \Rightarrow Party)] \Rightarrow [(Food \wedge Drinks) \Rightarrow Party]]$$
$$[(Food \Rightarrow Party) \vee (Drinks \Rightarrow Party)] \wedge \neg[(Food \wedge Drinks) \Rightarrow Party]$$
$$(\neg Food \vee \neg Drinks \vee Party) \wedge Food \wedge Drinks \wedge \neg Party$$

Each of the three unit clauses resolves in turn against the first clause, leaving an empty clause.

7.19

a. Each possible world can be expressed as the conjunction of all the literals that hold in the model. The sentence is then equivalent to the disjunction of all these conjunctions, i.e., a DNF expression.

b. A trivial conversion algorithm would enumerate all possible models and include terms corresponding to those in which the sentence is true; but this is necessarily exponential-time. We can convert to DNF using the same algorithm as for CNF except that we distribute $\wedge$ over $\vee$ at the end instead of the other way round.

c. A DNF expression is satisfiable if it contains at least one term that has no contradictory literals. This can be checked in linear time, or even during the conversion process. Any completion of that term, filling in missing literals, is a model.

d. The first steps give

$$(\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee \neg A) \ .$$

Converting to DNF means taking one literal from each clause, in all possible ways, to generate the terms (8 in all). Choosing each literal corresponds to choosing the truth value of each variable, so the process is very like enumerating all possible models. Here, the first term is $(\neg A \wedge \neg B \wedge \neg C)$, which is clearly satisfiable.

e. The problem is that the final step typically results in DNF expressions of exponential size, so we require both exponential time AND exponential space.

7.20 The CNF representations are as follows:

S1: $(\neg A \vee B \vee E) \wedge (\neg B \vee A) \wedge (\neg E \vee A)$.
S2: $(\neg E \vee D)$.
S3: $(\neg C \vee \neg F \vee \neg B)$.
S4: $(\neg E \vee B)$.
S5: $(\neg B \vee F)$.
S6: $(\neg B \vee C)$.

We omit the DPLL trace, which is easy to obtain from the version in the code repository.

7.21 It is more likely to be solvable: adding literals to disjunctive clauses makes them easier to satisfy.

7.22

a. This is a disjunction with 28 disjuncts, each one saying that two of the neighbors are true and the others are false. The first disjunct is

$$X_{2,2} \wedge X_{1,2} \wedge \neg X_{0,2} \wedge \neg X_{0,1} \wedge \neg X_{2,1} \wedge \neg X_{0,0} \wedge \neg X_{1,0} \wedge \neg X_{2,0}$$

The other 27 disjuncts each select two different $X_{i,j}$ to be true.

b. There will be $\binom{n}{k}$ disjuncts, each saying that k of the n symbols are true and the others false.

c. For each of the cells that have been probed, take the resulting number n revealed by the game and construct a sentence with $\binom{n}{8}$ disjuncts. Conjoin all the sentences together. Then use DPLL to answer the question of whether this sentence entails $X_{i,j}$ for the particular i, j pair you are interested in.

d. To encode the global constraint that there are M mines altogether, we can construct a disjunct with $\binom{M}{N}$ disjuncts, each of size N. Remember, $\binom{M}{N=M!/(M-N)!}$. So for

a Minesweeper game with 100 cells and 20 mines, this will be morre than 10^{39}, and thus cannot be represented in any computer. However, we can represent the global constraint within the DPLL algorithm itself. We add the parameter *min* and *max* to the DPLL function; these indicate the minimum and maximum number of unassigned symbols that must be true in the model. For an unconstrained problem the values 0 and N will be used for these parameters. For a mineseeper problem the value M will be used for both *min* and *max*. Within DPLL, we fail (return false) immediately if *min* is less than the number of remaining symbols, or if *max* is less than 0. For each recursive call to DPLL, we update *min* and *max* by subtracting one when we assign a true value to a symbol.

e. No conclusions are invalidated by adding this capability to DPLL and encoding the global constraint using it.

f. Consider this string of alternating 1's and unprobed cells (indicated by a dash):

```
|-|1|-|1|-|1|-|1|-|1|-|1|-|1|-|
```

There are two possible models: either there are mines under every even-numbered dash, or under every odd-numbered dash. Making a probe at either end will determine whether cells at the far end are empty or contain mines.

7.23 It will take time proportional to the number of pure symbols plus the number of unit clauses. We assume that $KB \Rightarrow \alpha$ is false, and prove a contradiction. $\neg(KB \Rightarrow \alpha)$ is equivalent to $KB \wedge \neg\alpha$. From this sentence the algorithm will first eliminate all the pure symbols, then it will work on unit clauses until it chooses either α or $\neg\alpha$ (both of which are unit clauses); at that point it will immediately recognize that either choice (true or false) for α leads to failure, which means that the original non-negated assertion α is entailed.

7.24 We omit the DPLL trace, which is easy to obtain from the version in the code repository. The behavior is very similar: the unit-clause rule in DPLL ensures that all known atoms are propagated to other clauses.

7.25

$$Locked^{t+1} \Leftrightarrow [Lock^t \vee (Locked^t \wedge \neg Unlock^t)] .$$

7.26 The remaining fluents are the orientation fluents ($FacingEast$ etc.) and $WumpusAlive$. The successor-state axioms are as follows:

$$\begin{aligned} FacingEast^{t+1} &\Leftrightarrow (FacingEast^t \wedge \neg(TurnLeft^t \vee TurnRight^t)) \\ &\vee (FacingNorth^t \wedge TurnRight^t) \\ &\vee (FacingSouth^t \wedge TurnLeft^t) \\ WumpusAlive^{t+1} &\Leftrightarrow WumpusAlive^t \wedge \neg(WumpusAhead^t \wedge HaveArrow^t \wedge Shoot^t) . \end{aligned}$$

The $WumpusAhead$ fluent does not need a successor-state axiom, since it is definable synchronously in terms of the agent location and orientation fluents and the wumpus location. The definition is extraordinarily tedious, illustrating the weakness of proposition logic. Note also that in the second edition we described a successor-state axiom (in the form of a circuit)

for $WumpusAlive$ that used the $Scream$ observation to infer the wumpus's death, with no need for describing the complicated physics of shooting. Such an axiom suffices for state estimation, but nor for planning.

7.27

The required modifications are to add definitional axioms such as

$$P_{3,1\ or\ 2,2} \Leftrightarrow P_{3,1} \vee P_{2,2}$$

and to include the new literals on the list of literals whose truth values are to be inferred at each time step.

One natural way to extend the 1-CNF representation is to add test additional non-literal sentences. The sentences we choose to test can depend on inferences from the current KB. This can work if the number of additional sentences we need to test is not too large.

For example, we can query the knowledge base to find out which squares we know have pits, which we know might have pits, and which states are breezy (we need to do this to compute the un-augmented 1-CNF belief state). Then, for each breezy square, test the sentence "one of the neighbours of this square which might have a bit does have a pit." For example, this would test $P_{3,1} \vee P_{2,2}$ if we had perceived a breeze in square (2,1). Under the Wumpus physics, this literal will be true iff the breezy square has no known pit around it.

Solutions for Chapter 8
First-Order Logic

8.1 This question will generate a wide variety of possible solutions. The key distinction between analogical and sentential representations is that the analogical representation automatically generates consequences that can be "read off" whenever suitable premises are encoded. When you get into the details, this distinction turns out to be quite hard to pin down—for example, what does "read off" mean?—but it can be justified by examining the time complexity of various inferences on the "virtual inference machine" provided by the representation system.

a. Depending on the scale and type of the map, symbols in the map language typically include city and town markers, road symbols (various types), lighthouses, historic monuments, river courses, freeway intersections, etc.

b. Explicit and implicit sentences: this distinction is a little tricky, but the basic idea is that when the map-drawer plunks a symbol down in a particular place, he says one explicit thing (e.g. that Coit Tower is here), but the analogical structure of the map representation means that many implicit sentences can now be derived. Explicit sentences: there is a monument called Coit Tower at this location; Lombard Street runs (approximately) east-west; San Francisco Bay exists and has this shape. Implicit sentences: Van Ness is longer than North Willard; Fisherman's Wharf is north of the Mission District; the shortest drivable route from Coit Tower to Twin Peaks is the following ....

c. Sentences unrepresentable in the map language: Telegraph Hill is approximately conical and about 430 feet high (assuming the map has no topographical notation); in 1890 there was no bridge connecting San Francisco to Marin County (map does not represent changing information); Interstate 680 runs either east or west of Walnut Creek (no disjunctive information).

d. Sentences that are easier to express in the map language: any sentence that can be written easily in English is not going to be a good candidate for this question. Any *linguistic* abstraction from the physical structure of San Francisco (e.g. San Francisco is on the end of a peninsula at the mouth of a bay) can probably be expressed equally easily in the predicate calculus, since that's what it was designed for. Facts such as the shape of the coastline, or the path taken by a road, are best expressed in the map language. Even then, one can argue that the coastline drawn on the map actually consists of lots of individual sentences, one for each dot of ink, especially if the map is drawn

using a digital plotter. In this case, the advantage of the map is really in the ease of inference combined with suitability for human "visual computing" apparatus.

e. Examples of other analogical representations:

- Analog audio tape recording. Advantages: simple circuits can record and reproduce sounds. Disadvantages: subject to errors, noise; hard to process in order to separate sounds or remove noise etc.
- Traditional clock face. Advantages: easier to read quickly, determination of how much time is available requires no additional computation. Disadvantages: hard to read precisely, cannot represent small units of time (ms) easily.
- All kinds of graphs, bar charts, pie charts. Advantages: enormous data compression, easy trend analysis, communicate information in a way which we can interpret easily. Disadvantages: imprecise, cannot represent disjunctive or negated information.

8.2 The knowledge base does not entail $\forall x\ P(x)$. To show this, we must give a model where $P(a)$ and $P(b)$ but $\forall x\, P(x)$ is false. Consider any model with three domain elements, where a and b refer to the first two elements and the relation referred to by P holds only for those two elements.

8.3 The sentence $\exists x, y \quad x = y$ is valid. A sentence is valid if it is true in every model. An existentially quantified sentence is true in a model if it holds under any extended interpretation in which its variables are assigned to domain elements. According to the standard semantics of FOL as given in the chapter, every model contains at least one domain element, hence, for any model, there is an extended interpretation in which x and y are assigned to the first domain element. In such an interpretation, $x = y$ is true.

8.4 $\forall x, y \quad x = y$ stipulates that there is exactly one object. If there are two objects, then there is an extended interpretation in which x and y are assigned to different objects, so the sentence would be false. Some students may also notice that any unsatisfiable sentence also meets the criterion, since there are no worlds in which the sentence is true.

8.5 We will use the simplest counting method, ignoring redundant combinations. For the constant symbols, there are D^c assignments. Each predicate of arity k is mapped onto a k-ary relation, i.e., a subset of the D^k possible k-element tuples; there are 2^{D^k} such mappings. Each function symbol of arity k is mapped onto a k-ary function, which specifies a value for each of the D^k possible k-element tuples. Including the invisible element, there are $D + 1$ choices for each value, so there are $(D + 1)^{D^k}$ functions. The total number of possible combinations is therefore

$$D^c \cdot \left(\sum_{k=1}^{A} 2^{D^k} \right) \cdot \left(\sum_{k=1}^{A} (D+1)^{D^k} \right) .$$

Two things to note: first, the number is finite; second, the maximum arity A is the most crucial complexity parameter.

8.6 Validity in first-order logic requires truth in all possible models:

a. $(\exists x\ x = x) \Rightarrow (\forall y\ \ \exists z\ y = z)$.
Valid. The LHS is valid by itself—in standard FOL, every model has at least one object; hence, the whole sentence is valid iff the RHS is valid. (Otherwise, we can find a model where the LHS is true and the RHS is false.) The RHS is valid because for every value of y in any given model, there is a z—namely, the value of y itself—that is identical to y.

b. $\forall x\ \ P(x) \vee \neg P(x)$.
Valid. For any relation denoted by P, every object x is either in the relation or not in it.

c. $\forall x\ \ Smart(x) \vee (x = x)$.
Valid. In every model, every object satisfies $x = x$, so the disjunction is satisfied regardless of whether x is smart.

8.7 This version of FOL, first studied in depth by Mostowski (1951), goes under the title of **free logic** (Lambert, 1967). By a natural extension of the truth values for empty conjunctions (true) and empty disjunctions (false), every universally quantified sentence is true in empty models and every existentially quantified sentence is false. The semantics also needs to be adjusted to handle the fact that constant symbols have no referent in an empty model.

Examples of sentences valid in the standard semantics but not in free logic include $\exists x\ \ x = x$ and $[\forall x\ \ P(x)] \Rightarrow [\exists x\ \ P(x)]$. More importantly, perhaps, the equivalence of $\phi \vee \exists x\ \ \psi$ and $\exists x\ \ \phi \vee \psi$ when x does not occur free in ϕ, which is used for putting sentences into CNF, does not hold.

One could argue that $\exists x\ \ x = x$, which simply states that the model is nonempty, is not naturally a valid sentence, and that it ought to be possible to contemplate a universe with no objects. However, experience has shown that free logic seems to require extra work to rule out the empty model in many commonly occurring cases of logical representation and reasoning.

8.8 The fact $\neg Spouse(George, Laura)$ does not follow. We need to assert that at most one person can be the spouse of any given person:

$$\forall x, y, z\ \ Spouse(x, z) \wedge Spouse(y, z) \Rightarrow x = y\ .$$

With this axiom, a resolution proof of $\neg Spouse(George, Laura)$ is straightforward.

If $Spouse$ is a unary function symbol, then the question is whether $\neg Spouse(Laura) = George$ follows from $Jim \neq George$ and $Spouse(Laura) = Jim$. The answer is yes, it does follow. They could not both be the value of the function applied to the same argument if they were different objects.

8.9

a. Paris and Marseilles are both in France.

(i) $In(Paris \wedge Marseilles, France)$.
(2) Syntactically invalid. Cannot use conjunction inside a term.

(ii) $In(Paris, France) \wedge In(Marseilles, France)$.
(1) Correct.

(iii) $In(Paris, France) \vee In(Marseilles, France)$.
(3) Incorrect. Disjunction does not express "both."

b. There is a country that borders both Iraq and Pakistan.

(i) $\exists\, c \quad Country(c) \wedge Border(c, Iraq) \wedge Border(c, Pakistan)$.
(1) Correct.

(ii) $\exists\, c \quad Country(c) \;\Rightarrow\; [Border(c, Iraq) \wedge Border(c, Pakistan)]$.
(3) Incorrect. Use of implication in existential.

(iii) $[\exists\, c \quad Country(c)] \;\Rightarrow\; [Border(c, Iraq) \wedge Border(c, Pakistan)]$.
(2) Syntactically invalid. Variable c used outside the scope of its quantifier.

(iv) $\exists\, c \quad Border(Country(c), Iraq \wedge Pakistan)$.
(2) Syntactically invalid. Cannot use conjunction inside a term.

c. All countries that border Ecuador are in South America.

(i) $\forall\, c \;\; Country(c) \wedge Border(c, Ecuador) \;\Rightarrow\; In(c, SouthAmerica)$.
(1) Correct.

(ii) $\forall\, c \;\; Country(c) \;\Rightarrow\; [Border(c, Ecuador) \;\Rightarrow\; In(c, SouthAmerica)]$.
(1) Correct. Equivalent to (i).

(iii) $\forall\, c \;\; [Country(c) \;\Rightarrow\; Border(c, Ecuador)] \;\Rightarrow\; In(c, SouthAmerica)$.
(3) Incorrect. The implication in the LHS is effectively an implication in an existential; in particular, it sanctions the RHS for all non-countries.

(iv) $\forall\, c \;\; Country(c) \wedge Border(c, Ecuador) \wedge In(c, SouthAmerica)$.
(3) Incorrect. Uses conjunction as main connective of a universal quantifier.

d. No region in South America borders any region in Europe.

(i) $\neg[\exists\, c, d \;\; In(c, SouthAmerica) \wedge In(d, Europe) \wedge Borders(c, d)]$.
(1) Correct.

(ii) $\forall\, c, d \;\; [In(c, SouthAmerica) \wedge In(d, Europe)] \;\Rightarrow\; \neg Borders(c, d)]$.
(1) Correct.

(iii) $\neg\forall\, c \;\; In(c, SouthAmerica) \;\Rightarrow\; \exists\, d \;\; In(d, Europe) \wedge \neg Borders(c, d)$.
(3) Incorrect. This says there is some country in South America that borders every country in Europe!

(iv) $\forall\, c \;\; In(c, SouthAmerica) \;\Rightarrow\; \forall\, d \;\; In(d, Europe) \;\Rightarrow\; \neg Borders(c, d)$.
(1) Correct.

e. No two adjacent countries have the same map color.

(i) $\forall\, x, y \;\; \neg Country(x) \vee \neg Country(y) \vee \neg Borders(x, y) \vee$
$\neg(MapColor(x) = MapColor(y))$.
(1) Correct.

(ii) $\forall\, x, y \;\; (Country(x) \wedge Country(y) \wedge Borders(x, y) \wedge \neg(x = y)) \;\Rightarrow$
$\neg(MapColor(x) = MapColor(y))$.
(1) Correct. The inequality is unnecessary because no country borders itself.

(iii) $\forall\, x, y \;\; Country(x) \wedge Country(y) \wedge Borders(x, y) \wedge$
$\neg(MapColor(x) = MapColor(y))$.
(3) Incorrect. Uses conjunction as main connective of a universal quantifier.

(iv) $\forall x, y \;\; (Country(x) \wedge Country(y) \wedge Borders(x, y)) \;\Rightarrow\; MapColor(x \neq y)$.
(2) Syntactically invalid. Cannot use inequality inside a term.

8.10

a. $O(E, S) \vee O(E, L)$.
b. $O(J, A) \wedge \exists p \;\; p \neq A \wedge O(J, p)$.
c. $\forall p \;\; O(p, S) \;\Rightarrow\; O(p, D)$.
d. $\neg\exists p \;\; C(J, p) \wedge O(p, L)$.
e. $\exists p \;\; B(p, E) \wedge O(p, L)$.
f. $\exists p \;\; O(p, L) \wedge \forall q \;\; C(q, p) \;\Rightarrow\; O(q, D)$.
g. $\forall p \;\; O(p, S) \;\Rightarrow\; \exists q \;\; O(q, L) \wedge C(p, q)$.

8.11

a. People who speak the same language understand each other.
b. Suppose that an extended interpretation with $x \rightarrow A$ and $y \rightarrow B$ satisfy

$$SpeaksLanguage(x, l) \wedge SpeaksLanguage(y, l)$$

for some l. Then from the second sentence we can conclude $Understands(A, B)$. The extended interpretation with $x \rightarrow B$ and $y \rightarrow A$ also must satisfy

$$SpeaksLanguage(x, l) \wedge SpeaksLanguage(y, l) \;,$$

allowing us to conclude $Understands(B, A)$. Hence, whenever the second sentence holds, the first holds.
c. Let $Understands(x, y)$ mean that x understands y, and let $Friend(x, y)$ mean that x is a friend of y.
(i) It is not completely clear if the English sentence is referring to mutual understanding and mutual friendship, but let us assume that is what is intended:
$\forall x, y \;\; Understands(x, y) \wedge Understands(y, x) \Rightarrow (Friend(x, y) \wedge Friend(y, x))$.
(ii) $\forall x, y, z \;\; Friend(x, y) \wedge Friend(y, z) \;\Rightarrow\; Friend(x, z)$.

8.12 This exercise requires a rewriting similar to the Clark completion of the two Horn clauses:

$$\forall n \;\; NatNum(n) \;\Leftrightarrow\; [n = 0 \vee \exists m \;\; NatNum(m) \wedge n = S(m)] \;.$$

8.13

a. The two implication sentences are

$$\forall s \;\; Breezy(s) \;\Rightarrow\; \exists r \;\; Adjacent(r, s) \wedge Pit(r)$$
$$\forall s \;\; \neg Breezy(s) \;\Rightarrow\; \neg\exists r \;\; Adjacent(r, s) \wedge Pit(r) \;.$$

The converse of the second sentence is

$$\forall s \;\; \exists r \;\; Adjacent(r, s) \wedge Pit(r) \;\Rightarrow\; Breezy(s)$$

which, combined with the first sentence, immediately gives

$$\forall s \;\; Breezy(s) \;\Leftrightarrow\; \exists r \;\; Adjacent(r, s) \wedge Pit(r) \;.$$

b. To say that a pit causes all adjacent squares to be breezy:

$$\forall s \;\; Pit(s) \;\Rightarrow\; [\forall r \;\; Adjacent(r,s) \;\Rightarrow\; Breezy(r)] \; .$$

This axiom allows for breezes to occur spontaneously with no adjacent pits. It would be incorrect to say that a non-pit causes all adjacent squares to be non-breezy, since there might be pits in other squares causing one of the adjacent squares to be breezy. But if *all* adjacent squares have no pits, a square is non-breezy:

$$\forall s \;\; [\forall r \;\; Adjacent(r,s) \;\Rightarrow\; \neg Pit(r)] \;\Rightarrow\; \neg Breezy(s) \; .$$

8.14 Make sure you write definitions with $\Leftrightarrow$. If you use $\Rightarrow$, you are only imposing constraints, not writing a real definition. Note that for aunts and uncles, we include the relations whom the OED says are more strictly defined as aunts-in-law and uncles-in-law, since the latter terms are not in common use.

$$Grandchild(c,a) \;\Leftrightarrow\; \exists b \;\; Child(c,b) \wedge Child(b,a)$$
$$Greatgrandparent(a,d) \;\Leftrightarrow\; \exists b,c \;\; Child(d,c) \wedge Child(c,b) \wedge Child(b,a)$$
$$Ancestor(a,x) \;\Leftrightarrow\; Child(x,a) \vee \exists b \;\; Child(b,a) \wedge Ancestor(b,x)$$
$$Brother(x,y) \;\Leftrightarrow\; Male(x) \wedge Sibling(x,y)$$
$$Sister(x,y) \;\Leftrightarrow\; Female(x) \wedge Sibling(x,y)$$
$$Daughter(d,p) \;\Leftrightarrow\; Female(d) \wedge Child(d,p)$$
$$Son(s,p) \;\Leftrightarrow\; Male(s) \wedge Child(s,p)$$
$$FirstCousin(c,d) \;\Leftrightarrow\; \exists p_1,p_2 \;\; Child(c,p_1) \wedge Child(d,p_2) \wedge Sibling(p_1,p_2)$$
$$BrotherInLaw(b,x) \;\Leftrightarrow\; \exists m \;\; Spouse(x,m) \wedge Brother(b,m)$$
$$SisterInLaw(s,x) \;\Leftrightarrow\; \exists m \;\; Spouse(x,m) \wedge Sister(s,m)$$
$$Aunt(a,c) \;\Leftrightarrow\; \exists p \;\; Child(c,p) \wedge [Sister(a,p) \vee SisterInLaw(a,p)]$$
$$Uncle(u,c) \;\Leftrightarrow\; \exists p \;\; Child(c,p) \wedge [Brother(a,p) \vee BrotherInLaw(a,p)]$$

There are several equivalent ways to define an mth cousin n times removed. One way is to look at the distance of each person to the nearest common ancestor. Define $Distance(c,a)$ as follows:

$$Distance(c,c) = 0$$
$$Child(c,b) \wedge Distance(b,a) = k \;\Rightarrow\; Distance(c,a) = k+1 \; .$$

Thus, the distance to one's grandparent is 2, great-great-grandparent is 4, and so on. Now we have

$$MthCousinNTimesRemoved(c,d,m,n) \;\Leftrightarrow$$
$$\exists a \;\; Distance(c,a) = m+1 \wedge Distance(d,a) = m+n+1 \; .$$

The facts in the family tree are simple: each arrow represents two instances of $Child$ (e.g., $Child(William, Diana)$ and $Child(William, Charles)$), each name represents a sex proposition (e.g., $Male(William)$ or $Female(Diana)$), each "bowtie" symbol indicates a $Spouse$ proposition (e.g., $Spouse(Charles, Diana)$). Making the queries of the logical reasoning system is just a way of debugging the definitions.

8.15 Although these axioms are sufficient to prove set membership when x is in fact a member of a given set, they have nothing to say about cases where x is not a member. For

example, it is not possible to prove that x is not a member of the empty set. These axioms may therefore be suitable for a logical system, such as Prolog, that uses negation-as-failure.

8.16 Here we translate $List?$ to mean "proper list" in Lisp terminology, i.e., a cons structure with Nil as the "rightmost" atom.

$List?(Nil)$
$\forall x, l \;\; List?(l) \;\Leftrightarrow\; List?(Cons(x, l))$
$\forall x, y \;\; First(Cons(x, y)) = x$
$\forall x, y \;\; Rest(Cons(x, y)) = y$
$\forall x \;\; Append(Nil, x) = x$
$\forall v, x, y, z \;\; List?(x) \;\Rightarrow\; (Append(x, y) = z \;\Leftrightarrow\; Append(Cons(v, x), y) = Cons(v, z))$
$\forall x \;\; \neg Find(x, Nil)$
$\forall x \;\; List?(z) \;\Rightarrow\; (Find(x, Cons(y, z)) \;\Leftrightarrow\; (x = y \vee Find(x, z))$

8.17 There are several problems with the proposed definition. It allows one to prove, say, $Adjacent([1,1],[1,2])$ but not $Adjacent([1,2],[1,1])$; so we need an additional symmetry axiom. It does not allow one to prove that $Adjacent([1,1],[1,3])$ is false, so it needs to be written as

$\forall s_1, s_2 \quad \Leftrightarrow \quad \ldots$

Finally, it does not work as the boundaries of the world, so some extra conditions must be added.

8.18 We need the following sentences:

$\forall s_1 \;\; Smelly(s_1) \;\Leftrightarrow\; \exists s_2 \;\; Adjacent(s_1, s_2) \wedge In(Wumpus, s_2)$
$\exists s_1 \;\; In(Wumpus, s_1) \wedge \forall s_2 \;\; (s_1 \neq s_2) \;\Rightarrow\; \neg In(Wumpus, s_2)$.

8.19

a. $\exists x \;\; Parent(Joan, x) \wedge Female(x)$.

b. $\exists^1 x \; Parent(Joan, x) \wedge Female(x)$.

c. $\exists x \;\; Parent(Joan, x) \wedge Female(x) \wedge [\forall y \;\; Parent(Joan, y) \;\Rightarrow\; y = x]$.
(This is sometimes abbreviated "$Female(\iota(x) Parent(Joan, x))$".)

d. $\exists^1 c \; Parent(Joan, c) \wedge Parent(Kevin, c)$.

e. $\exists c \;\; Parent(Joan, c) \wedge Parent(Kevin, c) \wedge \forall d, p \;\; [Parent(Joan, d) \wedge Parent(p, d)] \;\Rightarrow\; [p = Joan \vee p = Kevin]$

8.20

a. $\forall x \;\; Even(x) \;\Leftrightarrow\; \exists y \;\; x = y + y$.

b. $\forall x \;\; Prime(x) \;\Leftrightarrow\; \forall y, z \;\; x = y \times z \;\Rightarrow\; y = 1 \vee z = 1$.

c. $\forall x \;\; Even(x) \;\Rightarrow\; \exists y, z \;\; Prime(y) \wedge Prime(z) \wedge x = y + z$.

8.21 If we have $WA = red$ and $Q = red$ then we could deduce $WA = Q$, which is undesirable to both Western Australians and Queenslanders.

8.22

$$\forall k \;\; Key(k) \;\Rightarrow\; [\exists t_0 \;\; Before(Now, t_0) \wedge \forall t \;\; Before(t_0, t) \;\Rightarrow\; Lost(k, t)]$$
$$\forall s_1, s_2 \;\; Sock(s_1) \wedge Sock(s_2) \wedge Pair(s_1, s_2) \;\Rightarrow$$
$$[\exists t_1 \;\; Before(Now, t_1) \wedge \forall t \;\; Before(t_1, t) \;\Rightarrow\; Lost(s_1, t)] \vee$$
$$[\exists t_2 \;\; Before(Now, t_2) \wedge \forall t \;\; Before(t_2, t) \;\Rightarrow\; Lost(s_2, t)] \; .$$

Notice that the disjunction allows for both socks to be lost, as the English sentence implies.

8.23

a. "No two people have the same social security number."

$$\neg\exists x, y, n \;\; Person(x) \wedge Person(y) \;\Rightarrow\; [HasSS\#(x, n) \wedge HasSS\#(y, n)].$$

This uses $\Rightarrow$ with $\exists$. It also says that no person has a social security number because it doesn't restrict itself to the cases where x and y are not equal. Correct version:

$$\neg\exists x, y, n \;\; Person(x) \wedge Person(y) \wedge \neg(x = y) \wedge [HasSS\#(x, n) \wedge HasSS\#(y, n)]$$

b. "John's social security number is the same as Mary's."

$$\exists n \;\; HasSS\#(John, n) \wedge HasSS\#(Mary, n).$$

This is OK.

c. "Everyone's social security number has nine digits."

$$\forall x, n \;\; Person(x) \;\Rightarrow\; [HasSS\#(x, n) \wedge Digits(n, 9)].$$

This says that everyone has every number. $HasSS\#(x, n)$ should be in the premise:

$$\forall x, n \;\; Person(x) \wedge HasSS\#(x, n) \;\Rightarrow\; Digits(n, 9)$$

d. Here $SS\#(x)$ denotes the social security number of x. Using a function enforces the rule that everyone has just one.

$$\neg\exists x, y \;\; Person(x) \wedge Person(y) \;\Rightarrow\; [SS\#(x) = SS\#(y)]$$
$$SS\#(John) = SS\#(Mary)$$
$$\forall x \;\; Person(x) \;\Rightarrow\; Digits(SS\#(x), 9)$$

8.24 In this exercise, it is best not to worry about details of tense and larger concerns with consistent ontologies and so on. The main point is to make sure students understand connectives and quantifiers and the use of predicates, functions, constants, and equality. Let the basic vocabulary be as follows:
$Takes(x, c, s)$: student x takes course c in semester s;
$Passes(x, c, s)$: student x passes course c in semester s;
$Score(x, c, s)$: the score obtained by student x in course c in semester s;
$x > y$: x is greater than y;
F and G: specific French and Greek courses (one could also interpret these sentences as referring to *any* such course, in which case one could use a predicate $Subject(c, f)$ meaning that the subject of course c is field f;
$Buys(x, y, z)$: x buys y from z (using a binary predicate with unspecified seller is OK but

less felicitous);
$Sells(x, y, z)$: x sells y to z;
$Shaves(x, y)$: person x shaves person y
$Born(x, c)$: person x is born in country c;
$Parent(x, y)$: x is a parent of y;
$Citizen(x, c, r)$: x is a citizen of country c for reason r;
$Resident(x, c)$: x is a resident of country c;
$Fools(x, y, t)$: person x fools person y at time t;
$Student(x)$, $Person(x)$, $Man(x)$, $Barber(x)$, $Expensive(x)$, $Agent(x)$, $Insured(x)$, $Smart(x)$, $Politician(x)$: predicates satisfied by members of the corresponding categories.

a. Some students took French in spring 2001.
$\exists x \;\; Student(x) \wedge Takes(x, F, Spring2001)$.

b. Every student who takes French passes it.
$\forall x, s \;\; Student(x) \wedge Takes(x, F, s) \;\Rightarrow\; Passes(x, F, s)$.

c. Only one student took Greek in spring 2001.
$\exists x \;\; Student(x) \wedge Takes(x, G, Spring2001) \wedge \forall y \;\; y \neq x \Rightarrow \neg Takes(y, G, Spring2001)$.

d. The best score in Greek is always higher than the best score in French.
$\forall s \;\; \exists x \;\; \forall y \;\; Score(x, G, s) > Score(y, F, s)$.

e. Every person who buys a policy is smart.
$\forall x \;\; Person(x) \wedge (\exists y, z \;\; Policy(y) \wedge Buys(x, y, z)) \;\Rightarrow\; Smart(x)$.

f. No person buys an expensive policy.
$\forall x, y, z \;\; Person(x) \wedge Policy(y) \wedge Expensive(y) \;\Rightarrow\; \neg Buys(x, y, z)$.

g. There is an agent who sells policies only to people who are not insured.
$\exists x \;\; Agent(x) \wedge \forall y, z \;\; Policy(y) \wedge Sells(x, y, z) \;\Rightarrow\; (Person(z) \wedge \neg Insured(z))$.

h. There is a barber who shaves all men in town who do not shave themselves.
$\exists x \;\; Barber(x) \wedge \forall y \;\; Man(y) \wedge \neg Shaves(y, y) \;\Rightarrow\; Shaves(x, y)$.

i. A person born in the UK, each of whose parents is a UK citizen or a UK resident, is a UK citizen by birth.
$\forall x \;\; Person(x) \wedge Born(x, UK) \wedge (\forall y \;\; Parent(y, x) \Rightarrow ((\exists r \;\; Citizen(y, UK, r)) \vee Resident(y, UK))) \;\Rightarrow\; Citizen(x, UK, Birth)$.

j. A person born outside the UK, one of whose parents is a UK citizen by birth, is a UK citizen by descent.

$$\forall x \;\; Person(x) \wedge \neg Born(x, UK) \wedge (\exists y \;\; Parent(y, x) \wedge Citizen(y, UK, Birth)) \;\Rightarrow\; Citizen(x, UK, Descent).$$

k. Politicians can fool some of the people all of the time, and they can fool all of the people some of the time, but they can't fool all of the people all of the time.

$$\begin{aligned} &\forall x \;\; Politician(x) \;\Rightarrow \\ &\quad (\exists y \;\; \forall t \;\; Person(y) \wedge Fools(x, y, t)) \wedge \\ &\quad (\exists t \;\; \forall y \;\; Person(y) \;\Rightarrow\; Fools(x, y, t)) \wedge \\ &\quad \neg(\forall t \;\; \forall y \;\; Person(y) \;\Rightarrow\; Fools(x, y, t)) \end{aligned}$$

l. All Greeks speak the same language.

$$\forall x, y, l \;\; Person(x) \wedge [\exists r \;\; Citizen(x, Greece, r)] \wedge Person(y) \wedge [\exists r \;\; Citizen(y, Greece, r)] \\ \wedge Speaks(x, l) \;\Rightarrow\; Speaks(y, l)$$

8.25 This is a very educational exercise but also highly nontrivial. Once students have learned about resolution, ask them to do the proof too. In most cases, they will discover missing axioms. Our basic predicates are $Heard(x, e, t)$ (x heard about event e at time t); $Occurred(e, t)$ (event e occurred at time t); $Alive(x, t)$ (x is alive at time t).

$$\exists t \;\; Heard(W, DeathOf(N), t)$$
$$\forall x, e, t \;\; Heard(x, e, t) \;\Rightarrow\; Alive(x, t)$$
$$\forall x, e, t_2 \;\; Heard(x, e, t_2) \;\Rightarrow\; \exists t_1 \;\; Occurred(e, t_1) \wedge t_1 < t_2$$
$$\forall t_1 \;\; Occurred(DeathOf(x), t_1) \;\Rightarrow\; \forall t_2 \;\; t_1 < t_2 \;\Rightarrow\; \neg Alive(x, t_2)$$
$$\forall t_1, t_2 \;\; \neg(t_2 < t_1) \;\Rightarrow\; ((t_1 < t_2) \vee (t_1 = t_2))$$
$$\forall t_1, t_2, t_3 \;\; (t_1 < t_2) \wedge ((t_2 < t_3) \vee (t_2 = t_3)) \;\Rightarrow\; (t_1 < t_3)$$
$$\forall t_1, t_2, t_3 \;\; ((t_1 < t_2) \vee (t_1 = t_2)) \wedge (t_2 < t_3) \;\Rightarrow\; (t_1 < t_3)$$

8.26 There are three stages to go through. In the first stage, we define the concepts of one-bit and n-bit addition. Then, we specify one-bit and n-bit adder circuits. Finally, we verify that the n-bit adder circuit does n-bit addition.

- One-bit addition is easy. Let Add_1 be a function of three one-bit arguments (the third is the carry bit). The result of the addition is a list of bits representing a 2-bit binary number, least significant digit first:

$$Add_1(0, 0, 0) = [0, 0]$$
$$Add_1(0, 0, 1) = [0, 1]$$
$$Add_1(0, 1, 0) = [0, 1]$$
$$Add_1(0, 1, 1) = [1, 0]$$
$$Add_1(1, 0, 0) = [0, 1]$$
$$Add_1(1, 0, 1) = [1, 0]$$
$$Add_1(1, 1, 0) = [1, 0]$$
$$Add_1(1, 1, 1) = [1, 1]$$

- n-bit addition builds on one-bit addition. Let $Add_n(x_1, x_2, b)$ be a function that takes two lists of binary digits of length n (least significant digit first) and a carry bit (initially 0), and constructs a list of length $n + 1$ that represents their sum. (It will always be exactly $n + 1$ bits long, even when the leading bit is 0—the leading bit is the overflow bit.)

$$Add_n([], [], b) = [b]$$
$$Add_1(b_1, b_2, b) = [b_3, b_4] \;\Rightarrow\; Add_n([b_1|x_1], [b_2|x_2], b) = [b_3|Add_n(x_1, x_2, b_4)]$$

- The next step is to define the structure of a one-bit adder circuit, as given in the text. Let $Add_1Circuit(c)$ be true of any circuit that has the appropriate components and

connections:

$$\begin{aligned}
&\forall c \;\; Add_1Circuit(c) \;\Leftrightarrow \\
&\quad \exists x_1, x_2, a_1, a_2, o_1 \;\; Type(x_1) = Type(x_2) = XOR \\
&\qquad \wedge Type(a_1) = Type(a_2) = AND \wedge Type(o_1) = OR \\
&\qquad \wedge Connected(Out(1, x_1), In(1, x_2)) \wedge Connected(In(1, c), In(1, x_1)) \\
&\qquad \wedge Connected(Out(1, x_1), In(2, a_2)) \wedge Connected(In(1, c), In(1, a_1)) \\
&\qquad \wedge Connected(Out(1, a_2), In(1, o_1)) \wedge Connected(In(2, c), In(2, x_1)) \\
&\qquad \wedge Connected(Out(1, a_1), In(2, o_1)) \wedge Connected(In(2, c), In(2, a_1)) \\
&\qquad \wedge Connected(Out(1, x_2), Out(1, c)) \wedge Connected(In(3, c), In(2, x_2)) \\
&\qquad \wedge Connected(Out(1, o_1), Out(2, c)) \wedge Connected(In(3, c), In(1, a_2))
\end{aligned}$$

Notice that this allows the circuit to have additional gates and connections, but they won't stop it from doing addition.

- Now we define what we mean by an n-bit adder circuit, following the design of Figure 8.6. We will need to be careful, because an n-bit adder is not just an $n-1$-bit adder plus a one-bit adder; we have to connect the overflow bit of the $n-1$-bit adder to the carry-bit input of the one-bit adder. We begin with the base case, where $n = 0$:

$$\begin{aligned}
&\forall c \;\; Add_nCircuit(c, 0) \;\Leftrightarrow \\
&Signal(Out(1, c)) = 0
\end{aligned}$$

Now, for the recursive case we specify that the first connect the "overflow" output of the $n-1$-bit circuit as the carry bit for the last bit:

$$\begin{aligned}
&\forall c, n \;\; n > 0 \;\Rightarrow\; [Add_nCircuit(c, n) \;\Leftrightarrow \\
&\quad \exists c_2, d \;\; Add_nCircuit(c_2, n-1) \wedge Add_1Circuit(d) \\
&\qquad \wedge \forall m \;\; (m > 0) \wedge (m < 2n-1) \;\Rightarrow\; In(m, c) = In(m, c_2) \\
&\qquad \wedge \forall m \;\; (m > 0) \wedge (m < n) \;\Rightarrow\; \wedge Out(m, c) = Out(m, c_2) \\
&\qquad \wedge Connected(Out(n, c_2), In(3, d)) \\
&\qquad \wedge Connected(In(2n-1, c), In(1, d)) \wedge Connected(In(2n, c), In(2, d)) \\
&\qquad \wedge Connected(Out(1, d), Out(n, c)) \wedge Connected(Out(2, d), Out(n+1, c))
\end{aligned}$$

- Now, to verify that a one-bit adder *circuit* actually adds correctly, we ask whether, given any setting of the inputs, the outputs equal the sum of the inputs:

$$\begin{aligned}
&\forall c \;\; Add_1Circuit(c) \;\Rightarrow \\
&\quad \forall i_1, i_2, i_3 \;\; Signal(In(1, c)) = i_1 \wedge Signal(In(2, c)) = i_2 \wedge Signal(In(3, c)) = i_3 \\
&\qquad \Rightarrow \; Add_1(i_1, i_2, i_3) = [Out(1, c), Out(2, c)]
\end{aligned}$$

If this sentence is entailed by the KB, then every circuit with the $Add_1Circuit$ design is in fact an adder. The query for the n-bit can be written as

$$\begin{aligned}
&\forall c, n \;\; Add_nCircuit(c, n) \;\Rightarrow \\
&\quad \forall x_1, x_2, y \;\; InterleavedInputBits(x_1, x_2, c) \wedge OutputBits(y, c) \\
&\qquad \Rightarrow \; Add_n(x_1, x_2, y)
\end{aligned}$$

where $InterleavedInputBits$ and $OutputBits$ are defined appropriately to map bit sequences to the actual terminals of the circuit. [*Note*: this logical formulation has not been tested in a theorem prover and we hesitate to vouch for its correctness.]

8.27 The answers here will vary by country. The two key rules for UK passports are given above.

8.28

a. $W(G,T)$.
b. $\neg W(G,E)$.
c. $W(G,T) \vee W(M,T)$.
d. $\exists\, s \;\; W(J,s)$.
e. $\exists\, x \;\; C(x,R) \wedge O(J,x)$.
f. $\forall\, s \;\; S(M,s,R) \;\Rightarrow\; W(M,s)$.
g. $\neg[\exists\, s \;\; W(G,s) \wedge \exists\, p \;\; S(p,s,R)]$.
h. $\forall\, s \;\; W(G,s) \;\Rightarrow\; \exists\, p,a \;\; S(p,s,a)$.
i. $\exists\, a \;\; \forall\, s \;\; W(J,s) \;\Rightarrow\; \exists\, p \;\; S(p,s,a)$.
j. $\exists\, d,a,s \;\; C(d,a) \wedge O(J,d) \wedge S(B,T,a)$.
k. $\forall\, a \;\; [\exists\, s \;\; S(M,s,a)] \;\Rightarrow\; \exists\, d \;\; C(d,a) \wedge O(J,d)$.
l. $\forall\, a \;\; [\forall\, s,p \;\; S(p,s,a) \;\Rightarrow\; S(B,s,a)] \;\Rightarrow\; \exists\, d \;\; C(d,a) \wedge O(J,d)$.

Solutions for Chapter 9
Inference in First-Order Logic

9.1 We want to show that any sentence of the form $\forall\, v \;\; \alpha$ entails any universal instantiation of the sentence. The sentence $\forall\, v \;\; \alpha$ asserts that α is true in all possible extended interpretations. For any model specifying the referent of ground term g, the truth of $\text{SUBST}(\{v/g\}, \alpha)$ must be identical to the truth value of some extended interpretation in which v is assigned to an object, and in all such interpretations α is true.

EI states: for any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base,

$$\frac{\exists\, v \;\; \alpha}{\text{SUBST}(\{v/k\}, \alpha)} \; .$$

If the knowledge base with the original existentially quantified sentence is KB and the result of EI is KB', then we need to prove that KB is satisfiable iff KB' is satisfiable. Forward: if KB is satisfiable, it has a model M for which an extended interpretation assigning v to some object o renders α true. Hence, we can construct a model M' that satisfies KB' by assigning k to refer to o; since k does not appear elsewhere, the truth values of all other sentences are unaffected. Backward: if KB' is satisfiable, it has a model M' with an assignment for k to some object o. Hence, we can construct a model M that satisfies KB with an extended interpretation assigning v to o; since k does not appear elsewhere, removing it from the model leaves the truth values of all other sentences are unaffected.

9.2 For any sentence α containing a ground term g and for any variable v not occuring in α, we have

$$\frac{\alpha}{\exists\, v \;\; \text{SUBST}^*(\{g/v\}, \alpha)}$$

where SUBST^* is a function that substitutes any or all of the occurrences of g with v. Notice that substituting just one occurrence and applying the rule multiple times is not the same, because it results in a weaker conclusion. For example, $P(a, a)$ should entail $\exists\, x \;\; P(x, x)$ rather than the weaker $\exists\, x, y \;\; P(x, y)$.

9.3 Both b and c are sound conclusions; a is unsound because it introduces the previously-used symbol *Everest*. Note that c does not imply that there are two mountains as high as Everest, because nowhere is it stated that *BenNevis* is different from *Kilimanjaro* (or *Everest*, for that matter).

9.4 This is an easy exercise to check that the student understands unification.

a. $\{x/A, y/B, z/B\}$ (or some permutation of this).

b. No unifier (x cannot bind to both A and B).

c. $\{y/John, x/John\}$.

d. No unifier (because the occurs-check prevents unification of y with $Father(y)$).

9.5

a. For the sentence $Employs(Mother(John), Father(Richard))$, the page isn't wide enough to draw the diagram as in Figure 9.2, so we will draw it with indentation denoting children nodes:

```
[1] Employs(x, y)
  [2] Employs(x, Father(z))
    [3] Employs(x, Father(Richard))
      [4] Employs(Mother(w), Father(Richard))
        [5] Employs(Mother(John), Father(Richard))
    [6] Employs(Mother(w), Father(z))
      [4] ...
      [7] Employs(Mother(John), Father(z))
        [5] ...
  [8] Employs(Mother(w), y)
    [9] Employs(Mother(John), y)
      [10] Employs(Mother(John), Father(z)
        [5] ...
    [6] ...
```

b. For the sentence $Employs(IBM, y)$, the lattice contains $Employs(x, y)$ and $Employs(y, y)$.

c.

9.6 We use a very simple ontology to make the examples easier:

a. $Horse(x) \Rightarrow Mammal(x)$
$Cow(x) \Rightarrow Mammal(x)$
$Pig(x) \Rightarrow Mammal(x)$.

b. $Offspring(x, y) \wedge Horse(y) \Rightarrow Horse(x)$.

c. $Horse(Bluebeard)$.

d. $Parent(Bluebeard, Charlie)$.

e. $Offspring(x, y) \Rightarrow Parent(y, x)$
$Parent(x, y) \Rightarrow Offspring(y, x)$.
(Note we couldn't do $Offspring(x, y) \Leftrightarrow Parent(y, x)$ because that is not in the form expected by Generalized Modus Ponens.)

f. $Mammal(x) \Rightarrow Parent(G(x), x)$ (here G is a Skolem function).

9.7

a. Let $P(x, y)$ be the relation "x is less than y" over the integers. Then $\forall x \;\; \exists y \;\; P(x, y)$ is true but $\exists x \;\; P(x, x)$ is false.

b. Converting the premise to clausal form gives $P(x, Sk0(x))$ and converting the negated goal to clausal form gives $\neg P(q, q)$. If the two formulas can be unified, then these resolve to the null clause.

c. If the premise is represented as $P(x, Sk0)$ and the negated goal has been correctly converted to the clause $\neg P(q, q)$ then these can be resolved to the null clause under the substitution $\{q/Sk0, x/Sk0\}$.

d. Suppose you are given the premise $\exists x \;\; Cat(x)$ and you wish to prove $Cat(Socrates)$. Converting the premise to clausal form gives the clause $Cat(Sk1)$. If this unifies with $Cat(Socrates)$ then you can resolve this with the negated goal $\neg Cat(Socrates)$ to give the null clause.

9.8 Consider a 3-SAT problem of the form

$$(x_{1,1} \vee x_{2,1} \vee \neg x_{3,1}) \wedge (\neg x_{1,2} \vee x_{2,2} \vee x_{3,2}) \vee \ldots$$

We want to rewrite this as a single definite clause of the form

$$A \wedge B \wedge C \wedge \ldots \;\Rightarrow\; Z,$$

along with a few ground clauses. We can do that with the definite clause

$$OneOf(x_{1,1}, x_{2,1}, Not(x_{3,1})) \wedge OneOf(Not(x_{1,2}), x_{2,2}, x_{3,2}) \wedge \ldots \;\Rightarrow\; Solved\,.$$

The key is that any solution to the definite clause has to assign the same value to each occurrence of any given variable, even if the variable is negated in some of the SAT clauses but not others. We also need to define $OneOf$. This can be done concisely as follows:

$OneOf(True, x, y)$
$OneOf(x, True, y)$
$OneOf(x, y, True)$
$OneOf(Not(False), x, y)$
$OneOf(x, Not(False), y)$
$OneOf(x, y, Not(False))$

9.9 This is quite tricky but students should be able to manage if they check each step carefully.

a. (Note: At each resolution, we rename the variables in the rule).

Goal G0: $7 \leq 3 + 9$	Resolve with (8) $\{x1/7, z1/3 + 9\}$.
Goal G1: $7 \leq y1$	Resolve with (4) $\{x2/7, y1/7 + 0\}$. Succeeds.
Goal G2: $7 + 0 \leq 3 + 9$.	Resolve with (8) $\{x3/7 + 0, z3/3 + 9\}$
Goal G3: $7 + 0 \leq y3$	Resolve with (6) $\{x4/7, y4/0, y3/0 + 7\}$ Succeeds.
Goal G4: $0 + 7 \leq 3 + 9$	Resolve with (7) $\{w5/0, x5/7, y5/3, z5/9\}$.
Goal G5: $0 \leq 3$.	Resolve with (1). Succeeds.
Goal G6: $7 \leq 9$.	Resolve with (2). Succeeds.

G4 succeeds
G2 succeeds.
G0 succeeds.

b. From (1),(2), (7) $\{w/0, x/7, y/3, z/9\}$ infer
(9) $0 + 7 \leq 3 + 9$.
From (9), (6), (8) $\{x1/0, y1/7, x2/0 + 7, y2/7 + 0, z2/3 + 9\}$ infer
(10) $7 + 0 \leq 3 + 9$.
($x1, y1$ are renamed variables in (6). $x2, y2, z2$ are renamed variables in (8).)
From (4), (10), (8) $\{x3/7, x4/7, y4/7 + 0, z4/3 + 9\}$ infer
(11) $7 \leq 3 + 9$.
($x3$ is a renamed variable in (4). $x4, y4, z4$ are renamed variables in (8).)

9.10 Surprisingly, the hard part to represent is "who is that man." We want to ask "what relationship does that man have to some known person," but if we represent relations with predicates (e.g., $Parent(x, y)$) then we cannot make the relationship be a variable in first-order logic. So instead we need to reify relationships. We will use $Rel(r, x, y)$ to say that the family relationship r holds between people x and y. Let Me denote me and MrX denote "that man." We will also need the Skolem constants FM for the father of Me and FX for the father of MrX. The facts of the case (put into implicative normal form) are:

(1) $Rel(Sibling, Me, x) \Rightarrow False$
(2) $Male(MrX)$
(3) $Rel(Father, FX, MrX)$
(4) $Rel(Father, FM, Me)$
(5) $Rel(Son, FX, FM)$

We want to be able to show that Me is the only son of my father, and therefore that Me is father of MrX, who is male, and therefore that "that man" is my son. The relevant definitions from the family domain are:

(6) $Rel(Parent, x, y) \wedge Male(x) \Leftrightarrow Rel(Father, x, y)$
(7) $Rel(Son, x, y) \Leftrightarrow Rel(Parent, y, x) \wedge Male(x)$
(8) $Rel(Sibling, x, y) \Leftrightarrow x \neq y \wedge \exists p \; Rel(Parent, p, x) \wedge Rel(Parent, p, y)$
(9) $Rel(Father, x_1, y) \wedge Rel(Father, x_2, y) \Rightarrow x_1 = x_2$

and the query we want is:

(Q) $Rel(r, MrX, y)$

We want to be able to get back the answer $\{r/Son, y/Me\}$. Translating 1-9 and Q into INF

(and negating Q and including the definition of $\neq$) we get:

$(6a)\ Rel(Parent, x, y) \wedge Male(x) \Rightarrow Rel(Father, x, y)$
$(6b)\ Rel(Father, x, y) \Rightarrow Male(x)$
$(6c)\ Rel(Father, x, y) \Rightarrow Rel(Parent, x, y)$
$(7a)\ Rel(Son, x, y) \Rightarrow Rel(Parent, y, x)$
$(7b)\ Rel(Son, x, y) \Rightarrow Male(x))$
$(7c)\ Rel(Parent, y, x) \wedge Male(x) \Rightarrow Rel(Son, x, y)$
$(8a)\ Rel(Sibling, x, y) \Rightarrow x \neq y$
$(8b)\ Rel(Sibling, x, y) \Rightarrow Rel(Parent, P(x, y), x)$
$(8c)\ Rel(Sibling, x, y) \Rightarrow Rel(Parent, P(x, y), y)$
$(8d)\ Rel(Parent, P(x, y), x) \wedge Rel(Parent, P(x, y), y) \wedge x \neq y \Rightarrow Rel(Sibling, x, y)$
$(9)\ Rel(Father, x_1, y) \wedge Rel(Father, x_2, y) \Rightarrow x_1 = x_2$
$(N)\ True \Rightarrow x = y \vee x \neq y$
$(N')\ x = y \wedge x \neq y \Rightarrow False$
$(Q')\ Rel(r, MrX, y) \Rightarrow False$

Note that (1) is non-Horn, so we will need resolution to be be sure of getting a solution. It turns out we also need demodulation to deal with equality. The following lists the steps of the proof, with the resolvents of each step in parentheses:

$(10)\ Rel(Parent, FM, Me)$	$(4, 6c)$
$(11)\ Rel(Parent, FM, FX)$	$(5, 7a)$
$(12)\ Rel(Parent, FM, y) \wedge Me \neq y \Rightarrow Rel(Sibling, Me, y)$	$(10, 8d)$
$(13)\ Rel(Parent, FM, y) \wedge Me \neq y \Rightarrow False$	$(12, 1)$
$(14)\ Me \neq FX \Rightarrow False$	$(13, 11)$
$(15)\ Me = FX$	$(14, N)$
$(16)\ Rel(Father, Me, MrX)$	$(15, 3, demodulation)$
$(17)\ Rel(Parent, Me, MrX)$	$(16, 6c)$
$(18)\ Rel(Son, MrX, Me)$	$(17, 2, 7c)$
$(19)\ False\ \{r/Son, y/Me\}$	$(18, Q')$

9.11 We will give the average-case time complexity for each query/scheme combination in the following table. (An entry of the form "$1;\ n$" means that it is $O(1)$ to find the first solution to the query, but $O(n)$ to find them all.) We make the following assumptions: hash tables give $O(1)$ access; there are n people in the data base; there are $O(n)$ people of any specified age; every person has one mother; there are H people in Houston and T people in Tiny Town; T is much less than n; in Q4, the second conjunct is evaluated first.

	Q1	Q2	Q3	Q4
S1	1	$1;\ H$	$1;\ n$	$T;\ T$
S2	1	$n;\ n$	$1;\ n$	$n;\ n$
S3	n	$n;\ n$	$1;\ n$	$n^2;\ n^2$
S4	1	$n;\ n$	$1;\ n$	$n;\ n$
S5	1	$1;\ H$	$1;\ n$	$T;\ T$

Anything that is $O(1)$ can be considered "efficient," as perhaps can anything $O(T)$. Note

that S1 and S5 dominate the other schemes for this set of queries. Also note that indexing on predicates plays no role in this table (except in combination with an argument), because there are only 3 predicates (which is $O(1)$). It would make a difference in terms of the constant factor.

9.12 This would work if there were no recursive rules in the knowledge base. But suppose the knowledge base contains the sentences:

$$Member(x, [x|r])$$
$$Member(x, r) \Rightarrow Member(x, [y|r])$$

Now take the query $Member(3, [1, 2, 3])$, with a backward chaining system. We unify the query with the consequent of the implication to get the substitution $\theta = \{x/3, y/1, r/[2, 3]\}$. We then substitute this in to the left-hand side to get $Member(3, [2, 3])$ and try to back chain on that with the substitution θ. When we then try to apply the implication again, we get a failure because y cannot be both 1 and 2. In other words, the failure to standardize apart causes failure in some cases where recursive rules would result in a solution if we did standardize apart.

9.13 This questions deals with the subject of looping in backward-chaining proofs. A loop is bound to occur whenever a subgoal arises that is a substitution instance of one of the goals on the stack. Not all loops can be caught this way, of course, otherwise we would have a way to solve the halting problem.

a. The proof tree is shown in Figure S9.1. The branch with $Offspring(Bluebeard, y)$ and $Parent(y, Bluebeard)$ repeats indefinitely, so the rest of the proof is never reached.

b. We get an infinite loop because of rule **b**, $Offspring(x, y) \wedge Horse(y) \Rightarrow Horse(x)$. The specific loop appearing in the figure arises because of the ordering of the clauses—it would be better to order $Horse(Bluebeard)$ before the rule from **b**. However, a loop will occur no matter which way the rules are ordered if the theorem-prover is asked for all solutions.

c. One should be able to prove that both Bluebeard and Charlie are horses.

d. Smith *et al.* (1986) recommend the following method. Whenever a "looping" goal occurs (one that is a substitution instance of a supergoal higher up the stack), suspend the attempt to prove that subgoal. Continue with all other branches of the proof for the supergoal, gathering up the solutions. Then use those solutions (suitably instantiated if necessary) as solutions for the suspended subgoal, continuing that branch of the proof to find additional solutions if any. In the proof shown in the figure, the $Offspring(Bluebeard, y)$ is a repeated goal and would be suspended. Since no other way to prove it exists, that branch will terminate with failure. In this case, Smith's method is sufficient to allow the theorem-prover to find both solutions.

9.14 Here is a goal tree:

```
goals = [Criminal(West)]
  goals = [American(West), Weapon(y), Sells(West, y, z), Hostile(z)]
  goals = [Weapon(y), Sells(West, y, z), Hostile(z)]
```

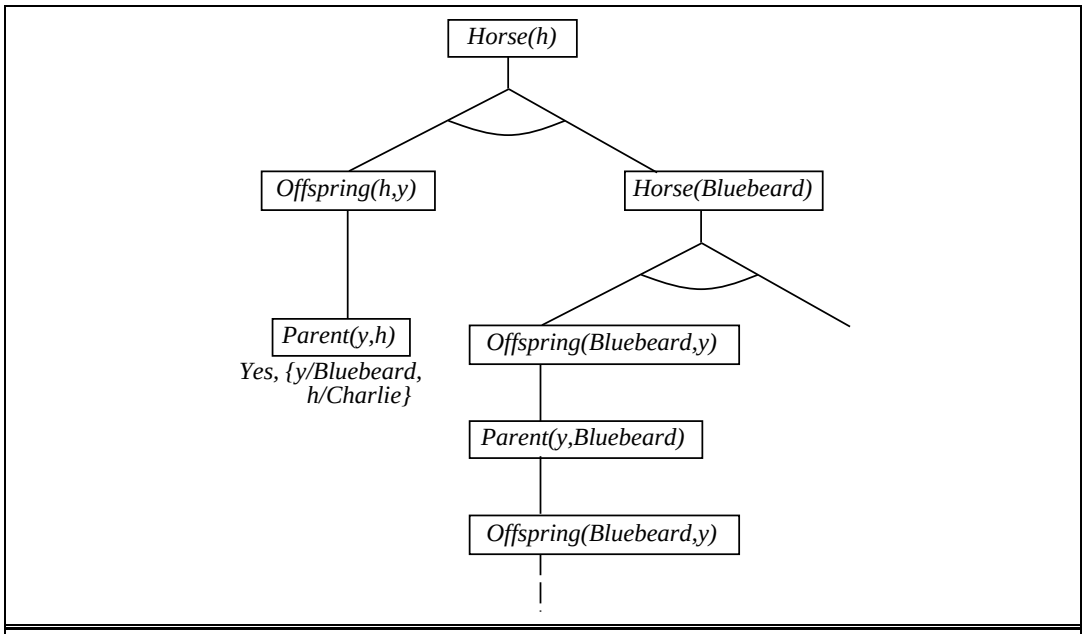


Figure S9.1 Partial proof tree for finding horses.

```
    goals = [Missle(y), Sells(West, y, z), Hostile(z)]
    goals = [Sells(West, M1, z), Hostile(z)]
      goals = [Missle(M1), Owns(Nono, M1), Hostile(Nono)]
      goals = [Owns(Nono, M1), Hostile(Nono)]
    goals = [Hostile(Nono)]
goals = []
```

9.15

a. In the following, an indented line is a step deeper in the proof tree, while two lines at the same indentation represent two alternative ways to prove the goal that is unindented above it. `P1` and `P2` refer to the first and second clauses of the definition respectively. We show each goal as it is generated and the result of unifying it with the head of each clause.

```
P(A, [2,1,3])                  goal
  P(2, [2|[1,3]])              unify with head of P1
    => solution, with A = 2
  P(A, [2|[1,3]])              unify with head of P2
    P(A, [1,3])                subgoal
      P(1, [1,3])              unify with head of P1
        => solution, with A = 1
      P(A, [1|[3]])            unify with head of P2
        P(A, [3])              subgoal
          P(3, [3|[]])         unify with head of P1
            => solution, with A = 3
          P(A, [3|[]])         unify with head of P2
```

```
            P(A, [])            subgoal (fails)

P(2, [1,A,3])                   goal
  P(2, [1|[A,3]])               unify with head of P2
    P(2, [A,3])                 subgoal
      P(2, [2,3])               unify with head of P1
        => solution, with A = 2
      P(2, [A|[3]])             unify with head of P2
        P(2, [3])               subgoal
          P(2, [3|[]])          unify with head of P2
            P(2, [])            subgoal
```

b. `P` could better be called `Member`; it succeeds when the first argument is an element of the list that is the second argument.

9.16 The different versions of `sort` illustrate the distinction between logical and procedural semantics in Prolog.

a.
```
sorted([]).
sorted([X]).
sorted([X,Y|L]) :- X<Y, sorted([Y|L]).
```

b.
```
perm([],[]).
perm([X|L],M) :-
  delete(X,M,M1),
  perm(L,M1).

delete(X,[X|L],L).       %% deleting an X from [X|L] yields L
delete(X,[Y|L],[Y|M]) :- delete(X,L,M).

member(X,[X|L]).
member(X,[_|L]) :- member(X,L).
```

c.
```
sort(L,M) :- perm(L,M), sorted(M).
```
This is about as close to an executable formal specification of sorting as you can get—it says the absolute minimum about what sort means: in order for `M` to be a sort of `L`, it must have the same elements as `L`, and they must be in order.

d. Unfortunately, this doesn't fare as well as a program as it does as a specification. It is a generate-and-test sort: `perm` generates candidate permutations one at a time, and `sorted` tests them. In the worst case (when there is only one sorted permutation, and it is the last one generated), this will take $O(n!)$ generations. Since each `perm` is $O(n^2)$ and each `sorted` is $O(n)$, the whole `sort` is $O(n!n^2)$ in the worst case.

e. Here's a simple insertion sort, which is $O(n^2)$:
```
isort([],[]).
isort([X|L],M) :- isort(L,M1), insert(X,M1,M).

insert(X,[],[X]).
insert(X,[Y|L],[X,Y|L]) :- X=<Y.
insert(X,[Y|L],[Y|M]) :- Y<X, insert(X,L,M).
```

9.17 This exercise illustrates the power of pattern-matching, which is built into Prolog.

a. The code for simplification looks straightforward, but students may have trouble finding the middle way between undersimplifying and looping infinitely.

```
simplify(X,X) :- primitive(X).
simplify(X,Y) :- evaluable(X), Y is X.
simplify(Op(X)) :- simplify(X,X1), simplify_exp(Op(X1)).
simplify(Op(X,Y)) :- simplify(X,X1), simplify(Y,Y1), simplify_exp(Op(X1,Y1)).

simplify_exp(X,Y) :- rewrite(X,X1), simplify(X1,Y).
simplify_exp(X,X).

primitive(X) :- atom(X).
```

b. Here are a few representative rewrite rules drawn from the extensive list in Norvig (1992).

```
Rewrite(X+0,X).
Rewrite(0+X,X).
Rewrite(X+X,2*X).
Rewrite(X*X,X^2).
Rewrite(X^0,1).
Rewrite(0^X,0).
Rewrite(X*N,N*X) :- number(N).
Rewrite(ln(e^X),X).
Rewrite(X^Y*X^Z,X^(Y+Z)).
Rewrite(sin(X)^2+cos(X)^2,1).
```

c. Here are the rules for differentiation, using `d(Y,X)` to represent the derivative of expression `Y` with respect to variable `X`.

```
Rewrite(d(X,X),1).
Rewrite(d(U,X),0) :- atom(U), U /= X.
Rewrite(d(U+V,X),d(U,X)+d(V,X)).
Rewrite(d(U-V,X),d(U,X)-d(V,X)).
Rewrite(d(U*V,X),V*d(U,X)+U*d(V,X)).
Rewrite(d(U/V,X),(V*d(U,X)-U*d(V,X))/(V^2)).
Rewrite(d(U^N,X),N*U^(N-1)*d(U,X)) :- number(N).
Rewrite(d(log(U),X),d(U,X)/U).
Rewrite(d(sin(U),X),cos(U)*d(U,X)).
Rewrite(d(cos(U),X),-sin(U)*d(U,X)).
Rewrite(d(e^U,X),d(U,X)*e^U).
```

9.18 Once you understand how Prolog works, the answer is easy:

```
solve(X,[X]) :- goal(X).
solve(X,[X|P]) :- successor(X,Y), solve(Y,P).
```

We could render this in English as "Given a start state, if it is a goal state, then the path consisting of just the start state is a solution. Otherwise, find some successor state such that there is a path from the successor to the goal; then a solution is the start state followed by that path."

Notice that `solve` can not only be used to find a path `P` that is a solution, it can also be used to verify that a given path is a solution.

If you want to add heuristics (or even breadth-first search), you need an explicit queue. The algorithms become quite similar to the versions written in Lisp or Python or Java or in pseudo-code in the book.

9.19

a. Results from forward chaining:

(i) $Ancestor(Mother(y), John)$: Yes, $\{y/John\}$ (immediate).

(ii) $Ancestor(Mother(Mother(y)), John)$: Yes, $\{y/John\}$ (second iteration).

(iii) $Ancestor(Mother(Mother(Mother(y))), Mother(y))$: Yes, $\{\}$ (second iteration).

(iv) $Ancestor(Mother(John), Mother(Mother(John)))$: Does not terminate.

b. Although resolution is complete, it cannot prove this because it does not follow. Nothing in the axioms rules out the possibility of everything being the ancestor of everything else.

c. Same answer.

9.20

a. $\exists p \ \forall q \ \ S(p, q) \ \Leftrightarrow \ \neg S(q, q)$.

b. There are two clauses, corresponding to the two directions of the implication.
C1: $\neg S(Sk1, q) \vee \neg S(q, q)$.
C2: $S(Sk1, q) \vee S(q, q)$.

c. Applying factoring to C1, using the substitution $q/Sk1$ gives:
C3: $\neg S(Sk1, Sk1)$.
Applying factoring to C2, using the substitution $q/Sk1$ gives:
C4: $S(Sk1, Sk1)$.
Resolving C3 with C4 gives the null clause.

9.21 This question tests both the student's understanding of resolution and their ability to think at a high level about relations among sets of sentences. Recall that resolution allows one to show that $KB \models \alpha$ by proving that $KB \wedge \neg\alpha$ is inconsistent. Suppose that in general the resolution system is called using $\text{ASK}(KB, \alpha)$. Now we want to show that a given sentence, say β is valid or unsatisfiable.

A sentence β is valid if it can be shown to be true without additional information. We check this by calling $\text{ASK}(KB_0, \beta)$ where KB_0 is the empty knowledge base.

A sentence β that is unsatisfiable is inconsistent by itself. So if we empty the knowledge base again and call $\text{ASK}(KB_0, \neg\beta)$ the resolution system will attempt to derive a contradiction starting from $\neg\neg\beta$. If it can do so, then it must be that $\neg\neg\beta$, and hence β, is inconsistent.

9.22 There are two ways to do this: one literal in one clause that is complementary to two different literals in the other, such as

$$P(x) \quad \neg P(a) \vee \neg P(b)$$

or two complementary pairs of literals, such as

$$P(x) \vee Q(x) \quad \neg P(a) \vee \neg Q(b) \ .$$

Note that this does not work in propositional logic: in the first case, the two literals in the second clause would have to be identical; in the second case, the remaining unresolved complementary pair after resolution would render the result a tautology.

9.23 This is a form of inference used to show that Aristotle's syllogisms could not capture all sound inferences.

a. $\forall x \;\; Horse(x) \Rightarrow Animal(x)$
$\forall x, h \;\; Horse(x) \wedge HeadOf(h, x) \Rightarrow \exists y \;\; Animal(y) \wedge HeadOf(h, y)$

b. $A. \neg Horse(x) \vee Animal(x)$
$B. \; Horse(G)$
$C. \; HeadOf(H, G)$
$D. \neg Animal(y) \vee \neg HeadOf(H, y)$
(Here $A.$ comes from the first sentence in **a.** while the others come from the second. H and G are Skolem constants.)

c. Resolve D and C to yield $\neg Animal(G)$. Resolve this with A to give $\neg Horse(G)$. Resolve this with B to obtain a contradiction.

9.24 This exercise tests the students' understanding of models and implication.

a. (A) translates to "For every natural number there is some other natural number that is smaller than or equal to it." (B) translates to "There is a particular natural number that is smaller than or equal to any natural number."

b. Yes, (A) is true under this interpretation. You can always pick the number itself for the "some other" number.

c. Yes, (B) is true under this interpretation. You can pick 0 for the "particular natural number."

d. No, (A) does not logically entail (B).

e. Yes, (B) logically entails (A).

f. We want to try to prove via resolution that (A) entails (B). To do this, we set our knowledge base to consist of (A) and the negation of (B), which we will call (-B), and try to derive a contradiction. First we have to convert (A) and (-B) to canonical form. For (-B), this involves moving the $\neg$ in past the two quantifiers. For both sentences, it involves introducing a Skolem function:

(A) $x \geq F_1(x)$
(-B) $\neg F_2(y) \geq y$

Now we can try to resolve these two together, but the occurs check rules out the unification. It looks like the substitution should be $\{x/F_2(y), \; y/F_1(x)\}$, but that is equivalent to $\{x/F_2(y), \; y/F_1(F_2(y))\}$, which fails because y is bound to an expression containing y. So the resolution fails, there are no other resolution steps to try, and therefore (B) does not follow from (A).

g. To prove that (B) entails (A), we start with a knowledge base containing (B) and the negation of (A), which we will call (-A):

(-A) $\neg F_1 \geq y$
(B) $x \geq F_2$

This time the resolution goes through, with the substitution $\{x/F_1,\ y/F_2\}$, thereby yielding $False$, and proving that (B) entails (A).

9.25 One way of seeing this is that resolution allows reasoning by cases, by which we can prove C by proving that either A or B is true, without knowing which one. If the query contains a variable, we cannot prove that any *particular* instantiation gives a fact that is entailed. With definite clauses, we always have a single chain of inference, for which we can follow the chain and instantiate variables; the solution is always a single MGU.

9.26 Not exactly. Part of the definition of algorithm is that it must terminate. Since there can be an infinite number of consequences of a set of sentences, no algorithm can generate them all. Another way to see that the answer is no is to remember that entailment for FOL is semidecidable. If there were an algorithm that generates the set of consequences of a set of sentences S, then when given the task of deciding if B is entailed by S, one could just check if B is in the generated set. But we know that this is not possible, therefore generating the set of sentences is impossible.

If we relax the definition of "algorithm" to allow for programs that *enumerate* the consequences, in the same sense that a program can enumerate the natural numbers by printing them out in order, the answer is yes. For example, we can enumerate them in order of the deepest allowable nesting of terms in the proof.

Solutions for Chapter 10
Classical Planning

10.1 Both problem solver and planner are concerned with getting from a start state to a goal using a set of defined operations or actions, typically in a deterministic, discrete, observable environment. In planning, however, we open up the representation of states, goals, and plans, which allows for a wider variety of algorithms that decompose the search space, search forwards or backwards, and use automated generation of heuristic functions.

10.2 This is an easy exercise, the point of which is to understand that "applicable" means satisfying the preconditions, and that a concrete action instance is one with the variables replaced by constants. The applicable actions are:

$$Fly(P_1, JFK, SFO)$$
$$Fly(P_1, JFK, JFK)$$
$$Fly(P_2, SFO, JFK)$$
$$Fly(P_2, SFO, SFO)$$

A minor point of this is that the action of flying nowhere—from one airport to itself—is allowable by the definition of Fly, and is applicable (if not useful).

10.3 This exercise is intended as a fairly easy exercise in describing a domain.

a. The initial state is:

$$At(Monkey, A) \wedge At(Bananas, B) \wedge At(Box, C) \wedge$$
$$Height(Monkey, Low) \wedge Height(Box, Low) \wedge Height(Bananas, High) \wedge$$
$$Pushable(Box) \wedge Climbable(Box)$$

b. The actions are:

$Action($ACTION:$Go(x, y)$, PRECOND:$At(Monkey, x)$,
EFFECT:$At(Monkey, y) \wedge \neg(At(Monkey, x)))$
$Action($ACTION:$Push(b, x, y)$, PRECOND:$At(Monkey, x) \wedge Pushable(b)$,
EFFECT:$At(b, y) \wedge At(Monkey, y) \wedge \neg At(b, x) \wedge \neg At(Monkey, x))$
$Action($ACTION:$ClimbUp(b)$,
PRECOND:$At(Monkey, x) \wedge At(b, x) \wedge Climbable(b)$,
EFFECT:$On(Monkey, b) \wedge \neg Height(Monkey, High))$
$Action($ACTION:$Grasp(b)$,
PRECOND:$Height(Monkey, h) \wedge Height(b, h)$
$\wedge At(Monkey, x) \wedge At(b, x)$,
EFFECT:$Have(Monkey, b))$
$Action($ACTION:$ClimbDown(b)$,
PRECOND:$On(Monkey, b) \wedge Height(Monkey, High)$,
EFFECT:$\neg On(Monkey, b) \wedge \neg Height(Monkey, High)$
$\wedge Height(Monkey, Low)$
$Action($ACTION:$UnGrasp(b)$, PRECOND:$Have(Monkey, b)$,
EFFECT:$\neg Have(Monkey, b))$

c. In situation calculus, the goal is a state s such that:

$$Have(Monkey, Bananas, s) \wedge (\exists x \;\; At(Box, x, s_0) \wedge At(Box, x, s))$$

In STRIPS, we can only talk about the goal state; there is no way of representing the fact that there must be some relation (such as equality of location of an object) between two states within the plan. So there is no way to represent this goal.

d. Actually, we did include the $Pushable$ precondition in the solution above.

10.4 The actions are quite similar to the monkey and bananas problem—you should probably assign only one of these two problems. The actions are:

$Action($ACTION:$Go(x, y)$, PRECOND:$At(Shakey, x) \wedge In(x, r) \wedge In(y, r)$,
EFFECT:$At(Shakey, y) \wedge \neg(At(Shakey, x)))$
$Action($ACTION:$Push(b, x, y)$, PRECOND:$At(Shakey, x) \wedge Pushable(b)$,
EFFECT:$At(b, y) \wedge At(Shakey, y) \wedge \neg At(b, x) \wedge \neg At(Shakey, x))$
$Action($ACTION:$ClimbUp(b)$, PRECOND:$At(Shakey, x) \wedge At(b, x) \wedge Climbable(b)$,
EFFECT:$On(Shakey, b) \wedge \neg On(Shakey, Floor))$
$Action($ACTION:$ClimbDown(b)$, PRECOND:$On(Shakey, b)$,
EFFECT:$On(Shakey, Floor) \wedge \neg On(Shakey, b))$
$Action($ACTION:$TurnOn(l)$, PRECOND:$On(Shakey, b) \wedge At(Shakey, x) \wedge At(l, x)$,
EFFECT:$TurnedOn(l))$
$Action($ACTION:$TurnOff(l)$, PRECOND:$On(Shakey, b) \wedge At(Shakey, x) \wedge At(l, x)$,
EFFECT:$\neg TurnedOn(l))$

The initial state is:

$In(Switch_1, Room_1) \wedge In(Door_1, Room_1) \wedge In(Door_1, Corridor)$
$In(Switch_1, Room_2) \wedge In(Door_2, Room_2) \wedge In(Door_2, Corridor)$
$In(Switch_1, Room_3) \wedge In(Door_3, Room_3) \wedge In(Door_3, Corridor)$
$In(Switch_1, Room_4) \wedge In(Door_4, Room_4) \wedge In(Door_4, Corridor)$
$In(Shakey, Room_3) \wedge At(Shakey, X_S)$
$In(Box_1, Room_1) \wedge In(Box_2, Room_1) \wedge In(Box_3, Room_1) \wedge In(Box_4, Room_1)$
$Climbable(Box_1) \wedge Climbable(Box_2) \wedge Climbable(Box_3) \wedge Climbable(Box_4)$
$Pushable(Box_1) \wedge Pushable(Box_2) \wedge Pushable(Box_3) \wedge Pushable(Box_4)$
$At(Box_1, X_1) \wedge At(Box_2, X_2) \wedge At(Box_3, X_3) \wedge At(Box_4, X_4)$
$TurnwdOn(Switch_1) \wedge TurnedOn(Switch_4)$

A plan to achieve the goal is:

$Go(X_S, Door_3)$
$Go(Door_3, Door_1)$
$Go(Door_1, X_2)$
$Push(Box_2, X_2, Door_1)$
$Push(Box_2, Door_1, Door_2)$
$Push(Box_2, Door_2, Switch_2)$

10.5 One representation is as follows. We have the predicates:

a. $HeadAt(c)$: tape head at cell location c, true for exactly one cell.

b. $State(s)$: machine state is s, true for exactly one cell.

c. $ValueOf(c, v)$: cell c's value is v.

d. $LeftOf(c_1, c_2)$: cell c_1 is one step left from cell c_2.

e. $TransitionLeft(s_1, v_1, s_2, v_2)$: the machine in state s_1 upon reading a cell with value v_1 may write value v_2 to the cell, change state to s_2, and transition to the left.

f. $TransitionRight(s_1, v_1, s_2, v_2)$: the machine in state s_1 upon reading a cell with value v_1 may write value v_2 to the cell, change state to s_2, and transition to the right.

The predicates $HeadAt$, $State$, and $ValueOf$ are fluents, the rest are constant descriptions of the machine and its tape. Two actions are required:

$Action(RunLeft(s_1, c_1, v_1, , s_2, c_2, v_2),$
PRECOND: $State(s_1) \wedge HeadAt(c_1) \wedge ValueOf(c_1, v_1)$
; $\wedge TransitionLeft(s_1, v_1, s_2, v_2) \wedge LeftOf(c_2, c_1)$
EFFECT: $\neg State(s_1) \wedge State(s_2) \wedge \neg HeadAt(c_1) \wedge HeadAt(c_2)$
$\wedge \neg ValueOf(c_1, v_1) \wedge ValueOf(c_1, v_2))$

$Action(RunRight(s_1, c_1, v_1, , s_2, c_2, v_2),$
PRECOND: $State(s_1) \wedge HeadAt(c_1) \wedge ValueOf(c_1, v_1)$
; $\wedge TransitionRight(s_1, v_1, s_2, v_2) \wedge LeftOf(c_1, c_2)$
EFFECT: $\neg State(s_1) \wedge State(s_2) \wedge \neg HeadAt(c_1) \wedge HeadAt(c_2)$
$\wedge \neg ValueOf(c_1, v_1) \wedge ValueOf(c_1, v_2))$

The goal will typically be to reach a fixed accept state. A simple example problem is:

$$\begin{aligned}&\mathit{Init}(\mathit{HeadAt}(C_0) \wedge \mathit{State}(S_1) \wedge \mathit{ValueOf}(C_0, 1) \wedge \mathit{ValueOf}(C_1, 1)\\ &\quad \wedge \mathit{ValueOf}(C_2, 1) \wedge \mathit{ValueOf}(C_3, 0) \wedge \mathit{LeftOf}(C_0, C_1) \wedge \mathit{LeftOf}(C_1, C_2)\\ &\quad \wedge \mathit{LeftOf}(C_2, C_3) \wedge \mathit{TransitionLeft}(S_1, 1, S_1, 0) \wedge \mathit{TransitionLeft}(S_1, 0, S_{\text{accept}}, 0)\\ &\mathit{Goal}(\mathit{State}(S_{\text{accept}}))\end{aligned}$$

Note that the number of literals in a state is linear in the number of cells, which means a polynomial space machine require polynomial state to represent.

10.6 Goals and preconditions can only be positive literals. So a negative effect can only make it harder to achieve a goal (or a precondition to an action that achieves the goal). Therefore, eliminating all negative effects only makes a problem easier. This would *not* be true if negative preconditions and goals were allowed.

10.7 The initial state is:

$$On(B, Table) \wedge On(C, A) \wedge On(A, Table) \wedge Clear(B) \wedge Clear(C)$$

The goal is:

$$On(A, B) \wedge On(B, C)$$

First we'll explain why it is an anomaly for a noninterleaved planner. There are two subgoals; suppose we decide to work on $On(A, B)$ first. We can clear C off of A and then move A on to B. But then there is no way to achieve $On(B, C)$ without undoing the work we have done. Similarly, if we work on the subgoal $On(B, C)$ first we can immediately achieve it in one step, but then we have to undo it to get A on B.

Now we'll show how things work out with an interleaved planner such as POP. Since $On(A, B)$ isn't true in the initial state, there is only one way to achieve it: $Move(A, x, B)$, for some x. Similarly, we also need a $Move(B, x', C)$ step, for some x'. Now let's look at the $Move(A, x, B)$ step. We need to achieve its precondition $Clear(A)$. We could do that either with $Move(b, A, y)$ or with $MoveToTable(b, A)$. Let's assume we choose the latter. Now if we bind b to C, then all of the preconditions for the step $MoveToTable(C, A)$ are true in the initial state, and we can add causal links to them. We then notice that there is a threat: the $Move(B, x', C)$ step threatens the $Clear(C)$ condition that is required by the $MoveToTable$ step. We can resolve the threat by ordering $Move(B, x', C)$ after the $MoveToTable$ step. Finally, notice that all the preconditions for $Move(B, x', C)$ are true in the initial state. Thus, we have a complete plan with all the preconditions satisfied. It turns out there is a well-ordering of the three steps:

$$\begin{aligned}&MoveToTable(C, A)\\ &Move(B, Table, C)\\ &Move(A, Table, B)\end{aligned}$$

10.8 Briefly, the reason is the same as for forward search: in the absence of function symbols, a PDDL state space is finite. Hence any complete search algorithm will be complete for PDDL planning, whether forward or backward.

10.9 The drawing is actually rather complex, and doesn't fit well on this page. Some key things to watch out for: (1) Both *Fly* and *Load* actions are possible at level A_0; the planes

can still fly when empty. (2) Negative effects appear in S_1, and are mutex with their positive counterparts.

10.10

a. Literals are persistent, so if it does not appear in the final level, it never will and never did, and thus cannot be achieved.

b. In a serial planning graph, only one action can occur per time step. The level cost (the level at which a literal first appears) thus represents the minimum number of actions in a plan that might possibly achieve the literal.

10.11 The nature of the relaxed problem is described on p.382.

10.12

a. It is feasible to use bidirectional search, because it is possible to invert the actions. However, most of those who have tried have concluded that biderectional search is generally not efficient, because the forward and backward searches tend to miss each other. This is due to the large state space. A few planners, such as PRODIGY (Fink and Blythe, 1998) have used bidirectional search.

b. Again, this is feasible but not popular. PRODIGY is in fact (in part) a partial-order planner: in the forward direction it keeps a total-order plan (equivalent to a state-based planner), and in the backward direction it maintains a tree-structured partial-order plan.

c. An action A can be added if all the preconditions of A have been achieved by other steps in the plan. When A is added, ordering constraints and causal links are also added to make sure that A appears after all the actions that enabled it and that a precondition is not disestablished before A can be executed. The algorithm does search forward, but it is not the same as forward state-space search because it can explore actions in parallel when they don't conflict. For example, if A has three preconditions that can be satisfied by the non-conflicting actions B, C, and D, then the solution plan can be represented as a single partial-order plan, while a state-space planner would have to consider all $3!$ permutations of B, C, and D.

10.13 A forward state-space planner maintains a partial plan that is a strict linear sequence of actions; the plan refinement operator is to add an applicable action to the end of the sequence, updating literals according to the action's effects.

A backward state-space planner maintains a partial plan that is a reversed sequence of actions; the refinement operator is to add an action to the beginning of the sequence as long as the action's effects are compatible with the state at the beginning of the sequence.

10.14

a. We can illustrate the basic idea using the axiom given. Suppose that $Shoot^t$ is true but $HaveArrow^t$ is false. Then the RHS of the axiom is false, so $HaveArrow^{t+1}$ is false, as we would hope. More generally, if an action precondition is violated, then both $ActionCausesF^t$ and $ActionCausesNotF^t$ are false, so the generic successor-state axiom reduces to

$$F^{t+1} \;\Leftrightarrow\; \mathit{False} \vee (F^t \wedge \mathit{True}) \; .$$

which is the same as saying $F^{t+1} \Leftrightarrow F^t$, i.e., nothing happens.

b. Yes, the plan plus the axioms will entail goal satisfaction; the axioms will copy every fluent across an illegal action and the rest of the plan will still work. Note that goal entailment is trivially achieved if we add precondition axioms, because then the plan is logically inconsistent with the axioms and every sentence is entailed by a contradiction. Precondition axioms are a way to *prevent* illegal actions in satisfiability-based planning methods.

c. No. As written in Section 10.4.2, the sucessor-state axioms preclude proving anything about the outcome of a plan with illegal actions. When $Poss(a, s)$ is false, the axioms say nothing about the situation resulting from the action.

10.15 The main point here is that writing each successor-state axiom correctly requires knowing *all* the actions that might add or delete a given fluent; writing a STRIPS axiom, on the other hand, requires knowing *all* the fluents that a given action might add or delete.

a.
$$Poss(Fly(p, from, to), s) \Leftrightarrow At(p, from, s) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to) .$$

b.
$$Poss(a, s) \Rightarrow (At(p, to, Result(a, s)) \Leftrightarrow (\exists\, from \;\; a = Fly(p, from, to)) \vee (At(p, to, s) \wedge \neg \exists\, new \;\; new \neq to \wedge a = Fly(p, to, new))) .$$

c. We must add the possibility axiom for the new action:

$$Poss(Teleport(p, from, to), s) \Leftrightarrow At(p, from, s) \wedge \neg Warped(p, s) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to) .$$

The successor-state axiom for location must be revised:

$$Poss(a, s) \Rightarrow (At(p, to, Result(a, s)) \Leftrightarrow (\exists\, from \;\; a = Fly(p, from, to)) \vee (\exists\, from \;\; a = Teleport(p, from, to)) \vee (At(p, to, s) \wedge \neg \exists\, new \;\; new \neq to \wedge (a = Fly(p, to, new) \vee a = Teleport(p, to, new)))) .$$

Finally, we must add a successor-state axiom for $Warped$:

$$Poss(a, s) \Rightarrow (Warped(p, Result(a, s)) \Leftrightarrow (\exists\, from, to \;\; a = Teleport(p, from, to)) \vee Warped(p, s)) .$$

d. The basic procedure is essentially given in the description of classical planning as Boolean satisfiability in 10.4.1, except that there is no grounding step, the precondition axioms become definitions of $Poss$ for each action, and the successor-state axioms use the structure given in 10.4.2 with existential quantifiers for all free variables in the actions, as shown in the examples above.

10.16

a. Yes, this will find a plan whenever the normal SATPLAN finds a plan no longer than T_{max}.

b. This will not cause SATPLAN to return an incorrect solution, but it might lead to plans that, for example, achieve and unachieve the goal several times.

c. There is no simple and clear way to induce WALKSAT to find short solutions, because it has no notion of the length of a plan—the fact that the problem is a planning problem is part of the encoding, not part of WALKSAT. But if we are willing to do some rather brutal surgery on WALKSAT, we can achieve shorter solutions by identifying the variables that represent actions and (1) tending to randomly initialize the action variables (particularly the later ones) to false, and (1) preferring to randomly flip an earlier action variable rather than a later one.

Solutions for Chapter 11
Planning and Acting in the Real World

11.1 The simplest extension allows for maintenance goals that hold in the initial state and must remain true throughout the execution of the plan. Safety goals (do no harm) are typically of this form. This extends classical planning problems to allow a maintenance goal. A plan solves the problem if the final state satisfies the regular goals, and all visited states satisfy the maintenance goal.

The life-support example cannot, however, be solved by a finite plan. An extension to infinite plans can capture this, where an infinite plan solves a planning problem if the goal is eventually satisfied by the plan, i.e., there is a point after which the goal is continuously true. Infinite solutions can be described finitely with loops.

For the chandelier example we can allow NoOp actions which do nothing except model the passing of physics. The idea is that a solution will have a finite prefix with an infinite tail (i.e., a loop) of NoOps. This will allow the problem specification to capture the instability of a thrown chandelier, as after a certain number of time steps it would no longer be suspended.

11.2 We first need to specify the primitive actions: for movement we have $Forward(t)$, $TurnLeft(t)$, and $TurnRight(t)$ where t is a truck, and for package delivery we have $Load(p, t)$ and $Unload(p, t)$ where p is a package and t is a truck. These can be given PDDL descriptions in the usual way.

The hierarchy can be built in a number of ways, but one is to use the HLA $Nagivate(t, [x, y])$ to take a truck t to coordinates $[x, y]$, and $Deliver(t, p)$ to deliver package p to its destination with truck t. We assume the fluent $At(o, [x, y])$ for trucks and packages o records their current position $[x, y]$, the predicate $Destination(p, [x', y'])$ gives the package's destination.

This hierarchy (Figure S11.1) encodes the knowledge that trucks can only carry one package at a time, that we need only drop packages off at their destinations not intermediate points, and that we can serialize deliveries (in reality, trucks would move in parallel, but we have no representation for parallel actions here). From a higher-level, the hierarchy says that the planner needs only to choose which trucks deliver which packages in what order, and trucks should navigate given their destinations.

11.3 To simplify the problem, we assume that at most one refinement of a high-level action will be applicable at a given time (not much of a restriction since there is a unique solution).

The algorithm shown below maintains at each point the net preconditions and effects

$Refinement(Deliver(t,\ p)$,
PRECOND: $Truck(t) \wedge Package(p) \wedge At(p,\ [x, y]) \wedge Destination(p,\ [x', y'])$
STEPS: $[Navigate(t,\ [x, y]),\ Load(p, t),\ Navigate(t,\ [x', y']),\ Unload(p, t)]$)
$Refinement(Navigate(t,\ [x, y])$,
PRECOND: $Truck(t) \wedge At(t,\ [x, y])$
STEPS: [])
$Refinement(Navigate(t,\ [x, y])$,
PRECOND: $Truck(t)$
STEPS: $[Forward(t), Navigate(t, [x, y])]$)
$Refinement(Navigate(t,\ [x, y])$,
PRECOND: $Truck(t)$
STEPS: $[TurnLeft(t), Navigate(t, [x, y])]$)
$Refinement(Navigate(t,\ [x, y])$,
PRECOND: $Truck(t)$
STEPS: $[TurnRight(t), Navigate(t, [x, y])]$)

Figure S11.1 Truck hierarchy.

of the prefix of h processed so far. This includes both preconditions and effects of primitive actions, and preconditions of refinements. Note that any literal not in effect is untouched by the prefix currently processed.

```
net_preconditions <- {}
net_effects <- {}
remaining <- [h]

while remaining not empty:
  a <- pop remaining

  if a is primitive:
    add to net_preconditions any precondition of a not in effects
    add to net_effects the effects of action a, first removing any
         complementary literals
  else:
    r <- the unique refinement whose preconditions do not include
         literals negated in net_effect or net_preconditions
    add to net_preconditions any preconditions of r not in effect
    prepend to remaining the sequence of actions in r
```

11.4 We cannot draw any conclusions. Just knowing that the optimistic reachable set is a superset of the goal is no more help than knowing only that it intersects the goal: the optimistic reachable set only guarantees that we cannot reach states outside of it, not that we can reach any of the states inside it. Similarly, the pessimistic reachable set only says we can definitely reach state inside of it, not that we cannot reach states outside of it.

11.5 To simplify, we don't model HLA precondition tests. (Comparing the preconditions to the optimistic and pessimistic descriptions can sometimes determine if preconditions are definitely or definitely not satisfied, respectively, but may be inconclusive.)

The operation to propagate 1-CNF descriptions through descriptions is the same for optimistic and pessimistic descriptions, and is as follows:

```
state <- initial state

for each HLA h in order:
  for each literal in the description of h:
    choose case depending on form of literal:
      +l:             state <- state - {-l} + {l}
      -l:             state <- state - {l} + {-l}
      poss add l:     state <- state + {l}
      poss del l:     state <- state + {-l}
      poss add del l: state <- state + {l,-l}

description <- conjunction of all literals which are
               not part of a complementary pair in state
```

11.6 The natural nondeterministic generalization of DURATION, USE, and CONSUME represents each as an *interval* of possible values rather than a single value. Algorithms that work with quantities can all be modified relatively easily to manage intervals over quantities—for example, by representing them as inequalities for the lower and upper bounds. Thus, if the agent starts with 10 screws and the first action in a plan consumes 2–4 screws, then a second action requiring 5 screws is still executable.

When it comes to conditional effects, however, the fields must be treated differently. The USE field refers to a constraint holding *during* the action, rather than *after* it is done. Thus, it has to remain a separate field, since it is not treated in the same way as an effect. The DURATION and CONSUME fields both describe effects (on the clock and on the quantity of a resource); thus, they can be folded into the conditional effect description for the action.

11.7 We need one action, $Assign$, which assigns the value in the source register (or variable if you prefer, but the term "register" makes it clearer that we are dealing with a physical location) sr to the destination register dr:

$Action($ACTION:$Assign(dr, sr)$,
PRECOND:$Register(dr) \wedge Register(sr) \wedge Value(dr, dv) \wedge Value(sr, sv)$,
EFFECT:$Value(dr, sv) \wedge \neg Value(dr, dv))$

Now suppose we start in an initial state with $Register(R_1) \wedge Register(R_2) \wedge Value(R_1, V_1) \wedge Value(R_2, V_2)$ and we have the goal $Value(R_1, V_2) \wedge Value(R_2, V_1)$. Unfortunately, there is no way to solve this as is. We either need to add an explicit $Register(R_3)$ condition to the initial state, or we need a way to create new registers. That could be done with an action for allocating a new register:

$Action($ACTION:$Allocate(r)$,
EFFECT:$Register(r))$

Then the following sequence of steps constitues a valid plan:

$Allocate(R_3)$
$Assign(R_3, R_1)$
$Assign(R_1, R_2)$
$Assign(R_2, R_1)$

11.8 Flip can be described using conditional effects:

$Action(Flip,$
EFFECT:**when** L: $\neg L \wedge$ **when** $\neg L$: $L)$.

To see that a 1-CNF belief state representation stays 1-CNF after $Flip$, observe that there are three cases. If L is true in the belief state, then it is false after $Flip$; conversely if it is false. Finally, if L is unknown before, then it is unknown after: either L or $\neg L$ can obtain. All other components of the belief state remain unchanged, since it is 1-CNF.

11.9 Using the second definition of $Clear$ in the chapter—namely, that there is a clear space for a block—the only change is that the destination remains clear if it is the table:

$Action(Move(b, x, y),$
PRECOND:$On(b, x) \wedge Clear(b) \wedge Clear(y),$
EFFECT:$On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge$ (**when** $y \neq Table$: $\neg Clear(y)))$

11.10 Let $CleanH$ be true iff the robot's current square is clean and $CleanO$ be true iff the other square is clean. Then $Suck$ is characterized by

$Action(Suck,$ PRECOND:, EFFECT:$CleanH)$

Unfortunately, moving affects these new literals! For $Left$ we have

$Action(Left,$ PRECOND:$AtR,$
EFFECT:$AtL \wedge \neg AtR \wedge$ **when** $CleanH$: $CleanO \wedge$ **when** $CleanO$: $CleanH$
$\wedge$ **when** $\neg CleanO$: $\neg CleanH \wedge$ **when** $\neg CleanH$: $\neg CleanO)$

with the dual for $Right$.

11.11 The main thing to notice here is that the vacuum cleaner moves repeatedly over dirty areas—presumably, until they are clean. Also, each forward move is typically short, followed by an immediate reversing over the same area. This is explained in terms of a disjunctive outcome: the area may be fully cleaned or not, the reversing enables the agent to check, and the repetition ensures completion (unless the dirt is ingrained). Thus, we have a strong cyclic plan with sensing actions.

11.12 One solution plan is $[Test,$ **if** $CultureGrowth$ **then** $[Drink, Medicate]]$.

Solutions for Chapter 12
Knowledge Representation

12.1 Sortal predicates:
$Player(p)$
$Mark(m)$
$Square(q)$

Constants:
Xp, Op: Players.
$X, O, Blank$: Marks.
$Q11, Q12 \ldots Q33$: Squares.
$S0$: Situation.

Atemporal:
$MarkOf(p)$: Function mapping player p to his/her mark.
$Winning(q1, q2, q3)$: Predicate. Squares $q1, q2, q3$ constitute a winning position.
$Opponent(p)$: Function mapping player p to his opponent.

Situation Calculus:
$Result(a, s)$.
$Poss(a, s)$.

State:
$TurnAt(s)$: Function mapping situation s to the player whose turn it is.
$Marked(q, s)$: Function mapping square q and situation s to the mark in q at s.
$Wins(p, s)$. Player p has won in situation s.

Action:
$Play(p, q)$: Function mapping player p and square q to the action of p marking q.

Atemporal axioms:
A1. $MarkOf(Xp) = X$.
A2. $MarkOf(Op) = O$.
A3. $Opponent(Xp) = Op$.
A4. $Opponent(Op) = Xp$.
A5. $\forall p \;\; Player(p) \;\Leftrightarrow\; p = Xp \vee p = Op$.
A6. $\forall m \;\; Mark(m) \;\Leftrightarrow\; m = X \vee m = O \vee m = Blank$.
A7. $\forall q \;\; Square(q) \;\Leftrightarrow\; q = Q11 \vee q = Q12 \vee \ldots \vee q = Q33$.
A8. $\forall q1, q2, q3 \;\; WinningPosition(q1, q2, q3) \;\Leftrightarrow$

$[q1 = Q11 \wedge q2 = Q12 \wedge q3 = Q13] \vee$
$[q1 = Q21 \wedge q2 = Q22 \wedge q3 = Q23] \vee$
... (Similarly for the other six winning positions $\vee$
$[q1 = Q31 \wedge q2 = Q22 \wedge q3 = Q13]$.

Definition of winning:

A9. $\forall p, s \quad Wins(p, s) \Leftrightarrow$
$\exists q1, q2, q3 \quad WinningPosition(q1, q2, q3) \wedge$
$MarkAt(q1, s) = MarkAt(q2, s) = MarkAt(q3, s) = MarkOf(p)$

Causal Axioms:

A10. $\forall p, q \;\; Player(p) \wedge Square(q) \;\Rightarrow$
$MarkAt(q, Result(Play(p, q), s)) = MarkOf(p)$.

A11. $\forall p, a, s \;\; TurnAt(p, s) \;\Rightarrow\; TurnAt(Opponent(p), Result(a, s))$.

Precondition Axiom:

A12. $Poss(Play(p, q), s) \;\Rightarrow\; TurnAt(s) = p \wedge MarkAt(q, s) = Blank$.

Frame Axiom: A13. $q1 \neq q2 \Rightarrow MarkAt(q1, Result(Play(p, q2), s)) = MarkAt(q1, s)$.

Unique names:

A14. $X \neq O \neq Blank$.

(Note: the unique property on players $Xp \neq Op$ follows from A14, A1, and A2.)

A15-A50. For each i, j, k, m between 1 and 3 such that either $i \neq k$ or $j \neq m$ assert the axiom $Qij \neq Qkm$.

Note: In many theories it is useful to posit unique names axioms between entities of different sorts e.g. $\forall p, q \quad Player(p) \wedge Square(q) \;\Rightarrow\; p \neq q$. In this theory these are not actually necessary; if you want to imagine a circumstance in which player Xp is actually the same entity as square $Q23$ or the same as the action $Play(Xp, Q23)$ there is no harm in it.

12.2 This exercise and the following two are rather complex, perhaps suitable for term projects. At this point, we want to strongly urge that you do assign some of these exercises (or ones like them) to give your students a feeling of what it is really like to do knowledge representation. In general, students find classification hierarchies easier than other representation tasks. A recent twist is to compare one's hierarchy with online ones such as `yahoo.com`.

12.3 A plausible language might contain the following primitives:

Temporal Predicates:

$Poss(a, s)$ – Predicate: Action a is possible in situation s. As in section 10.3
$Result(a, s)$ – Function from action a and situation s to situation. As in section 10.3.

Arithmetic: $x < y$, $x \leq y$, $x + y$, 0.

Window State:

$Minimized(w, s)$, $Displayed(w, s)$, $Nonexistent(w, s)$, $Active(w, s)$ – Predicates. In all these w is a window and s is a situation. ("$Displayed(w, s)$" means existent and non-minimized; it includes the case where all of w is actually occluded by other windows.)

Window Position:

$RightEdge(w, s)$, $LeftEdge(w, s)$, $TopEdge(w, s)$, $BottomEdge(w, s)$: Functions from a window w and situation s to a coordinate.

$ScreenWidth, ScreenHeight$: Constants.

Window Order:

$InFront(w1, w2, s)$: Predicate. Window $w1$ is in front of window $w2$ in situation s.

Actions:

$Minimize(w)$, $MakeVisible(w)$, $Destroy(w)$, $BringToFront(w)$ – Functions from a window w to an action.

$Move(w, dx, dy)$ – Move window w by dx to the left and dy upward. (Quantities dx and dy may be negative.)

$Resize(w, dxl, dxr, dyb, dyt)$ – Resize window w by dxl on the left, dxr on the right, dyb on bottom, and dyt on top.

12.4

a $LeftEdge(W1, S0) < LeftEdge(W2, S0) \wedge RightEdge(W2, S0) < RightEdge(W1, S0) \wedge TopEdge(W1, S0) \leq TopEdge(W2, S0) \wedge BottomEdge(W2, S0) \leq BottomEdge(W1, S0) \wedge InFront(W2, W1, S0)$.

b $\forall w, s \;\; Displayed(w, s) \Rightarrow BottomEdge(w, s) < TopEdge(w, s)$.

c $\forall w, s \;\; Poss(Create(w), s) \Rightarrow Displayed(w, Result(Create(w), s))$.

d $Displayed(w, s) \Rightarrow Poss(Minimize(w), s)$

12.5 This is the most involved representation problem. It is suitable for a group project of 2 or 3 students over the course of at least 2 weeks. Solutions should include a taxonomy, a choice of situation calculus, fluent calculus, or event calculus for handling time and change, and enough background knowledge. If a logic programming system or theorem prover is not used, students might want to write out the proofs for at least some of the answers.

12.6 Normally one would assign the preceding exercise in one assignment, and then when it is done, add this exercise (posibly varying the questions). That way, the students see whether they have made sufficient generalizations in their initial answer, and get experience with debugging and modifying a knowledge base.

12.7 Remember that we defined substances so that $Water$ is a category whose elements are all those things of which one might say "it's water." One tricky part is that the English language is ambiguous. One sense of the word "water" includes ice ("that's frozen water"), while another sense excludes it: ("that's not water—it's ice"). The sentences here seem to

use the first sense, so we will stick with that. It is the sense that is roughly synonymous with H_2O.

The other tricky part is that we are dealing with objects that change (freeze and melt) over time. Thus, it won't do to say $w \in Liquid$, because w (a mass of water) might be a liquid at one time and a solid at another. For simplicity, we will use a situation calculus representation, with sentences such as $T(w \in Liquid, s)$. There are many possible correct answers to each of these. The key thing is to be *consistent* in the way that information is represented. For example, do not use $Liquid$ as a predicate on objects if $Water$ is used as a substance category.

a. "Water is a liquid between 0 and 100 degrees." We will translate this as "For any water and any situation, the water is liquid iff and only if the water's temperature in the situation is between 0 and 100 centigrade."

$$\forall w, s \;\; w \in Water \Rightarrow (Centigrade(0) < Temperature(w, s) < Centigrade(100)) \Leftrightarrow T(w \in Liquid, s)$$

b. "Water boils at 100 degrees." It is a good idea here to do some tool-building. On page 243 we used $MeltingPoint$ as a predicate applying to individual instances of a substance. Here, we will define $SBoilingPoint$ to denote the boiling point of all instances of a substance. The basic meaning of boiling is that instances of the substance becomes gaseous above the boiling point:

$$SBoilingPoint(c, bp) \Leftrightarrow \forall x, s \;\; x \in c \Rightarrow (\forall t \;\; T(Temperature(x, t), s) \wedge t > bp \Rightarrow T(x \in Gas, s))$$

Then we need only say $SBoilingPoint(Water, Centigrade(100))$.

c. "The water in John's water bottle is frozen."

We will use the constant Now to represent the situation in which this sentence holds. Note that it is easy to make mistakes in which one asserts that only some of the water in the bottle is frozen.

$$\exists b \;\; \forall w \;\; w \in Water \wedge b \in WaterBottles \wedge Has(John, b, Now) \wedge Inside(w, b, Now) \Rightarrow (w \in Solid, Now)$$

d. "Perrier is a kind of water."

$$Perrier \subset Water$$

e. "John has Perrier in his water bottle."

$$\exists b \;\; \forall w \;\; w \in Water \wedge b \in WaterBottles \wedge Has(John, b, Now) \wedge Inside(w, b, Now) \Rightarrow w \in Perrier$$

f. "All liquids have a freezing point."

Presumably what this means is that all substances that are liquid at room temperature have a freezing point. If we use $RTLiquidSubstance$ to denote this class of substances, then we have

$$\forall c \;\; RTLiquidSubstance(c) \Rightarrow \exists t \;\; SFreezingPoint(c, t)$$

where $SFreezingPoint$ is defined similarly to $SBoilingPoint$. Note that this statement is false in the real world: we can invent categories such as "blue liquid" which do not have a unique freezing point. An interesting exercise would be to define a "pure" substance as one all of whose instances have the same chemical composition.

g. "A liter of water weighs more than a liter of alcohol."

$$\forall w, a \;\; w \in Water \wedge a \in Alcohol \wedge Volume(w) = Liters(1) \\ \wedge Volume(a) = Liters(1) \;\Rightarrow\; Mass(w) > Mass(a)$$

12.8 This is a fairly straightforward exercise that can be done in direct analogy to the corresponding definitions for sets.

a. $ExhaustivePartDecomposition$ holds between a set of parts and a whole, saying that anything that is a part of the whole must be a part of one of the set of parts.

$$\forall s, w \;\; ExhaustivePartDecomposition(s, w) \;\Leftrightarrow \\ (\forall p \;\; PartOf(p, w) \;\Rightarrow\; \exists p_2 \;\; p_2 \in s \wedge PartOf(p, p_2))$$

b. $PartPartition$ holds between a set of parts and a whole when the set is disjoint and is an exhaustive decomposition.

$$\forall s, w \;\; PartPartition(s, w) \;\Leftrightarrow \\ PartwiseDisjoint(s) \wedge ExhaustivePartDecomposition(s, w)$$

c. A set of parts is $PartwiseDisjoint$ if when you take any two parts from the set, there is nothing that is a part of both parts.

$$\forall s \;\; PartwiseDisjoint(s) \;\Leftrightarrow \\ \forall p_1, p_2 \;\; p_1 \in s \wedge p_2 \in s \wedge p_1 \neq p_2 \;\Rightarrow\; \neg\exists p_3 \; PartOf(p_3, p_1) \wedge PartOf(p_3, p_2)$$

It is *not* the case that $PartPartition(s, BunchOf(s))$ for any s. A set s may consist of physically overlapping objects, such as a hand and the fingers of the hand. In that case, $BunchOf(s)$ is equal to the hand, but s is not a partition of it. We need to ensure that the elements of s are partwise disjoint:

$$\forall s \;\; PartwiseDisjoint(s) \;\Rightarrow\; PartPartition(s, BunchOf(s)) \; .$$

12.9 In the scheme in the chapter, a conversion axiom looks like this:

$$\forall x \;\; Centimeters(2.54 \times x) = Inches(x) \; .$$

"50 dollars" is just $\$(50)$, the name of an abstract monetary quantity. For any measure function such as \$, we can extend the use of $>$ as follows:

$$\forall x, y \;\; x > y \;\Rightarrow\; \$(x) > \$(y) \; .$$

Since the conversion axiom for dollars and cents has

$$\forall x \;\; Cents(100 \times x) = \$(x)$$

it follows immediately that $\$(50) > Cents(50)$.

In the new scheme, we must introduce objects whose lengths are converted:

$$\forall x \;\; Centimeters(Length(x)) = 2.54 \times Inches(Length(x)) \; .$$

There is no obvious way to refer directly to "50 dollars" or its relation to "50 cents". Again, we must introduce objects whose monetary value is 50 dollars or 50 cents:

$$\forall x, y \;\; \$(Value(x)) = 50 \wedge Cents(Value(y)) = 50 \;\Rightarrow\; \$(Value(x)) > \$(Value(y))$$

12.10 Plurals can be handled by a *Plural* relation between strings, e.g.,

$$Plural(\text{“}computer\text{”}, \text{“}computers\text{”})$$

plus an assertion that the plural (or singular) of a name is also a name for the same category:

$$\forall c, s_1, s_2 \;\; Name(s_1, c) \wedge (Plural(s_1, s_2) \vee Plural(s_2, s_1)) \;\Rightarrow\; Name(s_2, c)$$

Conjunctions can be handled by saying that any conjunction string is a name for a category if one of the conjuncts is a name for the category:

$$\forall c, s, s_2 \;\; Conjunct(s_2, s) \wedge Name(s_2, c) \;\Rightarrow\; Name(s, c)$$

where $Conjunct$ is defined appropriately in terms of concatenation. Probably it would be better to redefine $RelevantCategoryName$ instead.

12.11 Section 12.3 includes a couple of axioms for the wumpus world:

$$Initiates(e, HaveArrow(a), t) \;\Leftrightarrow\; e = Start$$
$$Terminates(e, HaveArrow(a), t) \;\Leftrightarrow\; e \in Shootings(a)$$

Here is an axiom for turning; the others are similar albeit more complex. Let the term $TurnRight(a)$ denote the event category of the agent turning right. We want to say about it that if (say) the agent is facing south up to the beginning of the action, then it is facing west after the action ends, and so on.

$$\begin{aligned} T(TurnRight(a), i) \quad \Leftrightarrow \quad & \\ & [\exists h \;\; Meets(h, i) \wedge T(FacingSouth(a), h) \;\Rightarrow \\ & \quad Clipped(FacingSouth(a), i) \wedge Restored(FacingWest(a), i)] \\ \vee \quad & \ldots \end{aligned}$$

12.12 $Starts(IK, LK)$.
$Finishes(PK, LK)$.
$During(LK, LJ)$.
$Meets(LK, PJ)$.
$Overlap(LK, LC)$.
$Before(IK, PK)$.
$During(IK, LJ)$.
$Before(IK, PJ)$.
$Before(IK, LC)$.
$During(PK, LJ)$.
$Meets(PK, PJ)$.
$During(PK, LC)$.
$During(PJ, LJ)$.

$Overlap(LJ, LC)$.
$During(PJ, LC)$.

12.13 The main difficulty with simultaneous (also called concurrent) events and actions is how to account correctly for possible interference. A good starting point is the expository paper by Shanahan (1999). Section 5 of that paper shows how to manage concurrent actions by the introduction of additional generic predicates $Cancels$ and $Canceled$, describing circumstances in which actions may interfere with each other. We avoid lots of "non-cancellation" assertions using the same predicate-completion trick as in successor-state axioms, and the meaning of cancellation is defined once and for all through its connection to clipping, restoring, etc.

12.14 For quantities such as length and time, the conversion axioms such as

$$Centimeters(2.54 \times d) = Inches(d)$$

are absolutes that hold (with a few exceptions) for all time. The same is true for conversion axioms within a given currency; for example, $US\$(1) = US\text{¢}(100)$. When it comes to conversion *between* currencies, we make the simplifying assumption that at any given time t there is a prevailing exchange rate:

$$T(ExchangeRate(UK\pounds(1), US\$(1)) = 1.55, t)$$

and the rate is reciprocal:

$$ExchangeRate(UK\pounds(1), US\$(1)) = 1/ExchangeRate(US\$(1), UK\pounds(1)) \ .$$

What we cannot do, however, is write

$$T(UK\pounds(1) = US\$(1.55), t)$$

thereby *equating* abstract amounts of money in different currencies. At any given moment, prevailing exchange rates across the world's currencies need not be consistent, and using equality across currencies would therefore introduce a *logical* inconsistency. Instead, exchange rates should be interpreted as indicating a willingness to exchange, perhaps with some commission; and exchange rate inconsistency is an opportunity for arbitrage. A more sophisticated model would include the entity offering the rate, limits on amounts and forms of payment, etc.

12.15 Any object x is an event, and $Location(x)$ is the event that for every subinterval of time, refers to the place where x is. For example, $Location(Peter)$ is the complex event consisting of his home from midnight to about 9:00 today, then various parts of the road, then his office from 10:00 to 1:30, and so on. To say that an event is fixed is to say that any two moments of the event have the same spatial extent:

$$\begin{aligned}&\forall e \ \ Fixed(e) \ \Leftrightarrow \\ &\quad (\forall a, b \ \ a \in Moments \wedge b \in Moments \wedge Subevent(a, e) \wedge Subevent(b, e) \\ &\qquad \Rightarrow \ SpatialExtent(a) = SpatialExtent(b))\end{aligned}$$

12.16 Let $Trade(b, x, a, y)$ denote the class of events where person b trades object y to person a for object x:

$$T(Trade(b, x, a, y), i) \Leftrightarrow$$
$$T(Owns(b, y), Start(i)) \wedge T(Owns(a, x), Start(i)) \wedge$$
$$T(Owns(b, x), End(i)) \wedge T(Owns(a, y), End(i))$$

Now the only tricky part about defining buying in terms of trading is in distinguishing a price (a measurement) from an actual collection of money.

$$T(Buy(b, x, a, p), i) \Leftrightarrow \exists m \;\; Money(m) \wedge Trade(b, x, a, m) \wedge Value(m) = p$$

12.17 There are many possible approaches to this exercise. The idea is for the students to think about doing knowledge representation for real; to consider a host of complications and find some way to represent the facts about them. Some of the key points are:

- Ownership occurs over time, so we need either a situation-calculus or interval-calculus approach.
- There can be joint ownership and corporate ownership. This suggests the owner is a group of some kind, which in the simple case is a group of one person.
- Ownership provides certain rights: to use, to resell, to give away, etc. Much of this is outside the definition of ownership *per se*, but a good answer would at least consider how much of this to represent.
- Own can own abstract obligations as well as concrete objects. This is the idea behind the futures market, and also behind banks: when you deposit a dollar in a bank, you are giving up ownership of that particular dollar in exchange for ownership of the right to withdraw another dollar later. (Or it could coincidentally turn out to be the exact same dollar.) Leases and the like work this way as well. This is tricky in terms of representation, because it means we have to reify transactions of this kind. That is, $Withdraw(person, money, bank, time)$ must be an object, not a predicate.

12.18 We refer the reader to Fagin *et al.* (1995) for several examples of the type of reasoning needed. Just to get you started: In Game 1, Alice says "I don't know." If Carlos had K-K, and given that Alice can see Bob's K-K, then she would know that Bob and Carlos had all four kings between them and she would announce A-A. Therefore, Carlos does not have K-K. Then Bob says "I don't know." If Carlos had A-A, and given that Bob can see Alice's A-A, then he would know that Alice and Carlos had all four aces between them and he would announce A-A. Therefore, Carlos does not have A-A. Therefore Carlos should announce A-K.

12.19

A. The logical omniscience assumption is a reasonable idealization. The limiting factor here is generally the information available to the players, not the difficulty of making inferences.

B. This kind of reasoning cannot be accommodated in a theory with logical omniscience. If logical omniscience were true, then every player could always figure out the optimal move instantaneously.

C. Logical omniscience is a reasonable idealization. The costs of getting the information are almost always much greater than the costs of reasoning with it.

D. It depends on the kind of reasoning you want to do. If you want to reason about the relation of cryptography to particular computational problems, then logical omniscience cannot be assumed, because the assumption entails that any computational problem can be solved instantly. On the other hand, if you are willing to idealize the encryption/decryption as a magical process with no computational basis, then it may be reasonable to apply a theory with logical omniscience to other aspects of the theory.

12.20 This corresponds to the following open formula:

$$\begin{aligned}
&Man(x) \wedge \exists\, s_1, s_2, s_3 \;\; Son(s_1, x) \wedge Son(s_2, x) \wedge Son(s_3, x) \\
&\qquad \wedge s_1 \neq s_2 \wedge s_1 \neq s_3 \wedge s_2 \neq s_3 \\
&\wedge \neg\exists\, d_1, d_2, d_3 \;\; Daughter(d_1, x) \wedge Daughter(d_2, x) \wedge Daughter(d_3, x) \\
&\qquad \wedge d_1 \neq d_2 \wedge d_1 \neq d_3 \wedge d_2 \neq d_3 \\
&\wedge \forall\, s \;\; Son(s, x) \;\Rightarrow\; Unemployed(s) \wedge Married(s) \wedge Doctor(Spouse(s)) \\
&\wedge \forall\, d \;\; Daughter(d, x) \;\Rightarrow\; Professor(d) \wedge \\
&\qquad (Department(d) = Physics \vee Department(d) = Math)\ .
\end{aligned}$$

12.21 In many AI and Prolog textbooks, you will find it stated plainly that implications suffice for the implementation of inheritance. This is true in the logical but not the practical sense.

a. Here are three rules, written in Prolog. We actually would need many more clauses on the right hand side to distinguish between different models, different options, etc.

```
worth(X,575) :- year(X,1973), make(X,dodge), style(X,van).
worth(X,27000) :- year(X,1994), make(X,lexus), style(X,sedan).
worth(X,5000) :- year(X,1987), make(X,toyota), style(X,sedan).
```

To find the value of JB, given a data base with `year(jb,1973)`, `make(jb,dodge)` and `style(jb,van)` we would call the backward chainer with the goal `worth(jb,D)`, and read the value for `D`.

b. The time efficiency of this query is $O(n)$, where n in this case is the 11,000 entries in the Blue Book. A semantic network with inheritance would allow us to follow a link from `JB` to `1973-dodge-van`, and from there to follow the `worth` slot to find the dollar value in $O(1)$ time.

c. With forward chaining, as soon as we are told the three facts about JB, we add the new fact `worth(jb,575)`. Then when we get the query `worth(jb,D)`, it is $O(1)$ to find the answer, assuming indexing on the predicate and first argument. This makes logical inference seem just like semantic networks except for two things: the logical inference does a hash table lookup instead of pointer following, and logical inference explicitly stores `worth` statements for each individual car, thus wasting space if there are a lot of individual cars. (For this kind of application, however, we will probably want to consider only a few individual cars, as opposed to the 11,000 different models.)

d. If each category has many properties—for example, the specifications of all the replacement parts for the vehicle—then forward-chaining on the implications will also be an impractical way to figure out the price of a vehicle.

e. If we have a rule of the following kind:

```
worth(X,D) :- year-make-style(X,Yr,Mk,St),
              year-make-style(Y,Yr,Mk,St), worth(Y,D).
```

together with facts in the database about some other specific vehicle of the same type as JB, then the query `worth(jb,D)` will be solved in $O(1)$ time with appropriate indexing, regardless of how many other facts are known about that type of vehicle and regardless of the number of types of vehicle.

12.22 When categories are reified, they can have properties as individual objects (such as $Cardinality$ and $Supersets$) that do not apply to their elements. Without the distinction between boxed and unboxed links, the sentence $Cardinality(SingletonSets, 1)$ might mean that every singleton set has one element, or that there si only one singleton set.

12.23 Here is an initial sketch of one approach. (Others are possible.) A given object to be purchased may *require* some additional parts (e.g., batteries) to be functional, and there may also be *optional* extras. We can represent requirements as a relation between an individual object and a class of objects, qualified by the number of objects required:

$$\forall x \;\; x \in \mathit{Coolpix995DigitalCamera} \;\Rightarrow\; \mathit{Requires}(x, AABattery, 4) \; .$$

We also need to know that a particular object is compatible, i.e., fills a given role appropriately. For example,

$$\forall x, y \;\; x \in \mathit{Coolpix995DigitalCamera} \wedge y \in \mathit{DuracellAABattery} \\ \Rightarrow \; Compatible(y, x, AABattery)$$

Then it is relatively easy to test whether the set of ordered objects contains compatible required objects for each object.

12.24 Chapter 23 explains how to use logic to parse text strings and extract semantic information. The outcome of this process is a definition of what objects are acceptable to the user for a specific shopping request; this allows the agent to go out and find offers matching the user's requirements. We omit the full definition of the agent, although a skeleton may appear on the AIMA project web pages.

12.25 Here is a simple version of the answer; it can be elaborated *ad infinitum*. Let the term $Buy(b, x, s, p)$ denote the event category of buyer b buying object x from seller s for price p. We want to say about it that b transfers the money to s, and s transfers ownership of x to b.

$$T(Buy(b, x, s, p), i) \;\Leftrightarrow \\ T(Owns(s, x), Start(i)) \wedge \\ \exists m \;\; Money(m) \wedge p = Value(m) \wedge T(Owns(b, m), Start(i)) \wedge \\ T(Owns(b, x), End(i)) \wedge T(Owns(s, m), End(i))$$

Solutions for Chapter 13
Quantifying Uncertainty

13.1 The "first principles" needed here are the definition of conditional probability, $P(X|Y) = P(X \wedge Y)/P(Y)$, and the definitions of the logical connectives. It is not enough to say that if $B \wedge A$ is "given" then A must be true! From the definition of conditional probability, and the fact that $A \wedge A \Leftrightarrow A$ and that conjunction is commutative and associative, we have

$$P(A|B \wedge A) = \frac{P(A \wedge (B \wedge A))}{P(B \wedge A)} = \frac{P(B \wedge A)}{P(B \wedge A)} = 1$$

13.2 The main axiom is axiom 3: $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$. For the discrete random variable X, let a be the event that $X = x_1$, and b be the event that X has any other value. Then we have

$$P(X = x_1 \vee X = other) = P(X = x_1) + P(X = other) + 0$$

where we know that $P(X = x_1 \wedge X = other)$ is 0 because a variable cannot take on two distinct values. If we now break down the case of $X = others$, we eventually get

$$P(X = x_1 \vee \cdots \vee X = x_n) = P(X = x_1) + \cdots + P(X = x_n) \ .$$

But the left-hand side is equivalent to $P(true)$, which is 1 by axiom 2, so the sum of the right-hand side must also be 1.

13.3

a. True. By the product rule we know $P(b, c)P(a|b, c) = P(a, c)P(b|a, c)$, which by assumption reduces to $P(b, c) = P(a, c)$. Dividing through by $P(c)$ gives the result.

b. False. The statement $P(a|b, c) = P(a)$ merely states that a is independent of b and c, it makes no claim regarding the dependence of b and c. A counter-example: a and b record the results of two independent coin flips, and $c = b$.

c. False. While the statement $P(a|b) = P(a)$ implies that a is independent of b, it does not imply that a is conditionally independent of b given c. A counter-example: a and b record the results of two independent coin flips, and c equals the xor of a and b.

13.4 Probably the easiest way to keep track of what's going on is to look at the probabilities of the atomic events. A probability assignment to a set of propositions is consistent with the axioms of probability if the probabilities are consistent with an assignment to the atomic events that sums to 1 and has all probabilities between 0 and 1 inclusive. We call the probabilities of the atomic events a, b, c, and d, as follows:

	B	$\neg B$
A	a	b
$\neg A$	c	d

We then have the following equations:

$$P(A) = a + b = 0.4$$
$$P(B) = a + c = 0.3$$
$$P(A \vee B) = a + b + c = 0.5$$
$$P(True) = a + b + c + d = 1$$

From these, it is straightforward to infer that $a = 0.2$, $b = 0.2$, $c = 0.1$, and $d = 0.5$. Therefore, $P(A \wedge B) = a = 0.2$. Thus the probabilities given are consistent with a rational assignment, and the probability $P(A \wedge B)$ is exactly determined. (This latter fact can be seen also from axiom 3 on page 422.)

If $P(A \vee B) = 0.7$, then $P(A \wedge B) = a = 0$. Thus, even though the bet outlined in Figure 13.3 loses if A and B are both true, the agent believes this to be impossible so the bet is still rational.

13.5

a. Each atomic event is a conjunction of n literals, one per variable, with each literal either positive or negative. For the events to be distinct, at least one pair of corresponding literals must be nonidentical; hence, the conjunction of the two events contains the literals X_i and $\neg X_i$ for some i, so the conjunction reduces to $False$.

b. Proof by induction on n. For $n = 0$, the only event is the empty conjunction $True$, and the disjunction containing only this event is also $True$. Inductive step: assume the claim holds for n variables. The disjunction for $n + 1$ variables consists of pairs of disjuncts of the form $(T_n \wedge X_{n+1}) \vee (T_n \wedge \neg X_{n+1})$ for all possible atomic event conjunctions T_n. Each pair logically reduces to T_n, so the entire disjunction reduces to the disjunction for n variables, which by hypothesis is equivalent to $True$.

c. Let α be the sentence in question and $\mu_1, \ldots, \mu_k$ be the atomic event sentences that entail its truth. Let M_i be the model corresponding to μ_i (its *only* model). To prove that $\mu_1 \vee \ldots \vee \mu_k \equiv \alpha$, simply observe the following:

- Because $\mu_i \models \alpha$, α is true in all the models of μ_i, so α is true in M_i.
- The models of $\mu_1 \vee \ldots \vee \mu_k$ are exactly $M_1, \ldots, M_k$ because any two atomic events are mutually exclusive, so any given model can satisfy at most one disjunct, and a model that satisfies a disjunct must be the model corresponding to that atomic event.
- If any model M satisfies α, then the corresponding atomic-event sentence μ entails α, so the models of α are exactly $M_1, \ldots, M_k$.

Hence, α and $\mu_1 \vee \ldots \vee \mu_k$ have the same models, so are logically equivalent.

13.6 Equation (13.4) states that $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$. This can be proved directly from Equation (13.2), using obvious abbreviations for the possible-world probabili-

ties:

$$\begin{aligned} P(a \vee b) &= p_{a,b} + p_{a,\neg b} + p_{\neg a,b} \\ P(a) &= p_{a,b} + p_{a,\neg b} \\ P(b) &= p_{a,b} + p_{\neg a,b} \\ P(a \wedge b) &= p_{a,b} \, . \end{aligned}$$

13.7 This is a classic combinatorics question that could appear in a basic text on discrete mathematics. The point here is to refer to the relevant axioms of probability: principally, axiom 3 on page 422. The question also helps students to grasp the concept of the joint probability distribution as the distribution over all possible states of the world.

a. There are $\binom{52}{5}$ = $(52 \times 51 \times 50 \times 49 \times 48)/(1 \times 2 \times 3 \times 4 \times 5)$ = 2,598,960 possible five-card hands.

b. By the fair-dealing assumption, each of these is equally likely. By axioms 2 and 3, each hand therefore occurs with probability 1/2,598,960.

c. There are four hands that are royal straight flushes (one in each suit). By axiom 3, since the events are mutually exclusive, the probability of a royal straight flush is just the sum of the probabilities of the atomic events, i.e., 4/2,598,960 = 1/649,740. For "four of a kind" events, There are 13 possible "kinds" and for each, the fifth card can be one of 48 possible other cards. The total probability is therefore $(13 \times 48)/2,598,960 = 1/4,165$.

These questions can easily be augmented by more complicated ones, e.g., what is the probability of getting a full house given that you already have two pairs? What is the probability of getting a flush given that you have three cards of the same suit? Or you could assign a project of producing a poker-playing agent, and have a tournament among them.

13.8 The main point of this exercise is to understand the various notations of bold versus non-bold P, and uppercase versus lowercase variable names. The rest is easy, involving a small matter of addition.

a. This asks for the probability that *Toothache* is true.

$$P(toothache) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2$$

b. This asks for the vector of probability values for the random variable *Cavity*. It has two values, which we list in the order $\langle true, false \rangle$. First add up $0.108 + 0.012 + 0.072 + 0.008 = 0.2$. Then we have

$$\mathbf{P}(Cavity) = \langle 0.2, 0.8 \rangle \, .$$

c. This asks for the vector of probability values for *Toothache*, given that *Cavity* is true.

$$\mathbf{P}(Toothache|cavity) = \langle (.108 + .012)/0.2, (0.072 + 0.008)/0.2 \rangle = \langle 0.6, 0.4 \rangle$$

d. This asks for the vector of probability values for *Cavity*, given that either *Toothache* or *Catch* is true. First compute $P(toothache \vee catch) = 0.108 + 0.012 + 0.016 + 0.064 + 0.072 + 0.144 = 0.416$. Then

$$\begin{aligned}&\mathbf{P}(Cavity|toothache \vee catch) = \\ &\quad \langle (0.108 + 0.012 + 0.072)/0.416, (0.016 + 0.064 + 0.144)/0.416 \rangle = \\ &\quad \langle 0.4615, 0.5384 \rangle\end{aligned}$$

13.9 Let e and o be the initial scores, m be the score required to win, and p be the probability that E wins each round. One can easily write down a recursive formula for the probability that E wins from the given initial state:

$$w_E(p,e,o,m) = \begin{cases} 1 & \text{if } e = m \\ 0 & \text{if } o = m \\ p \cdot w_E(p,e+1,o,m) + (1-p) \cdot w_E(p,e,o+1,m) & \text{otherwise} \end{cases}$$

This translates directly into code that can be used to compute the answer,

$$w_E(0.5,4,2,7) = 0.7734375 \ .$$

With a bit more work, we can derive a nonrecursive formula:

$$w_E(p,e,o,m) = p^{m-e} \sum_{i=0}^{m-o-1} \binom{i+m-e-1}{i} (1-p)^i \ .$$

Each term in the sum corresponds to the probability of winning by exactly a particular score; e.g., starting from 4–2, one can win by 7–2, 7–3, 7–4, 7–5, or 7–6. Each final score requires E to win exactly $m-e$ rounds while the opponent wins exactly i rounds, where $i = 0, 1, \ldots, m-o-1$; and the combinatorial term counts the number of ways this can happen without E winning first by a larger margin. One can check the nonrecursive formula by showing that it satisfies the recursive formula. (It may be helpful to suggest to students that they start by building the lattice of states implied by the above recursive formula and calculating (bottom-up) the symbolic win probabilities in terms of p rather than 0.5, so that they can see the general shape emerging.)

13.10

a. To compute the expected payback for the machine, we determine the probability for each winning outcome, multiply it by the amount that would be won in that instance, and sum all possible winning combinations. Since each symbol is equally likely, the first four cases have probability $(1/4)^3 = 1/64$.

However, in the case of computing winning probabilities for cherries, we must only consider the highest paying case, so we must subtract the probability for dominating winning cases from each subsequent case (e.g., in the case of two cherries, we subtract off the probability of getting three cherries):

CHERRY/CHERRY/? $3/64 = (1/4)^2 - 1/64$
CHERRY/?/? $12/64 = (1/4)^1 - 3/64 - 1/64$

The expectation is therefore

$$20 \cdot 1/64 + 15 \cdot 1/64 + 5 \cdot 1/64 + 3 \cdot 1/64 + 2 \cdot 3/64 + 1 \cdot 12/64 = 61/64.$$

Thus, the expected payback percentage is 61/64 (which is less than 1 as we would expect of a slot machine that was actually generating revenue for its owner).

b. We can tally up the probabilities we computed in the previous section, to get

$$1/64 + 1/64 + 1/64 + 1/64 + 3/64 + 12/64 = 19/64.$$

Alternatively, we can observe that we win if either all symbols are the same (denote this event S), or if the first symbol is cherry (denote this event C). Then applying the inclusion-exclusion identity for disjunction:

$$P(S \vee C) = P(S) + P(C) - P(S \wedge C) = (1/4)^2 + 1/4 - 1/64 = 19/64.$$

c. Using a simple Python simulation, we find a mean of about 210, and a median of 21. This shows the distribution of number of plays is heavy tailed: most of the time you run out of money relatively quickly, but occasionally you last for thousands of plays.

```
import random

def trial():
        funds = 10
        plays = 0
        while funds >= 1:
                funds -=1
                plays += 1
                slots = [random.choice(
                          ["bar", "bell", "lemon", "cherry"])
                         for i in range(3)]
                if slots[0] == slots[1]:
                        if slots[1] == slots[2]:
                                num_equal = 3
                        else:
                                num_equal = 2
                else:
                        num_equal = 1
                if slots[0] == "cherry":
                        funds += num_equal
                elif num_equal == 3:
                        if slots[0] == "bar":
                                funds += 20
                        elif slots[0] == "bell":
                                funds += 15
                        else:
                                funds += 5
        return plays

def test(trials):
        results = [trial() for i in xrange(trials)]
        mean = sum(results) / float(trials)
        median = sorted(results)[trials/2]
        print "%s trials: mean=%s, median=%s" % (trials, mean, median)
```

```
test(10000)
```

13.11 The correct message is received if either zero or one of the $n+1$ bits are corrupted. Since corruption occurs independently with probability ϵ, the probability that zero bits are corrupted is $(1-\epsilon)^{n+1}$. There are $n+1$ mutually exclusive ways that exactly one bit can be corrupted, one for each bit in the message. Each has probability $\epsilon(1-\epsilon)^n$, so the overall probability that exactly one bit is corrupted is $n\epsilon(1-\epsilon)^n$. Thus, the probability that the correct message is received is $(1-\epsilon)^{n+1} + n\epsilon(1-\epsilon)^n$.

The maximum feasible value of n, therefore, is the largest n satisfying the inequality

$$(1-\epsilon)^{n+1} + n\epsilon(1-\epsilon)^n \geq 1-\delta.$$

Numerically solving this for $\epsilon = 0.001$, $\delta = 0.01$, we find $n = 147$.

13.12 Independence is symmetric (that is, a and b are independent iff b and a are independent) so $P(a|b) = P(a)$ is the same as $P(b|a) = P(b)$. So we need only prove that $P(a|b) = P(a)$ is equivalent to $P(a \wedge b) = P(a)P(b)$. The product rule, $P(a \wedge b) = P(a|b)P(b)$, can be used to rewrite $P(a \wedge b) = P(a)P(b)$ as $P(a|b)P(b) = P(a)P(b)$, which simplifies to $P(a|b) = P(a)$.

13.13

Let V be the statement that the patient has the virus, and A and B the statements that the medical tests A and B returned positive, respectively. The problem statement gives:

$$\begin{aligned} P(V) &= 0.01 \\ P(A|V) &= 0.95 \\ P(A|\neg V) &= 0.10 \\ P(B|V) &= 0.90 \\ P(B|\neg V) &= 0.05 \end{aligned}$$

The test whose positive result is more indicative of the virus being present is the one whose posterior probability, $P(V|A)$ or $P(V|B)$ is largest. One can compute these probabilities directly from the information given, finding that $P(V|A) = 0.0876$ and $P(V|B) = 0.1538$, so B is more indicative.

Equivalently, the questions is asking which test has the highest posterior odds ratio $P(V|A)/P(\neg V|A)$. From the odd form of Bayes theorem:

$$\frac{P(V|A)}{P(\neg V|A)} = \frac{P(A|V)}{P(A|\neg V)}\frac{P(V)}{P(\neg V)}$$

we see that the ordering is independent of the probability of V, and that we just need to compare the likelihood ratios $P(A|V)/P(A|\neg V) = 9.5$ and $P(B|V)/P(V|\neg V) = 18$ to find the answer.

13.14

If the probability x is known, then successive flips of the coin are independent of each other, since we know that each flip of the coin will land $heads$ with probability x. Formally, if $F1$ and $F2$ represent the results of two successive flips, we have

$$P(F1 = heads, F2 = heads|x) = x * x = P(F1 = heads|x)P(F2 = heads|x)$$

Thus, the events $F1 = heads$ and $F2 = heads$ are independent.

If we do not know the value of x, however, the probability of each successive flip is dependent on the result of all previous flips. The reason for this is that each successive flip gives us information to better estimate the probability x (i.e., determining the posterior estimate for x given our prior probability and the evidence we see in the most recent coin flip). This new estimate of x would then be used as our "best guess" of the probability of the coin coming up $heads$ on the next flip. Since this estimate for x is based on all the previous flips we have seen, the probability of the next flip coming up $heads$ depends on how many $heads$ we saw in all previous flips, making them dependent.

For example, if we had a uniform prior over the probability x, then one can show that after n flips if m of them come up heads then the probability that the next one comes up heads is $(m + 1)/(n + 2)$, showing dependence on previous flips.

13.15 We are given the following information:

$$\begin{aligned} P(test|disease) &= 0.99 \\ P(\neg test|\neg disease) &= 0.99 \\ P(disease) &= 0.0001 \end{aligned}$$

and the observation $test$. What the patient is concerned about is $P(disease|test)$. Roughly speaking, the reason it is a good thing that the disease is rare is that $P(disease|test)$ is proportional to $P(disease)$, so a lower prior for $disease$ will mean a lower value for $P(disease|test)$. Roughly speaking, if 10,000 people take the test, we expect 1 to actually have the disease, and most likely test positive, while the rest do not have the disease, but 1% of them (about 100 people) will test positive anyway, so $P(disease|test)$ will be about 1 in 100. More precisely, using the normalization equation from page 480:

$$\begin{aligned} &P(disease|test) \\ &= \frac{P(test|disease)P(disease)}{P(test|disease)P(disease)+P(test|\neg disease)P(\neg disease)} \\ &= \frac{0.99\times 0.0001}{0.99\times 0.0001+0.01\times 0.9999} \\ &= .009804 \end{aligned}$$

The moral is that when the disease is much rarer than the test accuracy, a positive test result does not mean the disease is likely. A false positive reading remains much more likely.

Here is an alternative exercise along the same lines: A doctor says that an infant who predominantly turns the head to the right while lying on the back will be right-handed, and one who turns to the left will be left-handed. Isabella predominantly turned her head to the left. Given that 90% of the population is right-handed, what is Isabella's probability of being right-handed if the test is 90% accurate? If it is 80% accurate?

The reasoning is the same, and the answer is 50% right-handed if the test is 90% accurate, 69% right-handed if the test is 80% accurate.

13.16 The basic axiom to use here is the definition of conditional probability:

a. We have

$$\mathbf{P}(A, B|E) = \frac{\mathbf{P}(A, B, E)}{\mathbf{P}(E)}$$

and

$$\mathbf{P}(A|B, E)\mathbf{P}(B|E) = \frac{\mathbf{P}(A, B, E)}{\mathbf{P}(B, E)}\frac{\mathbf{P}(B, E)}{\mathbf{P}(E)} = \frac{\mathbf{P}(A, B, E)}{\mathbf{P}(E)}$$

hence

$$\mathbf{P}(A, B|E) = \mathbf{P}(A|B, E)\mathbf{P}(B|E)$$

b. The derivation here is the same as the derivation of the simple version of Bayes' Rule on page 426. First we write down the dual form of the conditionalized product rule, simply by switching A and B in the above derivation:

$$\mathbf{P}(A, B|E) = \mathbf{P}(B|A, E)\mathbf{P}(A|E)$$

Therefore the two right-hand sides are equal:

$$\mathbf{P}(B|A, E)\mathbf{P}(A|E) = \mathbf{P}(A|B, E)\mathbf{P}(B|E)$$

Dividing through by $\mathbf{P}(B|E)$ we get

$$\mathbf{P}(A|B, E) = \frac{\mathbf{P}(B|A, E)\mathbf{P}(A|E)}{\mathbf{P}(B|E)}$$

13.17 The key to this exercise is rigorous and frequent application of the definition of conditional probability, $\mathbf{P}(X|Y) = \mathbf{P}(X, Y)/\mathbf{P}(Y)$. The original statement that we are given is:

$$\mathbf{P}(A, B|C) = \mathbf{P}(A|C)\mathbf{P}(B|C)$$

We start by applying the definition of conditional probability to two of the terms in this statement:

$$\mathbf{P}(A, B|C) = \frac{\mathbf{P}(A, B, C)}{\mathbf{P}(C)} \text{ and } \mathbf{P}(B|C) = \frac{\mathbf{P}(B, C)}{\mathbf{P}(C)}$$

Now we substitute the right hand side of these definitions for the left hand sides in the original statement to get:

$$\frac{\mathbf{P}(A, B, C)}{\mathbf{P}(C)} = \mathbf{P}(A|C)\frac{\mathbf{P}(B, C)}{\mathbf{P}(C)}$$

Now we need the definition once more:

$$\mathbf{P}(A, B, C) = \mathbf{P}(A|B, C)\mathbf{P}(B, C)$$

We substitute this right hand side for $\mathbf{P}(A, B, C)$ to get:

$$\frac{\mathbf{P}(A|B, C)\mathbf{P}(B, C)}{\mathbf{P}(C)} = \mathbf{P}(A|C)\frac{\mathbf{P}(B, C)}{\mathbf{P}(C)}$$

Finally, we cancel the $\mathbf{P}(B, C)$ and $\mathbf{P}(C)$s to get:

$$\mathbf{P}(A|B, C) = \mathbf{P}(A|C)$$

The second part of the exercise follows from by a similar derivation, or by noticing that A and B are interchangeable in the original statement (because multiplication is commutative and A, B means the same as B, A).

In Chapter 14, we will see that in terms of Bayesian networks, the original statement means that C is the lone parent of A and also the lone parent of B. The conclusion is that knowing the values of B and C is the same as knowing just the value of C in terms of telling you something about the value of A.

13.18

a. A typical "counting" argument goes like this: There are n ways to pick a coin, and 2 outcomes for each flip (although with the fake coin, the results of the flip are indistinguishable), so there are $2n$ total atomic events, each equally likely. Of those, only 2 pick the fake coin, and $2 + (n-1)$ result in heads. So the probability of a fake coin given heads, $P(fake|heads)$, is $2/(2+n-1) = 2/(n+1)$.

Often such counting arguments go astray when the situation gets complex. It may be better to do it more formally:

$$\begin{aligned}\mathbf{P}(Fake|heads) &= \alpha\mathbf{P}(heads|Fake)\mathbf{P}(Fake)\\ &= \alpha\langle 1.0, 0.5\rangle\langle 1/n, (n-1)/n\rangle\\ &= \alpha\langle 1/n, (n-1)/2n\rangle\\ &= \langle 2/(n+1), (n-1)/(n+1)\rangle\end{aligned}$$

b. Now there are $2^k n$ atomic events, of which 2^k pick the fake coin, and $2^k + (n-1)$ result in heads. So the probability of a fake coin given a run of k heads, $P(fake|heads^k)$, is $2^k/(2^k + (n-1))$. Note this approaches 1 as k increases, as expected. If $k = n = 12$, for example, than $P(fake|heads^{10}) = 0.9973$.

Doing it the formal way:

$$\begin{aligned}\mathbf{P}(Fake|heads^k) &= \alpha\mathbf{P}(heads^k|Fake)\mathbf{P}(Fake)\\ &= \alpha\langle 1.0, 0.5^k\rangle\langle 1/n, (n-1)/n\rangle\\ &= \alpha\langle 1/n, (n-1)/2^k n\rangle\\ &= \langle 2^k/(2^k+n-1), (n-1)/(2^k+n-1)\rangle\end{aligned}$$

c. The procedure makes an error if and only if a fair coin is chosen and turns up heads k times in a row. The probability of this

$$P(heads^k|\neg fake)P(\neg fake) = (n-1)/2^k n\ .$$

13.19 The important point here is that although there are often many possible routes by which answers can be calculated in such problems, it is usually better to stick to systematic "standard" routes such as Bayes' Rule plus normalization. Chapter 14 describes general-purpose, systematic algorithms that make heavy use of normalization. We could guess that $P(S|\neg M) \approx 0.05$, or we could calculate it from the information already given (although the idea here is to assume that $P(S)$ is *not* known):

$$P(S|\neg M) = \frac{P(\neg M|S)P(S)}{P(\neg M)} = \frac{(1-P(M|S))P(S)}{1-P(\neg M)} = \frac{0.9998 \times 0.05}{0.99998} = 0.049991$$

Normalization proceeds as follows:

$$P(M|S) \propto P(S|M)P(M) = 0.5/50,000 = 0.00001$$
$$P(\neg M|S) \propto P(S|\neg M)P(\neg M) = 0.049991 \times 0.99998 = 0.04999$$
$$P(M|S) = \frac{0.00001}{0.00001+0.04999} = 0.0002$$
$$P(\neg M|S) = \frac{0.00001}{0.00001+0.04999} = 0.9998$$

13.20 Let the probabilities be as follows:

x	y	z	$P(x,y,z)$
F	F	F	a
F	F	T	b
F	T	F	c
F	T	T	d
T	F	F	e
T	F	T	f
T	T	F	g
T	T	T	h

Conditional independence asserts that

$$\mathbf{P}(X, Y \mid Z) = \mathbf{P}(X \mid Z)\mathbf{P}(Y \mid Z)$$

which we can rewrite in terms of the joint distribution using the definition of conditional probability and marginals:

$$\frac{\mathbf{P}(X,Y,Z)}{\mathbf{P}(Z)} = \frac{\mathbf{P}(X,Z)}{\mathbf{P}(Z)} \cdot \frac{\mathbf{P}(Y,Z)}{\mathbf{P}(Z)}$$
$$\mathbf{P}(X,Y,Z) = \frac{\mathbf{P}(X,Z)\mathbf{P}(Y,Z)}{\mathbf{P}(Z)} = \frac{\left(\sum_y \mathbf{P}(X,y,Z)\right)\left(\sum_x \mathbf{P}(x,Y,Z)\right)}{\sum_{x,y} \mathbf{P}(x,y,Z)} .$$

Now we instantiate X, Y, Z in all 8 ways to obtain the following 8 equations:

$$a = (a+c)(a+e)/(a+c+e+g) \text{ or } ag = ce$$
$$b = (b+d)(b+f)/(b+d+f+h) \text{ or } bh = df$$
$$c = (a+c)(c+g)/(a+c+e+g) \text{ or } ce = ag$$
$$d = (b+d)(d+h)/(b+d+f+h) \text{ or } df = bh$$
$$e = (e+g)(a+e)/(a+c+e+g) \text{ or } ce = ag$$
$$f = (f+h)(b+f)/(b+d+f+h) \text{ or } df = bh$$
$$g = (e+g)(c+g)/(a+c+e+g) \text{ or } ag = ce$$
$$h = (f+h)(d+h)/(b+d+f+h) \text{ or } bh = df .$$

Thus, there are only 2 nonredundant equations, $ag = ce$ and $bh = df$. This is what we would expect: the general distribution requires $8 - 1 = 7$ parameters, whereas the Bayes net with Z as root and X and Y as conditionally indepednent children requires 1 parameter for Z and 2 each for X and Y, or 5 in all. Hence the conditional independence assertion removes two degrees of freedom.

13.21 The relevant aspect of the world can be described by two random variables: B means the taxi *was* blue, and LB means the taxi *looked blue*. The information on the reliability of color identification can be written as

$$P(LB|B) = 0.75 \qquad P(\neg LB|\neg B) = 0.75$$

We need to know the probability that the taxi was blue, given that it looked blue:

$$P(B|LB) \propto P(LB|B)P(B) \propto 0.75P(B)$$
$$P(\neg B|LB) \propto P(LB|\neg B)P(\neg B) \propto 0.25(1 - P(B))$$

Thus we cannot decide the probability without some information about the prior probability of blue taxis, $P(B)$. For example, if we knew that all taxis were blue, i.e., $P(B) = 1$, then obviously $P(B|LB) = 1$. On the other hand, if we adopt Laplace's *Principle of Indifference*, which states that propositions can be deemed equally likely in the absence of any differentiating information, then we have $P(B) = 0.5$ and $P(B|LB) = 0.75$. Usually we will have *some* differentiating information, so this principle does not apply.

Given that 9 out of 10 taxis are green, and *assuming the taxi in question is drawn randomly from the taxi population*, we have $P(B) = 0.1$. Hence

$$P(B|LB) \propto 0.75 \times 0.1 \propto 0.075$$
$$P(\neg B|LB) \propto 0.25 \times 0.9 \propto 0.225$$
$$P(B|LB) = \frac{0.075}{0.075+0.225} = 0.25$$
$$P(\neg B|LB) = \frac{0.225}{0.075+0.225} = 0.75$$

13.22 This question is essentially previewing material in Chapter 23 (page 842), but students should have little difficulty in figuring out how to estimate a conditional probability from complete data.

a. The model consists of the prior probability $\mathbf{P}(Category)$ and the conditional probabilities $\mathbf{P}(Word_i|Category)$. For each category c, $\mathbf{P}(Category = c)$ is estimated as the fraction of all documents that are of category c. Similarly, $\mathbf{P}(Word_i = true|Category = c)$ is estimated as the fraction of documents of category c that contain word i.

b. See the answer for 13.17. Here, every evidence variable is observed, since we can tell if any given word appears in a given document or not.

c. The independence assumption is clearly violated in practice. For example, the word pair "artificial intelligence" occurs more frequently in any given document category than would be suggested by multiplying the probabilities of "artificial" and "intelligence".

13.23 This probability model is also appropriate for Minesweeper (Ex. 7.11). If the total number of pits is fixed, then the variables $P_{i,j}$ and $P_{k,l}$ are no longer independent. In general,

$$P(P_{i,j} = true|P_{k,l} = true) < P(P_{i,j} = true|P_{k,l} = false)$$

because learning that $P_{k,l} = true$ makes it less likely that there is a mine at $[i, j]$ (as there are now fewer to spread around). The joint distribution places equal probability on all assignments to $P_{1,2} \ldots P_{4,4}$ that have exactly 3 pits, and zero on all other assignments. Since there are 15 squares, the probability of each 3-pit assignment is $1/\binom{15}{3} = 1/455$.

To calculate the probabilities of pits in $[1,3]$ and $[2,2]$, we start from Figure 13.7. We have to consider the probabilities of complete assignments, since the probability of the "other" region assignment does not cancel out. We can count the total number of 3-pit assignments that are consistent with each partial assignment in 13.7(a) and 13.7(b).

In 13.7(a), there are three partial assignments with $P_{1,3} = true$:

- The first fixes all three pits, so corresponds to 1 complete assignment.
- The second leaves 1 pit in the remaining 10 squares, so corresponds to 10 complete assignments.
- The third also corresponds to 10 complete assignments.

Hence, there are 21 complete assignments with $P_{1,3} = true$.

In 13.7(b), there are two partial assignments with $P_{1,3} = false$:

- The first leaves 1 pit in the remaining 10 squares, so corresponds to 10 complete assignments.
- The second leaves 2 pits in the remaining 10 squares, so corresponds to $\binom{10}{2} = 45$ complete assignments.

Hence, there are 55 complete assignments with $P_{1,3} = false$. Normalizing, we obtain

$$\mathbf{P}(P_{1,3}) = \alpha\langle 21, 55\rangle = \langle 0.276, 0.724\rangle \ .$$

With $P_{2,2} = true$, there are four partial assignments with a total of $\binom{10}{2} + 2 \cdot \binom{10}{1} + \binom{10}{0} = 66$ complete assignments. With $P_{2,2} = false$, there is only one partial assignment with $\binom{10}{1} = 10$ complete assignments. Hence

$$\mathbf{P}(P_{2,2}) = \alpha\langle 66, 10\rangle = \langle 0.868, 0.132\rangle \ .$$

13.24 First we redo the calculations of $P(frontier)$ for each model in Figure 13.6. The three models with $P_{1,3} = true$ have probabilities 0.0001, 0.0099, 0.0099; the two models with $P_{1,3} = false$ have probabilities 0.0001, 0.0099. Then

$$\begin{aligned}\mathbf{P}(P_{1,3} \mid known, b) &= \alpha' \langle 0.01(0.0001 + 0.0099 + 0.0099),\ 0.99(0.0001 + 0.0099)\rangle \\ &\approx \langle 0.1674, 0.8326\rangle \ .\end{aligned}$$

The four models with $P_{2,2} = true$ have probabilities 0.0001, 0.0099, 0.0099, 0.9801; the one model with $P_{2,2} = false$ has probability 0.0001. Then

$$\begin{aligned}\mathbf{P}(P_{2,2} \mid known, b) &= \alpha' \langle 0.01(0.0001 + 0.0099 + 0.0099 + 0.9801),\ 0.99 \times 0.0001\rangle \\ &\approx \langle 0.9902, 0.0098\rangle \ .\end{aligned}$$

This means that [2,2] is almost certain death; a probabilistic agent can figure this out and choose [1,3] or [3,1] instead. Its chance of death at this stage will be 0.1674, while a logical agent choosing at random among the three squares will die with probability $(0.1674 + 0.9902 + 0.1674)/3 = 0.4416$. The reason that [2,2] is so much more likely to be a pit in this case is that, for it *not* to be a pit, *both* of [1,3] and [3,1] must contain pits, which is very unlikely. Indeed, as the prior probability of pits tends to 0, the posterior probability of [2,2] tends to 1.

13.25 The solution for this exercise is omitted. The main modification to the agent in Figure 7.20 is to calculate, after each move, the safety probability for each square that is not provably safe or fatal, and choose the safest if there is no unvisited safe square.

Solutions for Chapter 14
Probabilistic Reasoning

14.1

a. With the random variable C denoting which coin $\{a, b, c\}$ we drew, the network has C at the root and X_1, X_2, and X_3 as children.

The CPT for C is:

C	$P(C)$
a	$1/3$
b	$1/3$
c	$1/3$

The CPT for X_i given C are the same, and equal to:

C	X_1	$P(C)$
a	$heads$	0.2
b	$heads$	0.6
c	$heads$	0.8

b. The coin most likely to have been drawn from the bag given this sequence is the value of C with greatest posterior probability $P(C|\text{2 heads, 1 tails})$. Now,

$$\begin{aligned} P(C|\text{2 heads, 1 tails}) &= P(\text{2 heads, 1 tails}|C)P(C)/P(\text{2 heads, 1 tails}) \\ &\propto P(\text{2 heads, 1 tails}|C)P(C) \\ &\propto P(\text{2 heads, 1 tails}|C) \end{aligned}$$

where in the second line we observe that the constant of proportionality $1/P(\text{2 heads, 1 tails})$ is independent of C, and in the last we observe that $P(C)$ is also independent of the value of C since it is, by hypothesis, equal to $1/3$.

From the Bayesian network we can see that X_1, X_2, and X_3 are conditionally independent given C, so for example

$$\begin{aligned} &P(X_1 = tails, X_2 = heads, X_3 = heads|C = a) \\ &= P(X_1 = tails|C = a)P(X_2 = heads|C = a)P(X_3 = heads|C = a) \\ &= 0.8 \times 0.2 \times 0.2 = 0.032 \end{aligned}$$

Note that since the CPTs for each coin are the same, we would get the same probability above for any ordering of 2 heads and 1 tails. Since there are three such orderings, we have

$$P(2heads, 1tails|C = a) = 3 \times 0.032 = 0.096.$$

Similar calculations to the above find that

$$P(2heads, 1tails|C = b) = 0.432$$
$$P(2heads, 1tails|C = c) = 0.384$$

showing that coin b is most likely to have been drawn.

Alternatively, one could directly compute the value of $P(C|2 \text{ heads, } 1 \text{ tails})$.

14.2 This question is quite tricky and students may require additional guidance, particularly on the last part. It does, however, help them become comfortable with operating on complex sum-of-product expressions, which are at the heart of graphical models.

a. By Equations (13.3) and (13.6), we have

$$P(z \,|\, y) = \frac{P(y,z)}{P(y)} = \frac{\sum_x P(x,y,z)}{\sum_{x,z'} P(x,y,z')} \; .$$

b. By Equation (14.1), this can be written as

$$P(z \,|\, y) = \frac{\sum_x \theta(x)\theta(y \,|\, x)\theta(z \,|\, y)}{\sum_{x,z'} \theta(x)\theta(y \,|\, x)\theta(z' \,|\, y)} \; .$$

c. For students who are not familiar with direct manipulation of summation expressions, the expanding-out step makes it a bit easier to see how to simplify the expressions. Expanding out the sums, collecting terms, using the sum-to-1 property of the parameters, and finally cancelling, we have

$$\begin{aligned} P(z \,|\, y) &= \frac{\theta(x)\theta(y \,|\, x)\theta(z \,|\, y) + \theta(\neg x)\theta(y \,|\, \neg x)\theta(z \,|\, y)}{\theta(x)\theta(y \,|\, x)\theta(z \,|\, y)+\theta(x)\theta(y \,|\, x)\theta(\neg z \,|\, y)+\theta(\neg x)\theta(y \,|\, \neg x)\theta(z \,|\, y)+\theta(\neg x)\theta(y \,|\, \neg x)\theta(\neg z \,|\, y)} \\ &= \frac{\theta(z \,|\, y)\; [\theta(x)\theta(y \,|\, x) + \theta(\neg x)\theta(y \,|\, \neg x)]}{[\theta(x)\theta(y \,|\, x) + \theta(\neg x)\theta(y \,|\, \neg x)]\; [\theta(z \,|\, y) + \theta(\neg z \,|\, y)]} \\ &= \frac{\theta(z \,|\, y)\; [\theta(x)\theta(y \,|\, x) + \theta(\neg x)\theta(y \,|\, \neg x)]}{[\theta(x)\theta(y \,|\, x) + \theta(\neg x)\theta(y \,|\, \neg x)]} \\ &= \theta(z \,|\, y) \; . \end{aligned}$$

If, instead, students are prepared to work on the summations directly, the key step is moving the sum over z' inwards::

$$\begin{aligned} P(z \,|\, y) &= \frac{\theta(z \,|\, y)\; \sum_x \theta(x)\theta(y \,|\, x)}{\sum_x \theta(x)\theta(y \,|\, x)\; \sum_{z'} \theta(z' \,|\, y)} \\ &= \frac{\theta(z \,|\, y)\; \sum_x \theta(x)\theta(y \,|\, x)}{\sum_x \theta(x)\theta(y \,|\, x)} \\ &= \theta(z \,|\, y) \; . \end{aligned}$$

(Note that the first printing has a typo, asking for $\theta(x \,|\, y)$ instead of $\theta(z \,|\, y)$.)

d. The general case is a bit more difficult—the key to a simple proof is figuring out how to split up all the variables. First, however, we need a little lemma: for any set of variables $\mathbf{V}$, we have

$$\sum_{\mathbf{v}} \prod_i \theta(v_i \,|\, pa(V_i)) = 1 \; .$$

This generalizes the sum-to-1 rule for a single variable, and is easily proved by induction given any topological ordering for the variables in **V**.

One of the principal rules for manipulating nested summations is that a particular summation can be pushed to the right as long as all occurrences of that variable remain to the right of the summation. For this reason, the *descendants* of Z, which we will call **U**, are a very useful subset of the variables in the network. In particular, they have the property that they cannot be parents of any other variable in the network. (If there was such a variable, it would be a descendant of Z by definition!) We will divide the variables into Z, **Y** (the parents of Z), **U** (the descendants of Z), and **X** (all other variables). We know that variables in **X** and **Y** have no parents in Z and **U**. So we have

$$
\begin{aligned}
P(z \mid \mathbf{y}) &= \frac{\sum_{\mathbf{x},\mathbf{u}} P(\mathbf{x}, \mathbf{y}, z, \mathbf{u})}{\sum_{\mathbf{x},z',\mathbf{u}} P(\mathbf{x}, \mathbf{y}, z', \mathbf{u})} \\
&= \frac{\sum_{\mathbf{x},\mathbf{u}} \prod_i \theta(x_i \mid pa(X_i)) \prod_j \theta(y_j \mid pa(Y_j)) \theta(z \mid \mathbf{y}) \prod_k \theta(u_k \mid pa(U_k))}{\sum_{\mathbf{x},z',\mathbf{u}} \prod_i \theta(x_i \mid pa(X_i)) \prod_j \theta(y_j \mid pa(Y_j)) \theta(z' \mid \mathbf{y}) \prod_k \theta(u_k \mid pa(U_k))} \\
&= \frac{\sum_{\mathbf{x}} \prod_i \theta(x_i \mid pa(X_i)) \prod_j \theta(y_j \mid pa(Y_j)) \theta(z \mid \mathbf{y}) \sum_{\mathbf{u}} \prod_k \theta(u_k \mid pa(U_k))}{\sum_{\mathbf{x}} \prod_i \theta(x_i \mid pa(X_i)) \prod_j \theta(y_j \mid pa(Y_j)) \sum_{z'} \theta(z' \mid \mathbf{y}) \sum_{\mathbf{u}} \prod_k \theta(u_k \mid pa(U_k))} \\
&\qquad \text{(moving the sums in as far as possible)} \\
&= \frac{\sum_{\mathbf{x}} \prod_i \theta(x_i \mid pa(X_i)) \prod_j \theta(y_j \mid pa(Y_j)) \theta(z \mid \mathbf{y})}{\sum_{\mathbf{x}} \prod_i \theta(x_i \mid pa(X_i)) \prod_j \theta(y_j \mid pa(Y_j)) \sum_{z'} \theta(z' \mid \mathbf{y})} \\
&\qquad \text{(using the generalized sum-to-1 rule for } \mathbf{u}) \\
&= \frac{\sum_{\mathbf{x}} \prod_i \theta(x_i \mid pa(X_i)) \prod_j \theta(y_j \mid pa(Y_j)) \theta(z \mid \mathbf{y})}{\sum_{\mathbf{x}} \prod_i \theta(x_i \mid pa(X_i)) \prod_j \theta(y_j \mid pa(Y_j))} \\
&\qquad \text{(using the sum-to-1 rule for } z') \\
&= \theta(z \mid \mathbf{y}) \, .
\end{aligned}
$$

14.3

a. Suppose that X and Y share l parents. After the reversal Y will gain $m-l$ new parents, the $m-l$ original parents of X that it does not share with Y, and loses one parent: X. After the reversal X will gain $n-l$ new parents, the $n-l-1$ original parents of Y that it does not share with X and isn't X itself, and plus Y. So, after the reversal Y will have $n+(m-l-1) = m+(n-l-1)$ parents, and X will have $m+(n-l) = n+(m-l)$ parents.

Observe that $m-l \geq 0$, since this is the number of original parents of X not shared with Y, and that $n-l-1 \geq 0$, since this is the number of original parents of Y not shared with X and not equal to X. This shows the number of parameters can only increase: before we had $k^m + k^n$, after we have $k^{m+(n-l-1)} + k^{n+(m-l)}$.

(As a sanity check on our counting above, if are reversing a single arc without any extra parents, we have $l = 0$, $m = 0$, and $n = 1$; the previous formulas say we will have $m' = 0$ and $n' = 1$ afterwards, which is correct.)

b. For the number of parameters to remain constant, assuming that $k > 1$, requires by our

previous calculation that $m - l = 0$ and $n - l - 1 = 0$. This holds exactly when X and Y share all their parents originally (except Y also has X as a parent).

c. For clarity, denote by $P'(Y|U,V,W)$ and $P'(X|U,V,W,Y)$ the new CPTs, and note that the set of variables $V \cup W$ does not include X. It suffices to show that

$$P'(X,Y|U,V,W) = P(X,Y|U,V,W)$$

To see this, let D denote the variables, outside of $\{X,Y\} \cup U \cup V \cup W$, which have either X or Y as ancestor in the original network, and $\overline{D}$ those which don't. Since the arc reversed graph only adds or removes arcs incoming to X or Y, it cannot change which variables lie in D or $\overline{D}$. We then have

$$\begin{aligned} P(D,\overline{D},X,Y,U,V,W) &= P(\overline{D},U,V,W)P(X,Y|U,V,W)P(D|X,Y,U,V,W) \\ &= P'(\overline{D},U,V,W)P(X,Y|U,V,W)P(D|X,Y,U,V,W) \\ &= P'(\overline{D},U,V,W)P(X,Y|U,V,W)P'(D|X,Y,U,V,W) \\ &= P'(\overline{D},U,V,W)P'(X,Y|U,V,W)P'(D|X,Y,U,V,W) \\ &= P'(D,\overline{D},X,Y,U,V,W) \end{aligned}$$

the second as arc reversal does not change the CPTs of variables in $\overline{D}, U, V, W$ contains all its parents, the third as if we condition on X, Y, U, V, W the original and arc-reversed Bayesian networks are the same, and the fourth by hypothesis.

Then, calculating:

$$\begin{aligned} &P'(X,Y|U,V,W) \\ &= P'(Y|U,V,W)P'(X|U,V,W,Y) \\ &= \left(\sum_x P(Y|V,W,x)P(x|U,V)\right) P(Y|X,V,W)P(X|U,V) \\ &= \left(\sum_x P(Y|U,V,W,x)P(x|U,V)/P(Y|U,V,W)\right) P(Y|X,V,W)P(X|U,V) \\ &= \left(\sum_x \frac{P(Y,U,V,W,x)P(x,U,V)P(U,V,W)}{P(U,V,W,x)P(U,V)P(Y,U,V,W)}\right) P(Y|X,V,W)P(X|U,V) \\ &= \left(\sum_x P(x|Y,U,V,W)P(x|U,V)/P(x|U,V,W)\right) P(Y|X,V,W)P(X|U,V) \\ &= \left(\sum_x P(x|Y,U,V,W)\right) P(Y|X,V,W)P(X|U,V) \\ &= P(Y|X,V,W)P(X|U,V) \end{aligned}$$

where the third step follows as V, W, x is the parent set of Y it's conditionally independent of U, and the second to last step follows as U, V is the parent set of X it's conditionally independent of x.

14.4

a. Yes. Numerically one can compute that $P(B, E) = P(B)P(E)$. Topologically B and E are d-separated by A.

b. We check whether $P(B, E|a) = P(B|a)P(E|a)$. First computing $P(B, E|a)$

$$\begin{aligned} P(B, E|a) &= \alpha P(a|B, E)P(B, E) \\ &= \alpha \begin{cases} .95 \times 0.001 \times 0.002 & \text{if } B = b \text{ and } E = e \\ .94 \times 0.001 \times 0.998 & \text{if } B = b \text{ and } E = \neg e \\ .29 \times 0.999 \times 0.002 & \text{if } B = \neg b \text{ and } E = e \\ .001 \times 0.999 \times 0.998 & \text{if } B = \neg b \text{ and } E = \neg e \end{cases} \\ &= \alpha \begin{cases} 0.0008 & \text{if } B = b \text{ and } E = e \\ 0.3728 & \text{if } B = b \text{ and } E = \neg e \\ 0.2303 & \text{if } B = \neg b \text{ and } E = e \\ 0.3962 & \text{if } B = \neg b \text{ and } E = \neg e \end{cases} \end{aligned}$$

where α is a normalization constant. Checking whether $P(b, e|a) = P(b|a)P(e|a)$ we find

$$P(b, e|a) = 0.0008 \neq 0.0863 = 0.3736 \times 0.2311 = P(b|a)P(e|a)$$

showing that B and E are not conditionally independent given A.

14.5 The question is not very specific about what "remove" means. When a node is a leaf node with no children, removing it and its CPT leaves the rest of the network unchanged. When a node has children, however, we have to decide what to do about the CPTs of those children, since one parent has disappeared. The only reasonable interpretation is that removal has to leave the posterior of all other variables unchanged *regardless* of what new CPTs are supplied for Y's children.

a. Let $\mathbf{X}$ be the set of all variables in the Bayesian Network except for Y and $MB(Y)$. Since Y is conditionally independent of all other nodes in the network, given $MB(Y)$, we have $P(\mathbf{X}|Y, mb(Y)) = P(\mathbf{X}|mb(Y)) = \alpha P(\mathbf{X}, mb(Y))$. By definition, the parents of Y's children are a subset of $\{Y\} \cup MB(Y)$, so not include any variables in $\mathbf{X}$. Hence, if we expand out $P(\mathbf{X}, mb(Y))$ in terms of CPT entries, all the CPT entries for Y's children are constants that can be subsumed in α.

b. We have argued that any removal operation as described above leaves posteriors unchanged; therefore, both algorithms will still return the correct answers.

14.6

a. (c) matches the equation. The equation describes absolute independence of the three genes, which requires no links among them.

b. (a) and (b). The *assertions* are the *absent* links; the extra links in (b) may be unnecessary but they do not assert an actual dependence. (c) asserts independence of genes which contradicts the inheritance scenario.

c. (a) is best. (b) has spurious links among the H variables, which are not directly causally connected in the scenario described. (In reality, handedness may also be passed down by example/training.)

d. Notice that the $l \rightarrow r$ and $r \rightarrow l$ mutations cancel when the parents have different genes, so we still get 0.5.

G_{mother}	G_{father}	$P(G_{child}=l\|\ldots)$	$P(G_{child}=r\|\ldots)$
l	l	$1-m$	m
l	r	0.5	0.5
r	l	0.5	0.5
r	r	m	$1-m$

e. This is a straightforward application of conditioning:

$$\begin{aligned} P(G_{child}=l) &= \sum_{g_m,g_f} P(G_{child}=l|g_m,g_f)P(g_m,g_f) \\ &= \sum_{g_m,g_f} P(G_{child}=l|g_m,g_f)P(g_m)P(g_f) \\ &= (1-m)q^2 + 0.5q(1-q) + 0.5(1-q)q + m(1-q)^2 \\ &= q^2 - mq^2 + q - q^2 + m - 2mq + mq^2 \\ &= q + m - 2mq \end{aligned}$$

f. Equilibrium means that $P(G_{child}=l)$ (the prior, with no parent information) must equal $P(G_{mother}=l)$ and $P(G_{father}=l)$, i.e.,

$$q + m - 2mq = q, \text{ hence } q = 0.5.$$

But few humans are left-handed ($x \approx 0.08$ in fact), so something is wrong with the symmetric model of inheritance and/or manifestation. The "high-school" explanation is that the "right-hand gene is dominant," i.e., preferentially inherited, but current studies suggest also that handedness is not the result of a single gene and may also involve cultural factors. An entire journal (*Laterality*) is devoted to this topic.

14.7 These proofs are tricky for those not accustomed to manipulating probability expressions, and students may require some hints.

a. There are several ways to prove this. Probably the simplest is to work directly from the global semantics. First, we rewrite the required probability in terms of the full joint:

$$\begin{aligned} P(x_i|x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n) &= \frac{P(x_1,\ldots,x_n)}{P(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)} \\ &= \frac{P(x_1,\ldots,x_n)}{\sum_{x_i} P(x_1,\ldots,x_n)} \\ &= \frac{\prod_{j=1}^n P(x_j|parentsX_j)}{\sum_{x_i}\prod_{j=1}^n P(x_j|parentsX_j)} \end{aligned}$$

Now, all terms in the product in the denominator that do not contain x_i can be moved outside the summation, and then cancel with the corresponding terms in the numerator. This just leaves us with the terms that do mention x_i, i.e., those in which X_i is a child

or a parent. Hence, $P(x_i|x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)$ is equal to

$$\frac{P(x_i|parentsX_i)\prod_{Y_j\in Children(X_i)}P(y_j|parents(Y_j))}{\sum_{x_i}P(x_i|parentsX_i)\prod_{Y_j\in Children(X_i)}P(y_j|parents(Y_j))}$$

Now, by reversing the argument in part (b), we obtain the desired result.

b. This is a relatively straightforward application of Bayes' rule. Let $\mathbf{Y}=Y_1,\ldots,y_\ell$ be the children of X_i and let $\mathbf{Z}_j$ be the parents of Y_j other than X_i. Then we have

$$\begin{aligned}
&\mathbf{P}(X_i|MB(X_i))\\
&= \mathbf{P}(X_i|Parents(X_i),\mathbf{Y},\mathbf{Z}_1,\ldots,\mathbf{Z}_\ell)\\
&= \alpha\mathbf{P}(X_i|Parents(X_i),\mathbf{Z}_1,\ldots,\mathbf{Z}_\ell)\mathbf{P}(\mathbf{Y}|Parents(X_i),X_i,\mathbf{Z}_1,\ldots,\mathbf{Z}_\ell)\\
&= \alpha\mathbf{P}(X_i|Parents(X_i))\mathbf{P}(\mathbf{Y}|X_i,\mathbf{Z}_1,\ldots,\mathbf{Z}_\ell)\\
&= \alpha\mathbf{P}(X_i|Parents(X_i))\prod_{Y_j\in Children(X_i)}P(Y_j|Parents(Y_j))
\end{aligned}$$

where the derivation of the third line from the second relies on the fact that a node is independent of its nondescendants given its children.

14.8 Adding variables to an existing net can be done in two ways. Formally speaking, one should insert the variables into the variable ordering and rerun the network construction process from the point where the first new variable appears. Informally speaking, one never really builds a network by a strict ordering. Instead, one asks what variables are direct causes or influences on what other ones, and builds local parent/child graphs that way. It is usually easy to identify where in such a structure the new variable goes, but one must be very careful to check for possible induced dependencies downstream.

a. $IcyWeather$ is not caused by any of the car-related variables, so needs no parents. It directly affects the battery and the starter motor. $StarterMotor$ is an additional precondition for $Starts$. The new network is shown in Figure S14.1.

b. Reasonable probabilities may vary a lot depending on the kind of car and perhaps the personal experience of the assessor. The following values indicate the general order of magnitude and relative values that make sense:

- A reasonable prior for IcyWeather might be 0.05 (perhaps depending on location and season).
- $P(Battery|IcyWeather)=0.95$, $P(Battery|\neg IcyWeather)=0.997$.
- $P(StarterMotor|IcyWeather)=0.98$, $P(Battery|\neg IcyWeather)=0.999$.
- $P(Radio|Battery)=0.9999$, $P(Radio|\neg Battery)=0.05$.
- $P(Ignition|Battery)=0.998$, $P(Ignition|\neg Battery)=0.01$.
- $P(Gas)=0.995$.
- $P(Starts|Ignition,StarterMotor,Gas)=0.9999$, other entries 0.0.
- $P(Moves|Starts)=0.998$.

c. With 8 Boolean variables, the joint has 2^8-1 = 255 independent entries.

d. Given the topology shown in Figure S14.1, the total number of independent CPT entries is 1+2+2+2+2+1+8+2= 20.

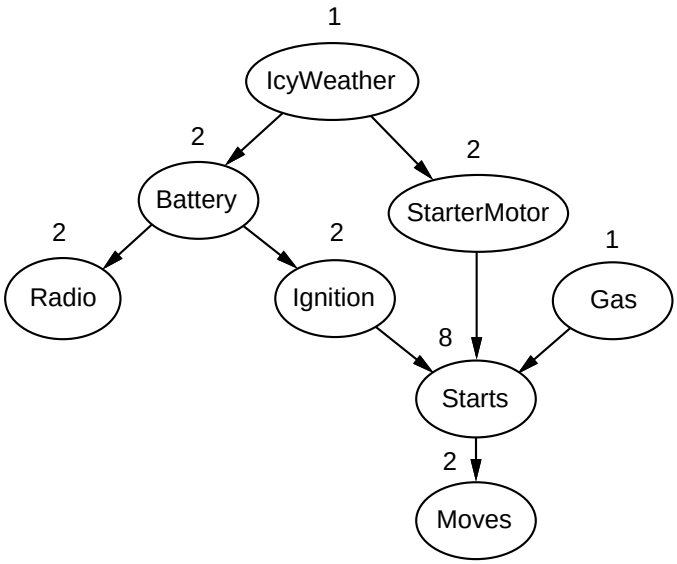


Figure S14.1 Car network amended to include $IcyWeather$ and $StarterMotorWorking$ (SMW).

e. The CPT for $Starts$ describes a set of nearly necessary conditions that are together almost sufficient. That is, all the entries are nearly zero except for the entry where all the conditions are true. That entry will be not quite 1 (because there is always some other possible fault that we didn't think of), but as we add more conditions it gets closer to 1. If we add a $Leak$ node as an extra parent, then the probability is exactly 1 when all parents are true. We can relate noisy-AND to noisy-OR using de Morgan's rule: $A \wedge B \equiv \neg(\neg A \vee \neg B)$. That is, noisy-AND is the same as noisy-OR except that the polarities of the parent and child variables are reversed. In the noisy-OR case, we have

$$P(Y = true|x_1, \ldots, x_k) = 1 - \prod_{\{i:x_i = true\}} q_i$$

where q_i is the probability that the *presence* of the ith parent *fails* to cause the child to be *true*. In the noisy-AND case, we can write

$$P(Y = true|x_1, \ldots, x_k) = \prod_{\{i:x_i = false\}} r_i$$

where r_i is the probability that the *absence* of the ith parent *fails* to cause the child to be *false* (e.g., it is magically bypassed by some other mechanism).

14.9 This exercise is a little tricky and will appeal to more mathematically oriented students.

a. The basic idea is to multiply the two densities, match the result to the standard form for a multivariate Gaussian, and hence identify the entries in the inverse covariance matrix. Let's begin by looking at the multivariate Gaussian. From page 982 in Appendix A we

have

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n|\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}\left((\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)} ,$$

where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Sigma}$ is the covariance matrix. In our case, $\mathbf{x}$ is $(x_1\ x_2)^\top$ and let the (as yet) unknown $\boldsymbol{\mu}$ be $(m_1\ m_2)^\top$. Suppose the inverse covariance matrix is

$$\boldsymbol{\Sigma}^{-1} = \begin{pmatrix} c & d \\ d & e \end{pmatrix}$$

Then, if we multiply out the exponent, we obtain

$$-\tfrac{1}{2}\left((\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) = \\ -\tfrac{1}{2}\cdot c(x_1-m_1)^2 + 2d(x_1-m_1)(x_2-m_2) + f(x_2-m_2)^2$$

Looking at the distributions themselves, we have

$$P(x_1) = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-(x_1-\mu_1)^2/(2\sigma_1^2)}$$

and

$$P(x_2|x_1) = \frac{1}{\sigma_2\sqrt{2\pi}} e^{-(x_2-(ax_1+b))^2/(2\sigma_2^2)}$$

hence

$$P(x_1,x_2) = \frac{1}{\sigma_1\sigma_2(2\pi)} e^{-(\sigma_2^2(x_2-(ax_1+b))^2+\sigma_1^2(x_2-(ax_1+b))^2)/(2\sigma_1^2\sigma_2^2)}$$

We can obtain equations for c, d, and e by picking out the coefficients of x_1^2, x_1x_2, and x_2^2:

$$\begin{aligned} c &= (\sigma_2^2 + a^2\sigma_1^2)/\sigma_1^2\sigma_2^2 \\ 2d &= -2a/\sigma_2^2 \\ e &= 1/\sigma_2^2 \end{aligned}$$

We can check these by comparing the normalizing constants.

$$\frac{1}{\sigma_1\sigma_2(2\pi)} = \frac{1}{\sqrt{(2\pi)^n|\boldsymbol{\Sigma}|}} = \frac{1}{(2\pi)\sqrt{1/|\boldsymbol{\Sigma}^{-1}|}} = \frac{1}{(2\pi)\sqrt{1/(ce-d^2)}}$$

from which we obtain the constraint

$$ce - d^2 = 1/\sigma_1^2\sigma_2^2$$

which is easily confirmed. Similar calculations yield m_1 and m_2, and plugging the results back shows that $P(x_1,x_2)$ is indeed multivariate Gaussian. The covariance matrix is

$$\boldsymbol{\Sigma} = \begin{pmatrix} c & d \\ d & e \end{pmatrix}^{-1} = \frac{1}{ce-d^2}\begin{pmatrix} e & -d \\ -d & c \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & a\sigma_1^2 \\ a\sigma_1^2 & \sigma_2^2 + a^2\sigma_1^2 \end{pmatrix}$$

b. The induction is on n, the number of variables. The base case for $n=1$ is trivial. The inductive step asks us to show that if any $P(x_1,\ldots,x_n)$ constructed with linear–Gaussian conditional densities is multivariate Gaussian, then any $P(x_1,\ldots,x_n,x_{n+1})$

constructed with linear–Gaussian conditional densities is also multivariate Gaussian. Without loss of generality, we can assume that X_{n+1} is a leaf variable added to a network defined in the first n variables. By the product rule we have

$$\begin{aligned} P(x_1, \ldots, x_n, x_{n+1}) &= P(x_{n+1}|x_1, \ldots, x_n)P(x_1, \ldots, x_n) \\ &= P(x_{n+1}|parents(X_{n+1}))P(x_1, \ldots, x_n) \end{aligned}$$

which, by the inductive hypothesis, is the product of a linear Gaussian with a multivariate Gaussian. Extending the argument of part (a), this is in turn a multivariate Gaussian of one higher dimension.

14.10

a. With multiple continuous parents, we must find a way to map the parent value vector to a single threshold value. The simplest way to do this is to take a linear combination of the parent values.

b. For ordered values $y_1 < y_2 < \cdots < y_d$, we assume some unobserved continuous dependent variable y^* that is normally distributed conditioned on the parent variables, and define cutpoints c_j such that $Y = y_j$ iff $c_{j-1} \leq y^* \leq c_j$. The probability of this event is given by subtracting the cumulative distributions at the adjacent cutpoints.

The unordered case is not obviously meaningful if we insist that the relationship between parents and child be mediated by a single, real-valued, normally distributed variable.

14.11 This question exercises many aspects of the student's understanding of Bayesian networks and uncertainty.

a. A suitable network is shown in Figure S14.2. The key aspects are: the failure nodes are parents of the sensor nodes, and the temperature node is a parent of both the gauge and the gauge failure node. It is exactly this kind of correlation that makes it difficult for humans to understand what is happening in complex systems with unreliable sensors.

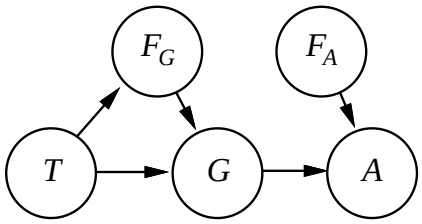


Figure S14.2 A Bayesian network for the nuclear alarm problem.

b. No matter which way the student draws the network, it should not be a polytree because of the fact that the temperature influences the gauge in two ways.

c. The CPT for G is shown below. Students should pay careful attention to the semantics of F_G, which is true when the gauge is *faulty*, i.e., *not* working.

	$T = Normal$		$T = High$	
	F_G	$\neg F_G$	F_G	$\neg F_G$
$G = Normal$	y	x	$1-y$	$1-x$
$G = High$	$1-y$	$1-x$	y	x

d. The CPT for A is as follows:

	$G = Normal$		$G = High$	
	F_A	$\neg F_A$	F_A	$\neg F_A$
A	0	0	0	1
$\neg A$	1	1	1	0

e. This part actually asks the student to do something usually done by Bayesian network algorithms. The great thing is that doing the calculation without a Bayesian network makes it easy to see the nature of the calculations that the algorithms are systematizing. It illustrates the magnitude of the achievement involved in creating complete and correct algorithms.

Abbreviating $T = High$ and $G = High$ by T and G, the probability of interest here is $P(T|A, \neg F_G, \neg F_A)$. Because the alarm's behavior is deterministic, we can reason that if the alarm is working and sounds, G must be $High$. Because F_A and A are d-separated from T, we need only calculate $P(T|\neg F_G, G)$.

There are several ways to go about doing this. The "opportunistic" way is to notice that the CPT entries give us $P(G|T, \neg F_G)$, which suggests using the generalized Bayes' Rule to switch G and T with $\neg F_G$ as background:

$$P(T|\neg F_G, G) \propto P(G|T, \neg F_G)P(T|\neg F_G)$$

We then use Bayes' Rule again on the last term:

$$P(T|\neg F_G, G) \propto P(G|T, \neg F_G)P(\neg F_G|T)P(T)$$

A similar relationship holds for $\neg T$:

$$P(\neg T|\neg F_G, G) \propto P(G|\neg T, \neg F_G)P(\neg F_G|\neg T)P(\neg T)$$

Normalizing, we obtain

$$P(T|\neg F_G, G) = \frac{P(G|T,\neg F_G)P(\neg F_G|T)P(T)}{P(G|T,\neg F_G)P(\neg F_G|T)P(T)+P(G|\neg T,\neg F_G)P(\neg F_G|\neg T)P(\neg T)}$$

The "systematic" way to do it is to revert to joint entries (noticing that the subgraph of T, G, and F_G is completely connected so no loss of efficiency is entailed). We have

$$P(T|\neg F_G, G) = \frac{P(T, \neg F_G, G)}{P(G, \neg F_G)} = \frac{P(T, \neg F_G, G)}{P(T, G, \neg F_G) + P(T, G, \neg F_G)}$$

Now we use the chain rule formula (Equation 15.1 on page 439) to rewrite the joint entries as CPT entries:

$$P(T|\neg F_G, G) = \frac{P(T)P(\neg F_G|T)P(G|T,\neg F_G)}{P(T)P(\neg F_G|T)P(G|T,\neg F_G)+P(\neg T)P(\neg F_G|\neg T)P(G|\neg T,\neg F_G)}$$

which of course is the same as the expression arrived at above. Letting $P(T) = p$, $P(F_G|T) = g$, and $P(F_G|\neg T) = h$, we get

$$P(T|\neg F_G, G) = \frac{p(1-g)(1-x)}{p(1-g)(1-x) + (1-p)(1-h)x}$$

14.12

a. Although (i) in some sense depicts the "flow of information" during calculation, it is clearly incorrect as a network, since it says that given the measurements M_1 and M_2, the number of stars is independent of the focus. (ii) correctly represents the causal structure: each measurement is influenced by the actual number of stars and the focus, and the two telescopes are independent of each other. (iii) shows a correct but more complicated network—the one obtained by ordering the nodes M_1, M_2, N, F_1, F_2. If you order M_2 before M_1 you would get the same network except with the arrow from M_1 to M_2 reversed.

b. (ii) requires fewer parameters and is therefore better than (iii).

c. To compute $\mathbf{P}(M_1|N)$, we will need to condition on F_1 (that is, consider both possible cases for F_1, weighted by their probabilities).

$$\begin{aligned}\mathbf{P}(M_1|N) &= \mathbf{P}(M_1|N, F_1)\mathbf{P}(F_1|N) + \mathbf{P}(M_1|N, \neg F_1)\mathbf{P}(\neg F_1|N) \\ &= \mathbf{P}(M_1|N, F_1)\mathbf{P}(F_1) + \mathbf{P}(M_1|N, \neg F_1)\mathbf{P}(\neg F_1)\end{aligned}$$

Let f be the probability that the telescope is out of focus. The exercise states that this will cause an "undercount of three or more stars," but if N = 3 or less the count will be 0 if the telescope is out of focus. If it is in focus, then we will assume there is a probability of e of counting one two few, and e of counting one too many. The rest of the time $(1 - 2e)$, the count will be accurate. Then the table is as follows:

	$N = 1$	$N = 2$	$N = 3$
$M_1 = 0$	f + e(1-f)	f	f
$M_1 = 1$	(1-2e)(1-f)	e(1-f)	0.0
$M_1 = 2$	e(1-f)	(1-2e)(1-f)	e(1-f)
$M_1 = 3$	0.0	e(1-f)	(1-2e)(1-f)
$M_1 = 4$	0.0	0.0	e(1-f)

Notice that each column has to add up to 1. Reasonable values for e and f might be 0.05 and 0.002.

d. This question causes a surprising amount of difficulty, so it is important to make sure students understand the reasoning behind an answer. One approach uses the fact that it is easy to reason in the forward direction, that is, try each possible number of stars N and see whether measurements $M_1 = 1$ and $M_2 = 3$ are possible. (This is a sort of mental simulation of the physical process.) An alternative approach is to enumerate the possible focus states and deduce the value of N for each. Either way, the solutions are $N = 2$, 4, or ≥ 6.

e. We cannot calculate the most likely number of stars without knowing the prior distribution $P(N)$. Let the priors be p_2, p_4, and $p_{\geq 6}$. The posterior for $N=2$ is $p_2 e^2(1-f)^2$; for $N=4$ it is at most $p_4 ef$ (at most, because with $N=4$ the out-of-focus telescope could measure 0 instead of 1); for $N \geq 6$ it is at most $p_{\geq 6} f^2$. If we assume that the priors are roughly comparable, then $N=2$ is most likely because we are told that f is much smaller than e.

For follow-up or alternate questions, it is easy to come up with endless variations on the same theme involving sensors, failure nodes, hidden state. One can also add in complex mechanisms, as for the $Starts$ variable in exercise 14.1.

14.13 The symbolic expression evaluated by the enumeration algorithm is

$$\begin{aligned}\mathbf{P}(N \mid M_1=2, M_2=2) &= \alpha \sum_{f_1,f_2} \mathbf{P}(f_1, f_2, N, M_1=2, M_2=2) \\ &= \alpha \sum_{f_1,f_2} P(f_1)P(f_2)\mathbf{P}(N)P(M_1=2 \mid f_1, N)P(M_2=2 \mid f_2, N) \ .\end{aligned}$$

Because an out-of-focus telescope cannot report 2 stars in the given circumstances, the only non-zero term in the summation is for $F_1 = F_2 = false$, so the answer is

$$\begin{aligned}\mathbf{P}(N \mid M_1=2, M_2=2) &= \alpha(1-f)(1-f)\langle p_1, p_2, p_3\rangle\langle e, (1-2e), e\rangle\langle e, (1-2e), e\rangle \\ &= \alpha'\langle p_1 e^2, p_2(1-2e)^2, p_3 e^2\rangle \ .\end{aligned}$$

14.14

a. The network asserts (ii) and (iii). (For (iii), consider the Markov blanket of M.)

b. $P(b, i, \neg m, g, j) = P(b)P(\neg m)P(i|b, \neg m)P(g|b, i, \neg m)P(j|g)$
$= .9 \times .9 \times .5 \times .8 \times .9 = .2916$

c. Since B, I, M are fixed true in the evidence, we can treat G as having a prior of 0.9 and just look at the submodel with G, J:
$\mathbf{P}(J|b, i, m) = \alpha \sum_g \mathbf{P}(J, g) = \alpha[\mathbf{P}(J, g) + \mathbf{P}(J, \neg g)]$
$= \alpha[\langle P(j, g), P(\neg j, g)\rangle + \langle P(j, \neg g), P(\neg j, \neg g)\rangle$
$= \alpha[\langle .81, .09\rangle + \langle 0, 0.1\rangle] = \langle .81, .19\rangle$
That is, the probability of going to jail is 0.81.

d. Intuitively, a person cannot be found guilty if not indicted, regardless of whether they broke the law and regardless of the prosecutor. This is what the CPT for G says; so G is context-specifically independent of B and M given $I = false$.

e. A pardon is unnecessary if the person is not indicted or not found guilty; so I and G are parents of P. One could also add B and M as parents of P, since a pardon is more likely if the person is actually innocent and if the prosecutor is politically motivated. (There are other causes of $Pardon$, such as $LargeDonationToPresidentsParty$, but such variables are not currently in the model.) The pardon (presumably) is a get-out-of-jail-free card, so P is a parent of J.

14.15 This question definitely helps students get a solid feel for variable elimination. Students may need some help with the last part if they are to do it properly.

a.

$$\begin{aligned}
&P(B|j,m)\\
&= \alpha P(B)\sum_e P(e)\sum_a P(a|b,e)P(j|a)P(m|a)\\
&= \alpha P(B)\sum_e P(e)\left[.9\times.7\times\begin{pmatrix}.95 & .29\\ .94 & .001\end{pmatrix}+.05\times.01\times\begin{pmatrix}.05 & .71\\ .06 & .999\end{pmatrix}\right]\\
&= \alpha P(B)\sum_e P(e)\begin{pmatrix}.598525 & .183055\\ .59223 & .0011295\end{pmatrix}\\
&= \alpha P(B)\left[.002\times\begin{pmatrix}.598525\\ .183055\end{pmatrix}+.998\times\begin{pmatrix}.59223\\ .0011295\end{pmatrix}\right]\\
&= \alpha\begin{pmatrix}.001\\ .999\end{pmatrix}\times\begin{pmatrix}.59224259\\ .001493351\end{pmatrix}\\
&= \alpha\begin{pmatrix}.00059224259\\ .0014918576\end{pmatrix}\\
&\approx \langle .284, .716\rangle
\end{aligned}$$

b. Including the normalization step, there are 7 additions, 16 multiplications, and 2 divisions. The enumeration algorithm has two extra multiplications.

c. To compute $\mathbf{P}(X_1|X_n = \mathit{true})$ using enumeration, we have to evaluate two complete binary trees (one for each value of X_1), each of depth $n-2$, so the total work is $O(2^n)$. Using variable elimination, the factors never grow beyond two variables. For example, the first step is

$$\begin{aligned}
&\mathbf{P}(X_1|X_n = \mathit{true})\\
&= \alpha\mathbf{P}(X_1)\ldots\sum_{x_{n-2}} P(x_{n-2}|x_{n-3})\sum_{x_{n-1}} P(x_{n-1}|x_{n-2})P(X_n = \mathit{true}|x_{n-1})\\
&= \alpha\mathbf{P}(X_1)\ldots\sum_{x_{n-2}} P(x_{n-2}|x_{n-3})\sum_{x_{n-1}} \mathbf{f}_{X_{n-1}}(x_{n-1},x_{n-2})\mathbf{f}_{X_n}(x_{n-1})\\
&= \alpha\mathbf{P}(X_1)\ldots\sum_{x_{n-2}} P(x_{n-2}|x_{n-3})\mathbf{f}_{\overline{X_{n-1}X_n}}(x_{n-2})
\end{aligned}$$

The last line is isomorphic to the problem with $n-1$ variables instead of n; the work done on the first step is a constant independent of n, hence (by induction on n, if you want to be formal) the total work is $O(n)$.

d. Here we can perform an induction on the number of nodes in the polytree. The base case is trivial. For the inductive hypothesis, assume that any polytree with n nodes can be evaluated in time proportional to the size of the polytree (i.e., the sum of the CPT sizes). Now, consider a polytree with $n+1$ nodes. Any node ordering consistent with the topology will eliminate first some leaf node from this polytree. To eliminate any

leaf node, we have to do work proportional to the size of its CPT. Then, *because the network is a polytree*, we are left with *independent* subproblems, one for each parent. Each subproblem takes total work proportional to the sum of its CPT sizes, so the total work for $n + 1$ nodes is proportional to the sum of CPT sizes.

14.16

a. Consider a 3-CNF formula $C_1 \wedge \ldots C_n$ with n clauses where each clause is a disjunct $C_i = (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$ of literals i.e., each ℓ_{ij} is either P_k or $\neg P_k$ for some atomic proposition $P_1, \ldots, P_m$.

Construct a Bayesian network with a (boolean) variable S for the whole formula, C_i for each clause, and P_k for each atomic proposition. We will define parents and CPTs such that for any assignment to the atomic propositions, S is true if and only if the 3-CNF formula is true.

Atomic propositions have no parents, and are true with probability 0.5. Each clause C_i has as its parents the atomic propositions corresponding to the literals ℓ_{i1}, ℓ_{i2}, and ℓ_{i3}). The clause variable is true iff one of its literals is true. Note that this is a deterministic CPT. Finally, S has all the clause variables C_i as its parents, and if true if any only if all clause variables are true.

Notice that $P(S = True) > 0$ if and only if the formula is satisfiable, and exact inference will answer this question.

b. Using the same network as in part (a), notice that $P(S = True) = s2^{-m}$ where s is the number of satisfying assignments to the atomic propositions $P_1, \ldots, P_m$.

14.17

a. To calculate the cumulative distribution of a discrete variable, we start from a vector representation p of the original distribution and a vector P of the same dimension. Then, we loop through i, adding up the p_i values as we go along and setting P_i to the running sum, $\sum_{j=i}^{i} p_j$. To sample from the distribution, we generate a random number r uniformly in $[0, 1]$, and then return x_i for the smallest i such that $P_i \geq r$. A naive way to find this is to loop through i starting at 1 until $P_i \geq r$. This takes $O(k)$ time. A more efficient solution is binary search: start with the full range $[1, k]$, choose i at the midpoint of the range. If $P_i < r$, set the range from i to the upper bound, otherwise set the range from the lower bound to i. After $O(\log k)$ iterations, we terminate when the bounds are identical or differ by 1.

b. If we are generating $N \gg k$ samples, we can afford to preprocess the cumulative distribution. The basic insight required is that *if* the original distribution were uniform, it would be possible to sample in $O(1)$ time by returning $\lceil kr \rceil$. That is, we can index directly into the correct part of the range (analog random access, one might say) instead of searching for it. Now, suppose we divide the range $[0, 1]$ into k equal parts and construct a k-element vector, each of whose entries is a list of all those i for which P_i is in the corresponding part of the range. The i we want is in the list with index $\lceil kr \rceil$. We retrieve this list in $O(1)$ time and search through it in order (as in the naive

implementation). Let n_j be the number of elements in list j. Then the expected runtime is given by

$$\sum_{j=1}^{k} n_j \cdot 1/k = 1/k \cdot \sum_{j=1}^{k} n_j = 1/k \cdot O(k) = O(1)$$

The variance of the runtime can be reduced by further subdividing any part of the range whose list contains more than some small constant number of elements.

c. One way to generate a sample from a univariate Gaussian is to compute the discretized cumulative distribution (e.g., integrating by Taylor's rule) and use the algorithm described above. We can compute the table once and for all for the standard Gaussian (mean 0, variance 1) and then scale each sampled value z to $\sigma z + \mu$. If we had a closed-form, invertible expression for the cumulative distribution $F(x)$, we could sample exactly, simply by returning $F^{-1}(r)$. Unfortunately the Gaussian density is not exactly integrable. Now, the density $\alpha x e^{-x^2/2}$ *is* exactly integrable, and there are cute schemes for using two samples and this density to obtain an exact Gaussian sample. We leave the details to the interested instructor.

d. When querying a continuous variable using Monte carlo inference, an exact closed-form posterior cannot be obtained. Instead, one typically defines discrete ranges, returning a histogram distribution simply by counting the (weighted) number of samples in each range.

14.18

a. There are two uninstantiated Boolean variables ($Cloudy$ and $Rain$) and therefore four possible states.

b. First, we compute the sampling distribution for each variable, conditioned on its Markov blanket.

$$\begin{aligned}
\mathbf{P}(C|r,s) &= \alpha\mathbf{P}(C)\mathbf{P}(s|C)\mathbf{P}(r|C) \\
&= \alpha\langle 0.5, 0.5\rangle\langle 0.1, 0.5\rangle\langle 0.8, 0.2\rangle = \alpha\langle 0.04, 0.05\rangle = \langle 4/9, 5/9\rangle \\
\mathbf{P}(C|\neg r,s) &= \alpha\mathbf{P}(C)\mathbf{P}(s|C)\mathbf{P}(\neg r|C) \\
&= \alpha\langle 0.5, 0.5\rangle\langle 0.1, 0.5\rangle\langle 0.2, 0.8\rangle = \alpha\langle 0.01, 0.20\rangle = \langle 1/21, 20/21\rangle \\
\mathbf{P}(R|c,s,w) &= \alpha\mathbf{P}(R|c)\mathbf{P}(w|s,R) \\
&= \alpha\langle 0.8, 0.2\rangle\langle 0.99, 0.90\rangle = \alpha\langle 0.792, 0.180\rangle = \langle 22/27, 5/27\rangle \\
\mathbf{P}(R|\neg c,s,w) &= \alpha\mathbf{P}(R|\neg c)\mathbf{P}(w|s,R) \\
&= \alpha\langle 0.2, 0.8\rangle\langle 0.99, 0.90\rangle = \alpha\langle 0.198, 0.720\rangle = \langle 11/51, 40/51\rangle
\end{aligned}$$

Strictly speaking, the transition matrix is only well-defined for the variant of MCMC in which the variable to be sampled is chosen randomly. (In the variant where the variables are chosen in a fixed order, the transition probabilities depend on where we are in the ordering.) Now consider the transition matrix.

- Entries on the diagonal correspond to self-loops. Such transitions can occur by sampling *either* variable. For example,
$$q((c,r) \rightarrow (c,r)) = 0.5P(c|r,s) + 0.5P(r|c,s,w) = 17/27$$
- Entries where one variable is changed must sample that variable. For example,
$$q((c,r) \rightarrow (c,\neg r)) = 0.5P(\neg r|c,s,w) = 5/54$$
- Entries where both variables change cannot occur. For example,
$$q((c,r) \rightarrow (\neg c,\neg r)) = 0$$

This gives us the following transition matrix, where the transition is from the state given by the row label to the state given by the column label:

$$\begin{array}{c} \\ (c,r) \\ (c,\neg r) \\ (\neg c,r) \\ (\neg c,\neg r) \end{array} \begin{array}{c} \begin{array}{cccc} (c,r) & (c,\neg r) & (\neg c,r) & (\neg c,\neg r) \end{array} \\ \begin{pmatrix} 17/27 & 5/54 & 5/18 & 0 \\ 11/27 & 22/189 & 0 & 10/21 \\ 2/9 & 0 & 59/153 & 20/51 \\ 0 & 1/42 & 11/102 & 310/357 \end{pmatrix} \end{array}$$

c. $\mathbf{Q}^2$ represents the probability of going from each state to each state in two steps.

d. $\mathbf{Q}^n$ (as $n \rightarrow \infty$) represents the long-term probability of being in each state starting in each state; for ergodic $\mathbf{Q}$ these probabilities are independent of the starting state, so every row of $\mathbf{Q}$ is the same and represents the posterior distribution over states given the evidence.

e. We can produce very large powers of $\mathbf{Q}$ with very few matrix multiplications. For example, we can get $\mathbf{Q}^2$ with one multiplication, $\mathbf{Q}^4$ with two, and $\mathbf{Q}^{2^k}$ with k. Unfortunately, in a network with n Boolean variables, the matrix is of size $2^n \times 2^n$, so each multiplication takes $O(2^{3n})$ operations.

14.19

a. Supposing that q_1 and q_2 are in detailed balance we have:
$$\begin{aligned} &\pi(\mathbf{x})(\alpha q_1(\mathbf{x} \rightarrow \mathbf{x}') + (1-\alpha)q_2(\mathbf{x} \rightarrow \mathbf{x}')) \\ &= \alpha\pi(\mathbf{x})q_1(\mathbf{x} \rightarrow \mathbf{x}') + (1-\alpha)\pi(\mathbf{x})q_2(\mathbf{x} \rightarrow \mathbf{x}') \\ &= \alpha\pi(\mathbf{x})q_1(\mathbf{x}' \rightarrow \mathbf{x}) + (1-\alpha)\pi(\mathbf{x})q_2(\mathbf{x}' \rightarrow \mathbf{x}) \\ &= \pi(\mathbf{x})(\alpha q_1(\mathbf{x}' \rightarrow \mathbf{x}) + (1-\alpha)q_2(\mathbf{x}' \rightarrow \mathbf{x})) \end{aligned}$$

b. The sequential composition is defined by
$$(q_1 \circ q_2)(\mathbf{x} \rightarrow \mathbf{x}') = \sum_{\mathbf{x}''} q_1(\mathbf{x} \rightarrow \mathbf{x}'')q_2(\mathbf{x}'' \rightarrow \mathbf{x}').$$
If q_1 and q_2 both have π as their stationary distribution, then:
$$\begin{aligned} \sum_{\mathbf{x}} \pi(\mathbf{x})(q_1 \circ q_2)(\mathbf{x} \rightarrow \mathbf{x}') &= \sum_{\mathbf{x}} \pi(\mathbf{x}) \sum_{\mathbf{x}''} q_1(\mathbf{x} \rightarrow \mathbf{x}'')q_2(\mathbf{x}'' \rightarrow \mathbf{x}') \\ &= \sum_{\mathbf{x}''} q_2(\mathbf{x}'' \rightarrow \mathbf{x}') \sum_{\mathbf{x}} \pi(\mathbf{x})q_1(\mathbf{x} \rightarrow \mathbf{x}'') \end{aligned}$$

$$\begin{aligned} &= \sum_{\mathbf{x}''} q_2(\mathbf{x}'' \to \mathbf{x}')\pi(\mathbf{x}'') \\ &= \pi(\mathbf{x}') \end{aligned}$$

14.20

a. Because a Gibbs transition step is in detailed balance with π, we have that the acceptance probability is one:

$$\begin{aligned} \alpha(\mathbf{x}' \,|\, \mathbf{x}) &= \min\left(1, \frac{\pi(\mathbf{x}')q(\mathbf{x} \,|\, \mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{x}' \,|\, \mathbf{x})}\right) \\ &= 1 \end{aligned}$$

since by definition of detailed balance we have

$$\pi(\mathbf{x}')q(\mathbf{x} \,|\, \mathbf{x}') = \pi(\mathbf{x})q(\mathbf{x}' \,|\, \mathbf{x})\,.$$

b. Two prove this in two stages. For $\mathbf{x} \neq \mathbf{x}'$ the transition probability distribution is $q(x' \,|\, x)\alpha(\mathbf{x}' \,|\, \mathbf{x})$ and we have:

$$\begin{aligned} \pi(\mathbf{x})q(x' \,|\, x)\alpha(\mathbf{x}' \,|\, \mathbf{x}) &= \pi(\mathbf{x})q(x' \,|\, x)\min\left(1, \frac{\pi(\mathbf{x}')q(\mathbf{x} \,|\, \mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{x}' \,|\, \mathbf{x})}\right) \\ &= \min\left(\pi(\mathbf{x})q(x' \,|\, x), \pi(\mathbf{x}')q(\mathbf{x} \,|\, \mathbf{x}')\right) \\ &= \pi(\mathbf{x}')q(x \,|\, x')\min\left(\frac{\pi(\mathbf{x})q(\mathbf{x}' \,|\, \mathbf{x})}{\pi(\mathbf{x}')q(\mathbf{x} \,|\, \mathbf{x}')}, 1\right) \\ \pi(\mathbf{x}')q(x \,|\, x')\alpha(\mathbf{x} \,|\, \mathbf{x}') \end{aligned}$$

For $\mathbf{x} = \mathbf{x}'$ the transition probability is some $q'(x \,|\, x)$ which always satisfies the equation for detailed balance:

$$\pi(\mathbf{x})q'(x \,|\, x) = \pi(\mathbf{x})q'(x \,|\, x).$$

14.21

a. The classes are $Team$, with instances A, B, and C, and $Match$, with instances AB, BC, and CA. Each team has a quality Q and each match has a $Team_1$ and $Team_2$ and an $Outcome$. The team names for each match are of course fixed in advance. The prior over quality could be uniform and the probability of a win for team 1 should increase with $Q(Team_1) - Q(Team_2)$.

b. The random variables are $A.Q$, $B.Q$, $C.Q$, $AB.Outcome$, $BC.Outcome$, and $CA.Outcome$. The network is shown in Figure S14.3.

c. The exact result will depend on the probabilities used in the model. With any prior on quality that is the same across all teams, we expect that the posterior over $BC.Outcome$ will show that C is more likely to win than B.

d. The inference cost in such a model will be $O(2^n)$ because all the team qualities become coupled.

e. MCMC appears to do well on this problem, provided the probabilities are not too skewed. Our results show scaling behavior that is roughly linear in the number of teams, although we did not investigate very large n.

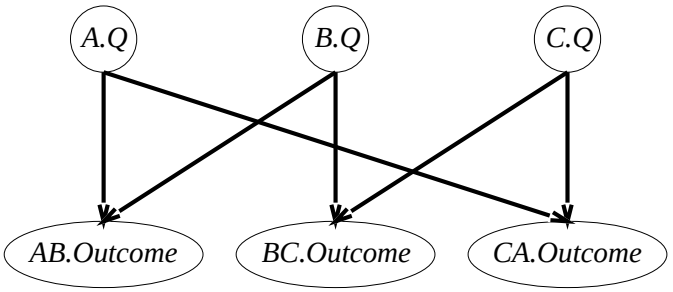


Figure S14.3 Bayes net showing the dependency structure for the team quality and game outcome variables in the soccer model.

Solutions for Chapter 15
Probabilistic Reasoning over Time

15.1 For each variable U_t that appears as a parent of a variable X_{t+2}, define an auxiliary variable U^{old}_{t+1}, such that U_t is parent of U^{old}_{t+1} and U^{old}_{t+1} is a parent of X_{t+2}. This gives us a first-order Markov model. To ensure that the joint distribution over the original variables is unchanged, we keep the CPT for X_{t+2} is unchanged except for the new parent name, and we require that $\mathbf{P}(U^{old}_{t+1}|U_t)$ is an identity mapping, i.e., the child has the same value as the parent with probability 1. Since the parameters in this model are fixed and known, there is no effective increase in the number of free parameters in the model.

15.2

a. For all t, we have the filtering formula

$$\mathbf{P}(R_t|u_{1:t}) = \alpha\mathbf{P}(u_t|R_t)\sum_{R_{t-1}}\mathbf{P}(R_t|R_{t-1})P(R_{t-1}|u_{1:t-1})\ .$$

At the fixed point, we additionally expect that $\mathbf{P}(R_t|u_{1:t}) = \mathbf{P}(R_{t-1}|u_{1:t-1})$. Let the fixed-point probabilities be $\langle\rho,\ 1-\rho\rangle$. This provides us with a system of equations:

$$\begin{aligned}\langle\rho,\ 1-\rho\rangle &= \alpha\langle 0.9,\ 0.2\rangle\langle 0.7,\ 0.3\rangle\rho + \langle 0.3,\ 0.7\rangle(1-\rho)\\ &= \alpha\langle 0.9,\ 0.2\rangle(\langle 0.4\rho,\ -0.4\rho\rangle + \langle 0.3,\ 0.7\rangle)\\ &= \frac{1}{0.9(0.4\rho+0.3)+0.2(-0.4\rho+0.7)}\langle 0.9,\ 0.2\rangle(\langle 0.4\rho,\ -0.4\rho\rangle + \langle 0.3,\ 0.7\rangle)\end{aligned}$$

Solving this system, we find that $\rho \approx 0.8933$.

b. The probability converges to $\langle 0.5, 0.5\rangle$ as illustrated in Figure S15.1. This convergence makes sense if we consider a fixed-point equation for $\mathbf{P}(R_{2+k}|U_1,U_2)$:

$$\begin{aligned}\mathbf{P}(R_{2+k}|U_1,U_2) &= \langle 0.7,0.3\rangle P(r_{2+k-1}|U_1,U_2) + \langle 0.3,0.7\rangle P(\neg r_{2+k-1}|U_1,U_2)\\ \mathbf{P}(r_{2+k}|U_1,U_2) &= 0.7P(r_{2+k-1}|U_1,U_2) + 0.3(1-P(r_{2+k-1}|U_1,U_2))\\ &= 0.4P(r_{2+k-1}|U_1,U_2) + 0.3\end{aligned}$$

That is, $P(r_{2+k}|U_1,U_2) = 0.5$.

Notice that the fixed point does not depend on the initial evidence.

15.3 This exercise develops the Island algorithm for smoothing in DBNs (Binder *et al.*, 1997).

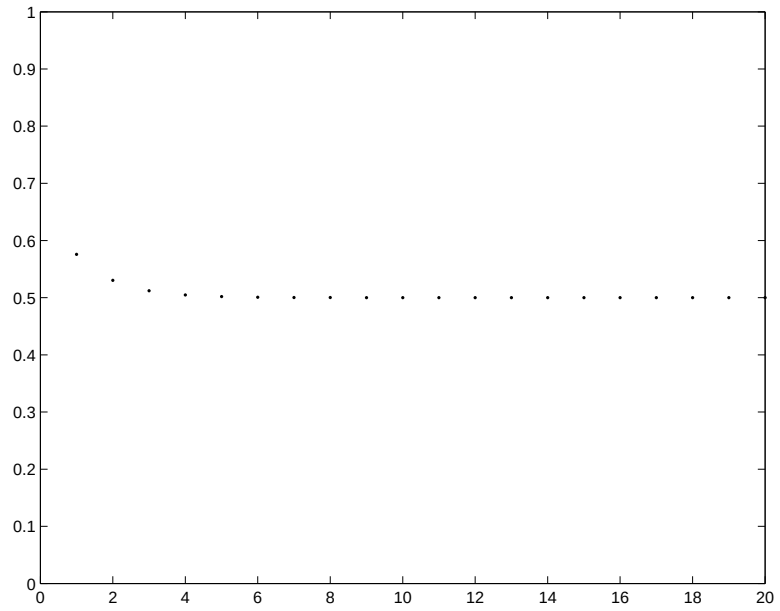


Figure S15.1 A graph of the probability of rain as a function of time, forecast into the future.

a. The chapter shows that $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ can be computed as

$$\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) = \alpha\,\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:k})\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) = \alpha\,\mathbf{f}_{1:k}\mathbf{b}_{k+1:t}$$

The forward recursion (Equation 15.3) shows that $\mathbf{f}_{1:k}$ can be computed from $\mathbf{f}_{1:k-1}$ and $\mathbf{e}_k$, which can in turn be computed from $\mathbf{f}_{1:k-2}$ and $\mathbf{e}_{k-1}$, and so on down to $\mathbf{f}_{1:0}$ and $\mathbf{e}_1$. Hence, $\mathbf{f}_{1:k}$ can be computed from $\mathbf{f}_{1:0}$ and $\mathbf{e}_{1:k}$. The backward recursion (Equation 15.7) shows that $\mathbf{b}_{k+1:t}$ can be computed from $\mathbf{b}_{k+2:t}$ and $\mathbf{e}_{k+1}$, which in turn can be computed from $\mathbf{b}_{k+3:t}$ and $\mathbf{e}_{k+2}$, and so on up to $\mathbf{b}_{h+1:t}$ and $\mathbf{e}_h$. Hence, $\mathbf{b}_{k+1:t}$ can be computed from $\mathbf{b}_{h+1:t}$ and $\mathbf{e}_{k+1:h}$. Combining these two, we find that $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ can be computed from $\mathbf{f}_{1:0}$, $\mathbf{b}_{h+1:t}$, and $\mathbf{e}_{1:h}$.

b. The reasoning for the second half is essentially identical: for k between h and t, $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ can be computed from $\mathbf{f}_{1:h}$, $\mathbf{b}_{t+1:t}$, and $\mathbf{e}_{h+1:t}$.

c. The algorithm takes 3 arguments: an evidence sequence, an initial forward message, and a final backward message. The forward message is propagated to the halfway point and the backward message is propagated backward. The algorithm then calls itself recursively on the two halves of the evidence sequence with the appropriate forward and backward messages. The base case is a sequence of length 1 or 2.

d. At each level of the recursion the algorithm traverses the entire sequence, doing $O(t)$ work. There are $O(\log_2 t)$ levels, so the total time is $O(t \log_2 t)$. The algorithm does a depth-first recursion, so the total space is proportional to the depth of the stack, i.e., $O(\log_2 t)$. With n islands, the recursion depth is $O(\log_n t)$, so the total time is $O(t \log_n t)$ but the space is $O(n \log_n t)$.

15.4 This is a very good exercise for deepening intuitions about temporal probabilistic reasoning. First, notice that the impossibility of the sequence of most likely states cannot come from an impossible observation because the smoothed probability at each time step includes the evidence likelihood at that time step as a factor. Hence, the impossibility of a sequence must arise from an impossible transition. Now consider such a transition from $X_k = i$ to $X_{k+1} = j$ for some i, j, k. For $X_{k+1} = j$ to be the most likely state at time $k+1$, even though it cannot be reached from the most likely state at time k, we can simply have an n-state system where, say, the smoothed probability of $X_k = i$ is $(1 + (n-1)\epsilon)/n$ and the remaining states have probability $(1-\epsilon)/n$. The remaining states all transition deterministically to $X_{k+1} = j$. From here, it is a simple matter to work out a specific model that behaves as desired.

15.5 The propagation of the $\boldsymbol{\ell}$ message is identical to that for filtering:

$$\boldsymbol{\ell}_{1:t+1} = \alpha\, \mathbf{O}_{t+1}\mathbf{T}^\top \boldsymbol{\ell}_{1:t}$$

Since $\boldsymbol{\ell}$ is a column vector, each entry ℓ_i of which gives $P(X_t = i, \mathbf{e}_{1:t})$, the likelihood is obtained simply by summing the entries:

$$L_{1:t} = P(\mathbf{e}_{1:t}) = \sum_i \ell_i \ .$$

15.6 Let ℓ be the single possible location under deterministic sensing. Certainly, as $\epsilon \to 0$, we expect intuitively that $P(X_t = \ell \,|\, e_{1:t}) \to 1$. If we assume that all reachable locations are equally likely to be reached under the uniform motion model, then the claim that ℓ is the most likely location under noisy sensing follows immediately: any other location must entail at least one sensor discrepancy—and hence a probability penalty factor of ϵ—on every path reaching it in $t-1$ steps, otherwise it would be logically possible in the deterministic setting. The assumption is incorrect, however: if the neighborhood graph has outdegree k, the probability of reaching any two locations could differ by a factor of $O(k^t)$. If we set ϵ smaller than this, the claim still holds. But for any fixed ϵ, there are neighborhood graphs and observation sequences such that the claim may be false for sufficiently large t. Essentially, if $t-1$ steps of random movement are much more likely to reach m than ℓ—e.g., if ℓ is at the end of a long tunnel of length exactly $t-1$—then that can outweigh the cost of a sensor error or two. Notice that this argument requires an environment of unbounded size; for any bounded environment, we can bound the reachability ratio and set ϵ accordingly.

15.7 This exercise is an important one: it makes very clear the difference between the actual environment and the agent's model of it. To generate the equired data, the student will need to run a world simulator (movment and percepts) using the true model (northwest prior, southeat movement tendency), while running he agent's state estimator using the assumed model (uniform prior, uniformly random movement). The student will also begin to appreciate the inexpressiveness of HMMs after constructing the 64×64 transition matrix from the more natural representation in terms of coordinates.

Perhaps surprisingly, the data for expected localization error (expected Manhattan distance between true location and the posterior state estimate) show that having an incorrect model is not too problematic. A "southeast" bias of b was implemented by multiplying the probability of any south or east move by b and then renormalizing the distribution before

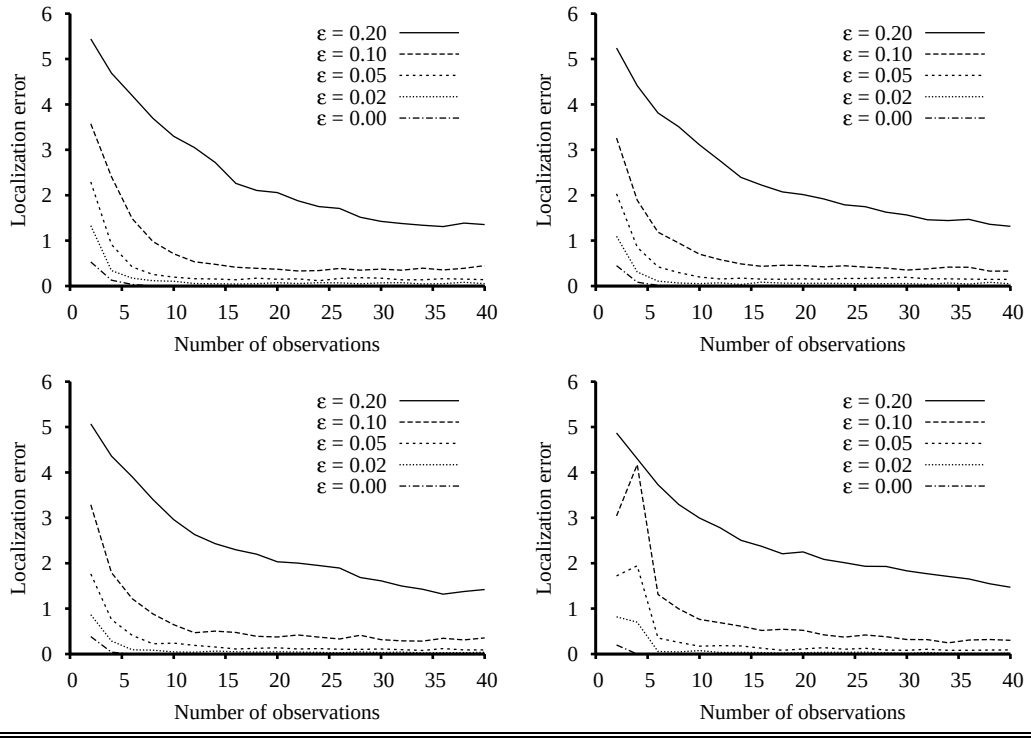


Figure S15.2 Graphs showiing the expected localization error as a function of time, for bias values of 1.0 (unbiased), 2.0, 5.0, 10.0.

sampling. Graphs for four different values of the bias are shown in Figure S15.2. The results suggest that the sensor data sequence overwhelms any error introduced by the incorrect motion model.

15.8 The code for this exercise is very similar to that for Exercise 15.6. The main difference is the state space: instead of 64 locations, the state space has 256 location–heading pairs, and the transition matrix is 256×256—starting to be a little painful when running hundreds of trials. We also need to add a "bump" bit to the percept vector, and we assume this is perfectly observed (else the robot could not always decide to pick a new heading). Generally we expect localization to be more accurate, since the sensor sequence need only disambiguate among a small number of possible headings rather than an exponentially growing set of random-walk paths. Also the exact bump sensor will eliminate many possible states completely.

15.9 The code for this exercise is very similar to that for Exercise 15.7. The state is again a location/heading pair with a 256×256 transition matrix. The observation model is different: instead of a 4-bit percept (16 possible percepts), the percept is a location (or null), for $n \times m+1$ possible percepts. Generally speaking, tracking works well near the walls because any bump (or even the lack thereof) helps to pin down the location. Away from the walls, the location uncertainty will be slightly worse but still generally accurate because the location errors are

independent and unbiased and the policy is fairly predictable. It is reasonable to expect students to provide snapshots or movies of true location and posterior estimate, particularly if they are given suitable graphics infrastructure to make this easy.

15.10

a. Looking at the fragment of the model containing just S_0, $\mathbf{X}_0$, and $\mathbf{X}_1$, we have

$$\mathbf{P}(\mathbf{X}_1) = \sum_{s_0=1}^{k} P(s_0) \int_{\mathbf{x}_0} P(\mathbf{x}_0)\mathbf{P}(X_1|\mathbf{x}_0, s_0)$$

From the properties of the Kalman filter, we know that the integral gives a Gaussian for each different value of s_0. Hence, the prediction distribution is a mixture of k Gaussians, each weighted by $P(s_0)$.

b. The update equation for the switching Kalman filter is

$$\begin{aligned}
&\mathbf{P}(\mathbf{X}_{t+1}, S_{t+1}|\mathbf{e}_{1:t+1}) \\
&= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, S_{t+1}) \sum_{s_t=1}^{k} \int_{\mathbf{x}_t} \mathbf{P}(\mathbf{x}_t, s_t|\mathbf{e}_{1:t})\mathbf{P}(\mathbf{X}_{t+1}, S_{t+1}|\mathbf{x}_t, s_t) \\
&= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{s_t=1}^{k} P(s_t|\mathbf{e}_{1:t})\mathbf{P}(S_{t+1}|s_t) \int_{\mathbf{x}_t} \mathbf{P}(\mathbf{x}_t|\mathbf{e}_{1:t})\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, s_t)
\end{aligned}$$

We are given that $\mathbf{P}(\mathbf{x}_t|\mathbf{e}_{1:t})$ is a mixture of m Gaussians. Each Gaussian is subject to k different linear–Gaussian projections and then updated by a linear-Gaussian observation, so we obtain a sum of km Gaussians. Thus, after t steps we have k^t Gaussians.

c. Each weight represents the probability of one of the k^t sequences of values for the switching variable.

15.11 This is a simple exercise in algebra. We have

$$\begin{aligned}
P(x_1|z_1) &= \alpha\, e^{-\frac{1}{2}\left(\frac{(z_1-x_1)^2}{\sigma_z^2}\right)} e^{-\frac{1}{2}\left(\frac{(x_1-\mu_0)^2}{\sigma_0^2+\sigma_x^2}\right)} \\
&= \alpha\, e^{-\frac{1}{2}\left(\frac{(\sigma_0^2+\sigma_x^2)(z_1-x_1)^2+\sigma_z^2(x_1-\mu_0)^2}{\sigma_z^2(\sigma_0^2+\sigma_x^2)}\right)} \\
&= \alpha\, e^{-\frac{1}{2}\left(\frac{(\sigma_0^2+\sigma_x^2)(z_1^2-2z_1x_1+x_1^2)+\sigma_z^2(x_1^2-2\mu_0x_1+\mu_0^2)}{\sigma_z^2(\sigma_0^2+\sigma_x^2)}\right)} \\
&= \alpha\, e^{-\frac{1}{2}\left(\frac{(\sigma_0^2+\sigma_x^2+\sigma_z^2)x_1^2-2((\sigma_0^2+\sigma_x^2)z_1+\sigma_z^2\mu_0)x_1+c}{\sigma_z^2(\sigma_0^2+\sigma_x^2)}\right)} \\
&= \alpha'\, e^{-\frac{1}{2}\left(\frac{(x_1-\frac{(\sigma_0^2+\sigma_x^2)z_1+\sigma_z^2\mu_0}{\sigma_0^2+\sigma_x^2+\sigma_z^2})^2}{(\sigma_0^2+\sigma_x^2)\sigma_z^2/(\sigma_0^2+\sigma_x^2+\sigma_z^2)}\right)}.
\end{aligned}$$

15.12

a. See Figure S15.3.

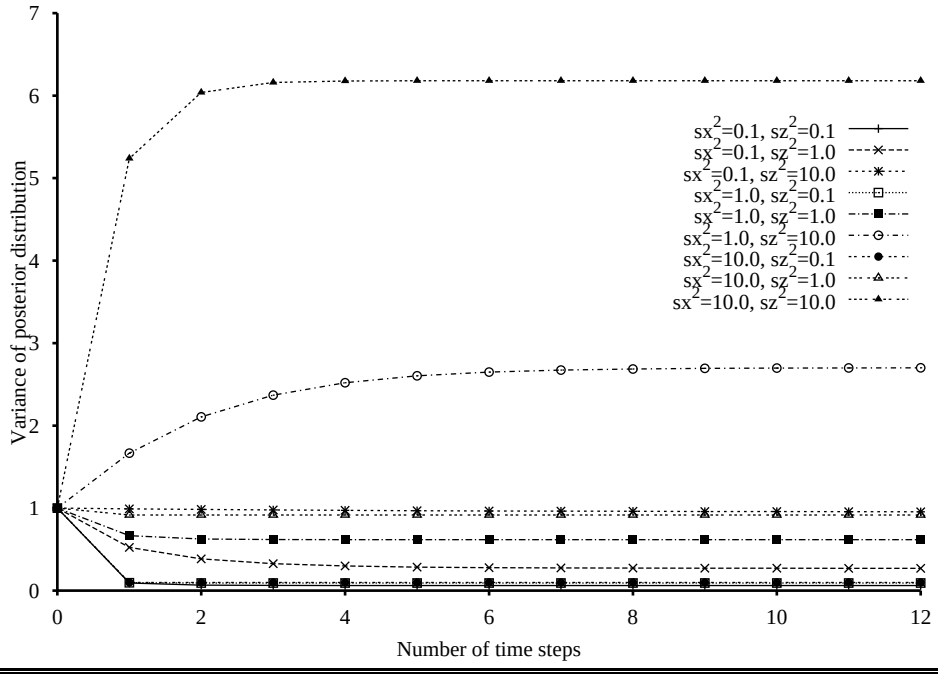


Figure S15.3 Graph for Ex. 15.7, showing the posterior variance σ_t^2 as a function of t for various values of σ_x^2 and σ_z^2.

b. We can find a fixed point by solving

$$\sigma^2 = \frac{(\sigma^2 + \sigma_x^2)\sigma_z^2}{\sigma^2 + \sigma_x^2 + \sigma_z^2}$$

for σ^2. Using the quadratic formula and requiring $\sigma^2 \geq 0$, we obtain

$$\sigma^2 = \frac{-\sigma_x^2 + \sqrt{\sigma_x^4 + 4\sigma_x^2\sigma_z^2}}{\sigma_z^2}$$

We omit the proof of convergence, which, presumably, can be done by showing that the update is a contraction (i.e., after updating, two different starting points for σ_t become closer).

c. As $\sigma_x^2 \to 0$, we see that the fixed point $\sigma^2 \to 0$ also. This is because $\sigma_x^2 = 0$ implies a deterministic path for the object. Each observation supplies more information about this path, until its parameters are known completely.

As $\sigma_z^2 \to 0$, the variance update gives $\sigma^{t+1} \to 0$ immediately. That is, if we have an exact observation of the object's state, then the posterior is a delta function about that observed value regardless of the transition variance.

15.13 The DBN has three variables: S_t, whether the student gets enough sleep; R_t, whether they have red eyes in class; C_t, whether the student sleeps in class. S_t is a parent of S_{t+1}, R_t,

and C_t. The CPTs are given by

$$\begin{aligned} P(s_0) &= 0.7 \\ P(s_{t+1}|s_t) &= 0.8 \\ P(s_{t+1}|\neg s_t) &= 0.3 \\ P(r_t|s_t) &= 0.2 \\ P(r_t|\neg s_t) &= 0.7 \\ P(c_t|s_t) &= 0.1 \\ P(c_t|\neg s_t) &= 0.3 \end{aligned}$$

To reformulate as an HMM with a single observation node, simply combine the 2-valued variables "having red eyes" and "sleeping in class" into a single 4-valued variable, multiplying together the emission probabilities. (Probability tables omitted.)

15.14

a. We apply the forward algorithm to compute these probabilities.

$$\begin{aligned} P(S_0) &= \langle 0.7, 0.3\rangle \\ P(S_1) &= \sum_{s_0} P(S_1|s_0)P(s_0) \\ &= (\langle 0.8, 0.2\rangle 0.7 + \langle 0.3, 0.7\rangle 0.3) \\ &= \langle 0.65, 0.35\rangle \\ P(S_1|e_1) &= \alpha\, P(e_1|S_1)P(S_1) \\ &= \alpha\, \langle 0.8 \times 0.9, 0.3 \times 0.7\rangle \langle 0.65, 0.35\rangle \\ &= \alpha\, \langle 0.72, 0.21\rangle \langle 0.65, 0.35\rangle \\ &= \langle 0.8643, 0.1357\rangle \\ P(S_2|e_1) &= \sum_{s_1} P(S_2|s_1)P(s_1|e_1) \\ &= \langle 0.7321, 0.2679\rangle \\ P(S_2|e_{1:2}) &= \alpha\, P(e_2|S_2)P(S_2|e_1) \\ &= \langle 0.5010, 0.4990\rangle \\ P(S_3|e_{1:2}) &= \sum_{s_2} P(S_3|s_2)P(s_2|e_{1:2}) \\ &= \langle 0.5505, 0.4495\rangle \\ P(S_3|e_{1:3}) &= \alpha\, P(e_3|S_3)P(S_3|e_{1:2}) \\ &= \langle 0.1045, 0.8955\rangle \end{aligned}$$

Similar to many students during the course of the school term, the student observed here seems to have a higher likelihood of being sleep deprived as time goes on!

b. First we compute the backwards messages:

$$P(e_3|S_3) = \langle 0.2 \times 0.1, 0.7 \times 0.3\rangle$$

$$
\begin{aligned}
&= \langle 0.02, 0.21 \rangle \\
P(e_3|S_2) &= \sum_{s_3} P(e_3|s_3)P(|s_3)P(s_3|S_2) \\
&= \langle 0.02 \times 0.8 + 0.21 \times 0.2, 0.02 \times 0.3 + 0.21 \times 0.7 \rangle \\
&= \langle 0.0588, 0.153 \rangle \\
P(e_{2:3}|S_1) &= \sum_{s_2} P(e_2|s_2)P(e_3|s_2)P(s_2|S_1) \\
&= \langle 0.0233, 0.0556 \rangle
\end{aligned}
$$

Then we combine these with the forwards messages computed previously and normalize:

$$
\begin{aligned}
P(S_1|e_{1:3}) &= \alpha\, P(S_1|e_1)P(e_{2:3}|S_1) \\
&= \langle 0.7277, 0.2723 \rangle \\
P(S_2|e_{1:3}) &= \alpha\, P(S_2|e_{1:2})P(e_3|S_1) \\
&= \langle 0.2757, 0.7243 \rangle \\
P(S_3|e_{1:3}) &= \langle 0.1045, 0.8955 \rangle
\end{aligned}
$$

c. The smoothed analysis places the time the student started sleeping poorly one step earlier than than filtered analysis, integrating future observations indicating lack of sleep at the last step.

15.15 The probability reaches a fixed point because there is always some chance of spontaneously starting to sleep well again, and students who sleep well sometimes have red eyes and sleep in class. Even if we knew for sure that the student didn't sleep well on day t, and that they slept in class with red eyes on day $t+1$, there would still be a chance that they slept well on day $t+1$.

Numerically one can repeatedly apply the forward equations to find equilibrium probabilities of $\langle 0.0432, 0.9568 \rangle$.

Analytically, we are trying to find the vector $(p_0, p_1)^T$ which is the fixed point to the forward equation, which one can pose in matrix form as

$$
(p_0, p_1)^T = \alpha \begin{pmatrix} 0.016 & 0.006 \\ 0.042 & 0.147 \end{pmatrix} (p_0, p_1)^T
$$

where α is a normalization constant. That is, $(p_0, p_1)^T$ is an eigenvector of the given matrix. Computing, we find that the only positive eigenvalue is 0.1487, which has eigenvector (normalized to sum to one) $(0.0432, 0.9568)^T$, just as we numerically computed.

15.16

a. The curve of interest is the one for $E(Battery_t|\ldots 5555000000\ldots)$. In the absence of any useful sensor information from the battery meter, the posterior distribution for the battery level is the same as the projection without evidence. The transition model for the battery includes a small probability for downward transitions in the battery level at each time step, but zero probability for upward transitions (there are no recharging

actions in the model). Thus, the stationary distribution towards which the battery level tends has value 0 with probability 1. The curve for $E(Battery_t|\ldots 5555000000\ldots)$ will asymptote to 0.

b. See Figure S15.4. The CPT for $BMeter_1$ has a probability of transient failure (i.e., reporting 0) that increases with temperature.

c. The agent can obviously calculate the posterior distribution over $Temp_t$ by filtering the observation sequence in the usual way. This posterior can be informative if the effect of temperature on transient failure is non-negligible and transient failures occur more frequently than do major changes in temperature. Essentially, the temperature is estimated from the frequency of "blips" in the sequence of battery meter readings.

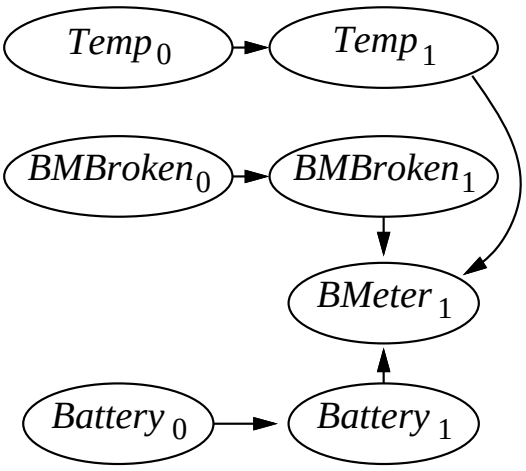


Figure S15.4 Modification of Figure 15.13(a) to include the effect of external temperature on the battery meter.

15.17 The process works exactly as on page 507. We start with the full expression:

$$\mathbf{P}(R_3|u_1,u_2,u_3) = \alpha \sum_{r_1}\sum_{r_2} P(r_1)P(u_1|r_1)P(r_2|r_1)P(u_2|r_2)\mathbf{P}(R_3|r_2)\mathbf{P}(u_3|R_3)$$

Whichever order we push in the summations, the variable elimination process never creates factors containing more than two variables, which is the same size as the CPTs in the original network. In fact, given an HMM sequence of arbitrary length, we can eliminate the state variables in any order.

Solutions for Chapter 16
Making Simple Decisions

16.1 It is interesting to create a histogram of accuracy on this task for the students in the class. It is also interesting to record how many times each student comes within, say, 10% of the right answer. Then you get a profile of each student: this one is an accurate guesser but overly cautious about bounds, etc.

16.2

Pat is more likely to have a better car than Chris because she has more information with which to choose. She is more likely to be disappointed, however, if she takes the expected utility of the best car at face value. Using the results of exercise 16.11, we can compute the expected disappointment to be about 1.54 times the standard deviation by numerical integration.

16.3

a. The probability that the first heads appears on the nth toss is 2^{-n}, so

$$EMV(L) = \sum_{n=1}^{\infty} 2^{-n} \cdot 2^n = \sum_{n=1}^{\infty} 1 = \infty$$

b. Typical answers range between \$4 and \$100.

c. Assume initial wealth (after paying c to play the game) of $\$(k-c)$; then

$$U(L) = \sum_{n=1}^{\infty} 2^{-n} \cdot (a \log_2(k - c + 2^n) + b)$$

Assume $k - c = \$0$ for simplicity. Then

$$\begin{aligned} U(L) &= \sum_{n=1}^{\infty} 2^{-n} \cdot (a \log_2(2^n) + b) \\ &= \sum_{n=1}^{\infty} 2^{-n} \cdot an + b \\ &= 2a + b \end{aligned}$$

d. The maximum amount c is given by the solution of

$$a \log_2 k + b = \sum_{n=1}^{\infty} 2^{-n} \cdot (a \log_2(k - c + 2^n) + b)$$

For our simple case, we have

$$a \log_2 c + b = 2a + b$$

or $c = \$4$.

16.4 The program itself is pretty trivial. But note that there are some studies showing you get better answers if you ask subjects to move a slider to indicate a proportion, rather than asking for a probability number. So having a graphical user interface is an advantage. The main point of the exercise is to examine the data, expose inconsistent behavior on the part of the subjects, and see how people vary in their choices.

16.5

a. Networks (ii) and (iii) can represent this network but not (i).

(ii) is fully connected, so it can represent any joint distribution.

(iii) follows the generative story given in the problem: the flavor is determined (presumably) by which machine the candy is made by, then the shape is randomly cut, and the wrapper randomly chosen, the latter choice independently of the former.

(i) cannot represent this, as this network implies that the wrapper color and shape are marginally independent, which is not so: a round candy is likely to be strawberry, which is in turn likely to be wrapped in red, whilst conversely a square candy is likely to be anchovy which is likely to be wrapped in brown.

b. Unlike (ii), (iii) has no cycles which we have seen simplifies inference. Its edges also follow the, so probabilities will be easier to elicit. Indeed, the problem statement has already given them.

c. Yes, because Wrapper and Shape are d-separated.

d. Once we know the Flavor we know the probability its wrapper will be red or brown. So we marginalize Flavor out:

$$\begin{aligned}
\mathbf{P}(Wrapper = red) &= \sum_f \mathbf{P}(Wrapper = red, Flavor = f) \\
&= \sum_f \mathbf{P}(Flavor = f)\mathbf{P}(Wrapper = red | Flavor = f) \\
&= 0.7 \times 0.8 + 0.3 \times 0.1 \\
&= 0.59
\end{aligned}$$

e. We apply Bayes theorem, by first computing the joint probabilities

$$\begin{aligned}
&\mathbf{P}(Flavor = strawberry, Shape = round, Wrapper = red) \\
&= \mathbf{P}(Flavor = strawberry) \times \mathbf{P}(Shape = round | Flavor = strawberry) \\
&\quad \times \mathbf{P}(Wrapper = red | Flavor = strawberry) \\
&= 0.7 \times 0.8 \times 0.8 \\
&= 0.448 \\
&\mathbf{P}(Flavor = anchovy, Shape = round, Wrapper = red) \\
&= \mathbf{P}(Flavor = anchovy) \times \mathbf{P}(Shape = round | Flavor = anchovy)
\end{aligned}$$

$$\begin{aligned} & \times \mathbf{P}(Wrapper = red | Flavor = anchovy) \\ = \; & 0.3 \times 0.1 \times 0.1 \\ = \; & 0.003 \end{aligned}$$

Normalizing these probabilities yields that it is strawberry with probability $0.448/(0.448+0.003) \approx 0.9933$.

f. Its value is the probability that you have a strawberry upon unwrapping times the value of a strawberry, plus the probability that you have a anchovy upon unwrapping times the value of an anchovy or

$$0.7s + 0.3a \; .$$

g. The value is the same, by the axiom of decomposability.

16.6

First observe that $C \sim [0.25, A; 0.75, \$0]$ and $D \sim [0.25, B; 0.75\$0]$. This follows from the axiom of decomposability. But by substitutability this means that the preference ordering between the lotteries A and B must be the same as that between C and D.

16.7

As mentioned in the text, agents whose preferences violate expected utility theory demonstrate irrational behavior, that is they can be made either to accept a bet that is a guaranteed loss for them (the case of violating transitivity is given in the text), or reject a bet that is a guaranteed win for them. This indicates a problem for the agent.

16.8 The expected monetary value of the lottery L is

$$EMV(L) = \frac{1}{50} \times \$10 + \frac{1}{2000000} \times \$1000000 = \$0.70$$

Although $\$0.70 < \$1$, it is not *necessarily* irrational to buy the ticket. First we will consider just the utilities of the monetary outcomes, ignoring the utility of actually playing the lottery game. Using $U(S_{k+n})$ to represent the utility to the agent of having n dollars more than the current state, and assuming that utility is linear for small values of money (i.e., $U(S_{k+n}) \approx n(U(S_{k+1}) - U(S_k))$ for $-10 \leq n \leq 10$), the utility of the lottery is:

$$\begin{aligned} U(L) \; &= \; \frac{1}{50} U(S_{k+10}) + \frac{1}{2,000,000} U(S_{k+1,000,000}) \\ &\approx \; \frac{1}{5} U(S_{k+1}) + \frac{1}{2,000,000} U(S_{k+1,000,000}) \end{aligned}$$

This is more than $U(S_{k+1})$ when $U(S_{k+1,000,000}) > 1,600,000 U(\$1)$. Thus, for a purchase to be rational (when only money is considered), the agent must be quite risk-seeking. This would be unusual for low-income individuals, for whom the price of a ticket is non-trivial. It is possible that some buyers do not internalize the magnitude of the very low probability of winning—to imagine an event is to assign it a "non-trivial" probability, in effect. Apparently, these buyers are better at internalizing the large magnitude of the prize. Such buyers are clearly acting irrationally.

Some people may feel their current situation is intolerable, that is, $U(S_k) \approx U(S_{k\pm1}) \approx u_\perp$. Therefore the situation of having one dollar more or less would be equally intolerable, and it would be rational to gamble on a high payoff, even if one that has low probability.

Gamblers also derive pleasure from the excitement of the lottery and the temporary possession of at least a non-zero chance of wealth. So we should add to the utility of playing the lottery the term t to represent the thrill of participation. Seen this way, the lottery is just another form of entertainment, and buying a lottery ticket is no more irrational than buying a movie ticket. Either way, you pay your money, you get a small thrill t, and (most likely) you walk away empty-handed. (Note that it could be argued that doing this kind of decision-theoretic computation decreases the value of t. It is not clear if this is a good thing or a bad thing.)

16.9 This is an interesting exercise to do in class. Choose $M_1 = \$100$, M_2 = \$100, \$1000, \$10000, \$1000000. Ask for a show of hands of those preferring the lottery at different values of p. Students will almost always display risk aversion, but there may be a wide spread in its onset. A curve can be plotted for the class by finding the smallest p yielding a majority vote for the lottery.

16.10 The protocol would be to ask a series of questions of the form "which would you prefer" involving a monetary gain (or loss) versus an increase (or decrease) in a risk of death. For example, "would you pay \$100 for a helmet that would eliminate completely the one-in-a-million chance of death from a bicycle accident."

16.11

First observe that the cumulative distribution function for $\max\{X_1, \ldots, X_k\}$ is $(F(x))^k$ since

$$\begin{aligned} P(\max\{X_1, \ldots, X_k\} \leq x) &= P(X_1 \leq x, \ldots, X_k \leq x) \\ &= P(X_1 \leq x) \ldots P(X_k \leq x) \\ &= F(x)^k \end{aligned}$$

the second to last step by independence. The result follows as the probability density function is the derivative of the cumulative distribution function.

16.12

a. This question originally had a misprint: $U(x) = -e^{x/R}$ instead of $U(x) = -e^{-x/R}$. With the former utility function, the agent would be rather unhappy receiving \$1000000 dollars.

   Getting \$400 for sure has expected utility

   $$-e^{-400/400} = -1/e \approx -0.3679$$

   while the getting \$5000 with probability 0.6 and \$0 otherwise has expected utility

   $$0.6 - e^{-5000/400} + 0.5 - e^{-0/400} = -(0.6e^{-12.5} + 0.5) \approx -0.5000$$

   so one would prefer the sure bet.

b. We want to find R such that

   $$e^{-100/R} = 0.5e^{-500/R} + 0.5$$

Solving this numerically, we find $R = 152$ up to 3sf.

16.13 The information associated with the utility node in Figure 16.6 is an action-value table, and can be constructed simply by averaging out the $Deaths$, $Noise$, and $Cost$ nodes in Figure 16.5. As explained in the text , modifications to aircraft noise levels or to the importance of noise do not result in simple changes to the action-value table. Probably the easiest way to do it is to go back to the original table in Figure 16.5. The exercise therefore illustrates the tradeoffs involved in using compiled representations.

16.14 The answer to this exercise depends on the probability values chosen by the student.

16.15

a. See Figure S16.1.

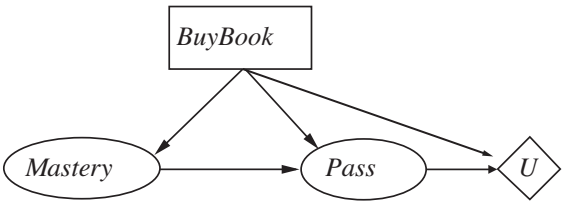


Figure S16.1 A decision network for the book-buying problem.

b. For each of $B = b$ and $B = \neg b$, we compute $P(p|B)$ and thus $P(\neg p|B)$ by marginalizing out M, then use this to compute the expected utility.

$$\begin{aligned} P(p|b) &= \sum_m P(p|b,m)P(m|b) \\ &= 0.9 \times 0.9 + 0.5 \times 0.1 \\ &= 0.86 \\ P(p|\neg b) &= \sum_m P(p|\neg b,m)P(m|\neg b) \\ &= 0.8 \times 0.7 + 0.3 \times 0.3 \\ &= 0.65 \end{aligned}$$

The expected utilities are thus:

$$\begin{aligned} EU[b] &= \sum_p P(p|b)U(p,b) \\ &= 0.86(2000 - 100) + 0.14(-100) \\ &= 1620 \\ EU[\neg b] &= \sum_p P(p|\neg b)U(p,\neg b) \\ &= 0.65 \times 2000 + 0.14 \times 0 \\ &= 1300 \end{aligned}$$

c. Buy the book, Sam.

16.16 This exercise can be solved using an influence diagram package such as IDEAL. The specific values are not especially important. Notice how the tedium of encoding all the entries in the utility table cries out for a system that allows the additive, multiplicative, and other forms sanctioned by MAUT.

One of the key aspects of the fully explicit representation in Figure 16.5 is its amenability to change. By doing this exercise as well as Exercise 16.9, students will augment their appreciation of the flexibility afforded by declarative representations, which can otherwise seem tedious.

a. For this part, one could use symbolic values (high, medium, low) for all the variables and not worry too much about the exact probability values, or one could use actual numerical ranges and try to assess the probabilities based on some knowledge of the domain. Even with three-valued variables, the cost CPT has 54 entries.

b. This part almost certainly should be done using a software package.

c. If each aircraft generates half as much noise, we need to adjust the entries in the $Noise$ CPT.

d. If the noise attribute becomes three times more important, the utility table entries must all be altered. If an appropriate (e.g., additive) representation is available, then one would only need to adjust the appropriate constants to reflect the change.

e. This part should be done using a software package. Some packages may offer VPI calculation already. Alternatively, one can invoke the decision-making package repeatedly to do all the what-if calculations of best actions and their utilities, as required in the VPI formula. Finally, one can write general-purpose VPI code as an add-on to a decision-making package.

16.17 This question is a simple exercise in sequential decision making, and helps in making the transition to Chapter 17. It also emphasizes the point that the value of information is computed by examining the *conditional* plan formed by determining the best action for each possible outcome of the test. It may be useful to introduce "decision trees" (as the term is used in the decision analysis literature) to organize the information in this question. (See Pearl (1988), Chapter 6.) Each part of the question analyzes some aspect of the tree. Incidentally, the question assumes that utility and monetary value coincide, and ignores the transaction costs involved in buying and selling.

a. The decision network is shown in Figure S16.2.

b. The expected net gain in dollars is

$$P(q^+)(2000 - 1500) + P(q^-)(2000 - 2200) = 0.7 \times 500 + 0.3 \times -200 = 290$$

c. The question could probably have been stated better: Bayes' theorem is used to compute $P(q^+|Pass)$, etc., whereas conditionalization is sufficient to compute $P(Pass)$.

$$\begin{aligned} P(Pass) &= P(Pass|q^+)P(q^+) + P(Pass|q^-)P(q^-) \\ &= 0.8 \times 0.7 + 0.35 \times 0.3 = 0.665 \end{aligned}$$

Using Bayes' theorem:

$$\begin{aligned} P(q^+|Pass) &= \frac{P(Pass|q^+)P(q^+)}{P(Pass)} = \frac{0.8 \times 0.7}{0.665} \approx 0.8421 \\ P(q^-|Pass) &\approx 1 - 0.8421 = 0.1579 \\ P(q^+|\neg Pass) &= \frac{P(\neg Pass|q^+)P(q^+)}{P(\neg Pass)} = \frac{0.2 \times 0.7}{0.335} \approx 0.4179 \\ P(q^-|\neg Pass) &\approx 1 - 0.4179 = 0.5821 \end{aligned}$$

d. If the car passes the test, the expected value of buying is

$$\begin{aligned} &P(q^+|Pass)(2000 - 1500) + P(q^-|Pass)(2000 - 2200) \\ &= 0.8421 \times 500 + 0.1579 \times -200 = 378.92 \end{aligned}$$

Thus buying is the best decision given a pass. If the car fails the test, the expected value of buying is

$$\begin{aligned} &P(q^+|\neg Pass)(2000 - 1500) + P(q^-|\neg Pass)(2000 - 2200) \\ &= 0.4179 \times 500 + 0.5821 \times -200 = 92.53 \end{aligned}$$

Buying is again the best decision.

e. Since the action is the same for both outcomes of the test, the test itself is worthless (if it is the only possible test) and the optimal plan is simply to buy the car without the test. (This is a trivial conditional plan.) For the test to be worthwhile, it would need to be more discriminating in order to reduce the probability $P(q^+|\neg Pass)$. The test would also be worthwhile if the market value of the car were less, or if the cost of repairs were more.

An interesting additional exercise is to prove the general proposition that if α is the best action for all the outcomes of a test then it must be the best action in the absence of the test outcome.

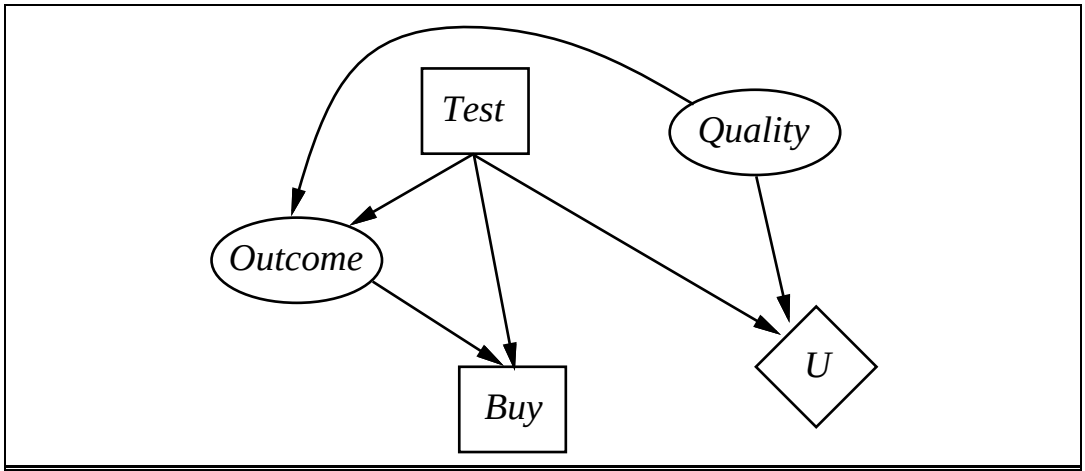


Figure S16.2 A decision network for the car-buying problem.

16.18

a. Intuitively, the value of information is nonnegative because in the worst case one could simply ignore the information and act as if it was not available. A formal proof therefore begins by showing that this policy results in the same expected utility. The formula for the value of information is

$$VPI_E(E_j) = \left(\sum_k P(E_j = e_{jk}|E)EU(\alpha_{e_{jk}}|E, E_j = e_{jk})\right) - EU(\alpha|E)$$

If the agent does α given the information E_j, its expected utility is

$$\sum_k P(E_j = e_{jk}|E)EU(\alpha|E, E_j = e_{jk}) = EU(\alpha|E)$$

where the equality holds because the LHS is just the conditionalization of the RHS with respect to E_j. By definition,

$$EU(\alpha_{e_{jk}}|E, E_j = e_{jk}) \geq EU(\alpha|E, E_j = e_{jk})$$

hence $VPI_E(E_j) \geq 0$.

b. One explanation is that people are aware of their own irrationality and may want to avoid making a decision on the basis of the extra information. Another might be that the value of information is small compared to the value of surprise—for example, many people prefer not to know in advance what their birthday present is going to be.

c. Value of information is not submodular in general. Suppose that A is the empty set and B is the set $Y = 1$; and suppose that the optimal decision remains unchanged unless both $X = 1$ and $Y = 1$ are observed.

Solutions for Chapter 17
Making Complex Decisions

17.1 The best way to calculate this is NOT by thinking of all ways to get to any given square and how likely all those ways are, but to compute the occupancy probabilities at each time step. These are as follows:

		Up	*Up*	*Right*	*Right*	*Right*
(1,1)	1	.1	.02	.026	.0284	.02462
(1,2)		.8	.24	.258	.2178	.18054
(1,3)			.64	.088	.0346	.02524
(2,1)		.1	.09	.034	.0276	.02824
(2,3)				.512	.1728	.06224
(3,1)			.01	.073	.0346	.02627
(3,2)				.001	.0073	.04443
(3,3)					.4097	.17994
(4,1)				.008	.0656	.08672
(4,2)					.0016	.01400
(4,3)						.32776

Projection in an HMM involves multiplying the vector of occupancy probabilities by the transition matrix. Here, the only difference is that there is a different transition matrix for each action.

17.2 If we pick the policy that goes *Right* in all the optional states, and construct the corresponding transition matrix $\mathbf{T}$, we find that the equilibrium distribution—the solution to $\mathbf{Tx} = \mathbf{x}$—has occupancy probabilities of 1/12 for (2,3), (3,1), (3,2), (3,3) and 2/3 for (4,1). These can be found most simply by computing $\mathbf{T}^n\mathbf{x}$ for any initial occupancy vector $\mathbf{x}$, for n large enough to achieve convergence.

17.3 Stationarity requires the agent to have identical preferences between the sequence pair $[s_0, s_1, s_2, \ldots]$, $[s_0, s'_1, s'_2, \ldots]$ and between the sequence pair $[s_1, s_2, \ldots]$, $[s'_1, s'_2, \ldots]$. If the utility of a sequence is its maximum reward, we can easily violate stationarity. For example,

$$[4, 3, 0, 0, 0, \ldots] \sim [4, 0, 0, 0, 0, \ldots]$$

but

$$[3, 0, 0, 0, \ldots] \succ [0, 0, 0, 0, \ldots] .$$

We can still define $U^\pi(s)$ as the expected maximum reward obtained by executing π starting in s. The agent's preferences seem peculiar, nonetheless. For example, if the current state s has reward R_{max}, the agent will be indifferent among all actions, but once the action is executed and the agent is no longer in s, it will suddenly start to care about what happens next.

17.4 This is a deceptively simple exercise that tests the student's understanding of the formal definition of MDPs. Some students may need a hint or an example to get started.

a. The key here is to get the max and summation in the right place. For $R(s, a)$ we have

$$U(s) = \max_a [R(s, a) + \gamma \sum_{s'} T(s, a, s')U(s')$$

and for $R(s, a, s')$ we have

$$U(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma U(s')] \, .$$

b. There are a variety of solutions here. One is to create a "pre-state" $pre(s, a, s')$ for every s, a, s', such that executing a in s leads not to s' but to $pre(s, a, s')$. In this state is encoded the fact that the agent came from s and did a to get here. From the pre-state, there is just one action b that always leads to s'. Let the new MDP have transition T', reward R', and discount γ'. Then

$$\begin{aligned}
&T'(s, a, pre(s, a, s')) = T(s, a, s') \\
&T'(pre(s, a, s'), b, s') = 1 \\
&R'(s, a) = 0 \\
&R'(pre(s, a, s'), b) = \gamma^{-\frac{1}{2}} R(s, a, s') \\
&\gamma' = \gamma^{\frac{1}{2}}
\end{aligned}$$

c. In keeping with the idea of part (b), we can create states $post(s, a)$ for every s, a, such that

$$\begin{aligned}
&T'(s, a, post(s, a, s')) = 1 \\
&T'(post(s, a, s'), b, s') = T(s, a, s') \\
&R'(s) = 0 \\
&R'(post(s, a, s')) = \gamma^{-\frac{1}{2}} R(s, a) \\
&\gamma' = \gamma^{\frac{1}{2}}
\end{aligned}$$

17.5 This can be done fairly simply by:

- Call `policy-iteration` (from `"uncertainty/algorithms/dp.lisp"`) on the Markov Decision Processes representing the 4x3 grid, with values for the step cost ranging from, say, 0.0 to 1.0 in increments of 0.02.
- For any two adjacent policies that differ, run binary search on the step cost to pinpoint the threshold value.
- Convince yourself that you haven't missed any policies, either by using too coarse an increment in step size (0.02), or by stopping too soon (1.0).

One useful observation in this context is that the expected total reward of any fixed policy is linear in r, the per-step reward for the empty states. Imagine drawing the total reward of a policy as a function of r—a straight line. Now draw all the straight lines corresponding to all possible policies. The reward of the *optimal* policy as a function of r is just the max of all these straight lines. Therefore it is a piecewise linear, convex function of r. Hence there is a very efficient way to find *all* the optimal policy regions:

- For any two consecutive values of r that have different optimal policies, find the optimal policy for the midpoint. Once two consecutive values of r give the same policy, then the interval between the two points must be covered by that policy.
- Repeat this until two points are known for each distinct optimal policy.
- Suppose (r_{a1}, v_{a1}) and (r_{a2}, v_{a2}) are points for policy a, and (r_{b1}, v_{b1}) and (r_{b2}, v_{b2}) are the next two points, for policy b. Clearly, we can draw straight lines through these pairs of points and find their intersection. This does not mean, however, that there is no other optimal policy for the intervening region. We can determine this by calculating the optimal policy for the intersection point. If we get a different policy, we continue the process.

The policies and boundaries derived from this procedure are shown in Figure S17.1. The figure shows that there are *nine* distinct optimal policies! Notice that as r becomes more negative, the agent becomes more willing to dive straight into the –1 terminal state rather than face the cost of the detour to the +1 state.

The somewhat ugly code is as follows. Notice that because the lines for neighboring policies are very nearly parallel, numerical instability is a serious problem.

```
(defun policy-surface (mdp r1 r2 &aux prev (unchanged nil))
  "returns points on the piecewise-linear surface
   defined by the value of the optimal policy of mdp as a
   function of r"
  (setq rvplist
    (list (cons r1 (r-policy mdp r1)) (cons r2 (r-policy mdp r2))))
  (do ()
      (unchanged rvplist)
    (setq unchanged t)
    (setq prev nil)
    (dolist (rvp rvplist)
      (let* ((rest (cdr (member rvp rvplist :test #'eq)))
             (next (first rest))
             (next-but-one (second rest)))
        (dprint (list (first prev) (first rvp)
                      '* (first next) (first next-but-one)))
        (when next
          (unless (or (= (first rvp) (first next))
                      (policy-equal (third rvp) (third next) mdp))
            (dprint "Adding new point(s)")
            (setq unchanged nil)
            (if (and prev next-but-one
                     (policy-equal (third prev) (third rvp) mdp)
                     (policy-equal (third next) (third next-but-one) mdp))
```

```
                (let* ((intxy (policy-vertex prev rvp next next-but-one))
                       (int (cons (xy-x intxy) (r-policy mdp (xy-x intxy)))))
                  (dprint (list "Found intersection" intxy))
                  (cond ((or (< (first int) (first rvp))
                             (> (first int) (first next)))
                         (dprint "Intersection out of range!")
                         (let ((int-r (/ (+ (first rvp) (first next)) 2)))
                           (setq int (cons int-r (r-policy mdp int-r))))
                         (push int (cdr (member rvp rvplist :test #'eq))))
                        ((or (policy-equal (third rvp) (third int) mdp)
                             (policy-equal (third next) (third int) mdp))
                         (dprint "Found policy boundary")
                         (push (list (first int) (second int) (third next))
                               (cdr (member rvp rvplist :test #'eq)))
                         (push (list (first int) (second int) (third rvp))
                               (cdr (member rvp rvplist :test #'eq))))
                        (t (dprint "Found new policy!")
                         (push int (cdr (member rvp rvplist :test #'eq))))))
              (let* ((int-r (/ (+ (first rvp) (first next)) 2))
                     (int (cons int-r (r-policy mdp int-r))))
                (dprint (list "Adding split point" (list int-r (second int))))
                (push int (cdr (member rvp rvplist :test #'eq))))))))
      (setq prev rvp))))

(defun r-policy (mdp r &aux U)
  (set-rewards mdp r)
  (setq U (value-iteration mdp
                           (copy-hash-table (mdp-rewards mdp) #'identity)
                           :epsilon 0.0000000001))
  (list (gethash '(1 1) U) (optimal-policy U (mdp-model mdp) (mdp-rewards mdp))))

(defun set-rewards (mdp r &aux (rewards (mdp-rewards mdp))
                               (terminals (mdp-terminal-states mdp)))
  (maphash #'(lambda (state reward)
               (unless (member state terminals :test #'equal)
                   (setf (gethash state rewards) r)))
           rewards))

(defun policy-equal (p1 p2 mdp &aux (match t)
                                    (terminals (mdp-terminal-states mdp)))
  (maphash #'(lambda (state action)
               (unless (member state terminals :test #'equal)
                 (unless (eq (caar (gethash state p1)) (caar (gethash state p2)))
                   (setq match nil))))
           p1)
  match)

(defun policy-vertex (rvp1 rvp2 rvp3 rvp4)
  (let  ((a (make-xy :x (first rvp1) :y (second rvp1)))
         (b (make-xy :x (first rvp2) :y (second rvp2)))
         (c (make-xy :x (first rvp3) :y (second rvp3)))
         (d (make-xy :x (first rvp4) :y (second rvp4))))
```

```
    (intersection-point (make-line :xy1 a :xy2 b)
                        (make-line :xy1 c :xy2 d))))

(defun intersection-point (l1 l2)
;;; l1 is line ab; l2 is line cd
;;; assume the lines cross at alpha a + (1-alpha) b,
;;;     also known as beta c + (1-beta) d
;;; returns the intersection point unless they're parallel
  (let* ((a (line-xy1 l1))
         (b (line-xy2 l1))
         (c (line-xy1 l2))
         (d (line-xy2 l2))
         (xa (xy-x a)) (ya (xy-y a))
         (xb (xy-x b)) (yb (xy-y b))
         (xc (xy-x c)) (yc (xy-y c))
         (xd (xy-x d)) (yd (xy-y d))
         (q (- (* (- xa xb) (- yc yd))
               (* (- ya yb) (- xc xd)))))
    (unless (zerop q)
      (let ((alpha (/ (- (* (- xd xb) (- yc yd))
                         (* (- yd yb) (- xc xd)))
                      q)))
        (make-xy :x (float (+ (* alpha xa) (* (- 1 alpha) xb)))
                 :y (float (+ (* alpha ya) (* (- 1 alpha) yb))))))))
```

17.6

a. To find the proof, it may help first to draw a picture of two arbitrary functions f and g and mark the maxima; then it is easy to find a point where the difference between the functions is bigger than the difference between the maxima. Assume, w.l.o.g., that $\max_a f(a) \geq \max_a g(a)$, and let f have its maximum value at a^*. Then we have

$$\begin{aligned}
|\max_a f(a) - \max_a g(a)| &= \max_a f(a) - \max_a g(a) && \text{(by assumption)} \\
&= f(a^*) - \max_a g(a) \\
&\leq f(a^*) - g(a^*) \\
&\leq \max_a |f(a) - g(a)| && \text{(by definition of max)}
\end{aligned}$$

b. From the definition of the B operator (Equation (17.6)) we have

$$\begin{aligned}
|(B\,U_i - B\,U_i')(s)| &= |R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s') \\
&\quad - R(s) - \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i'(s')| \\
&= \gamma |\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s') - \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i'(s')| \\
&\leq \gamma \max_{a \in A(s)} |\sum_{s'} P(s' \mid s, a) U_i(s') - \sum_{s'} P(s' \mid s, a) U_i'(s')|
\end{aligned}$$

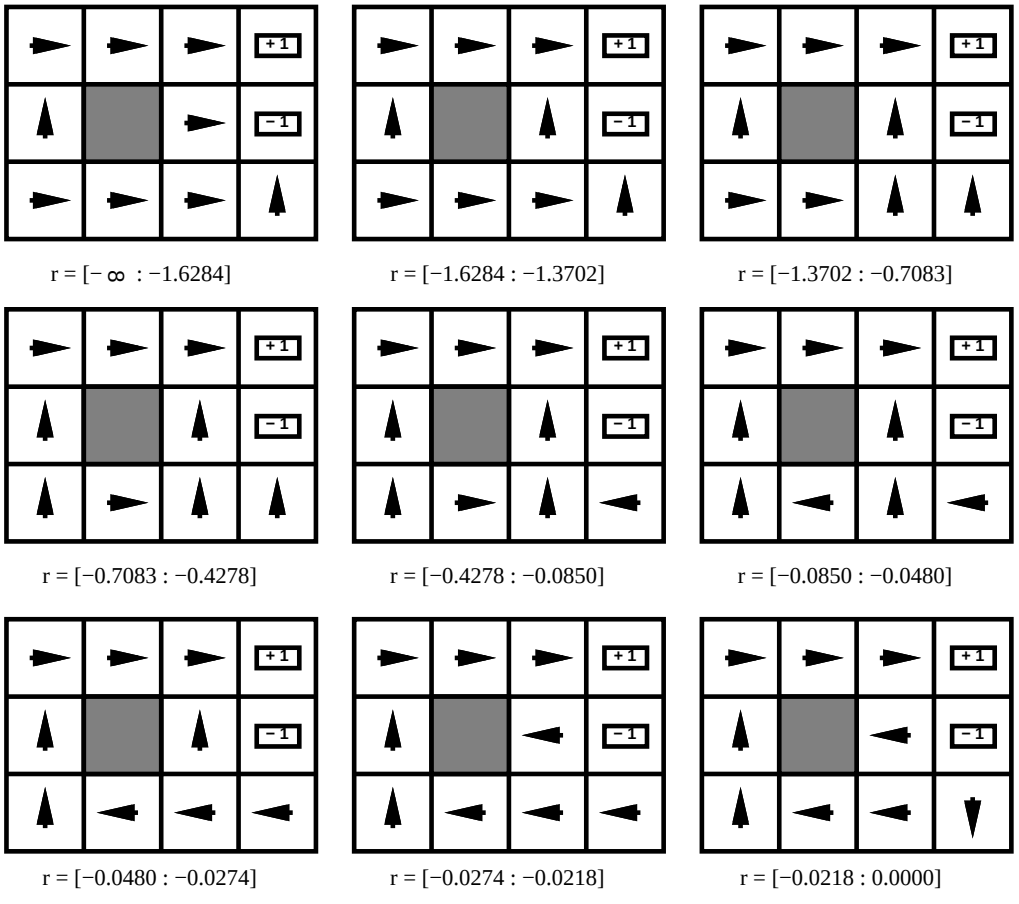


Figure S17.1 Optimal policies for different values of the cost of a step in the 4×3 environment, and the boundaries of the regions with constant optimal policy.

$$
\begin{aligned}
&= \gamma |\sum_{s'} P(s' \mid s, a^*(s)) U_i(s') - \sum_{s'} P(s' \mid s, a^*(s)) U'_i(s')| \\
&= \gamma |\sum_{s'} P(s' \mid s, a^*(s)) (U_i(s') - U'_i(s')|
\end{aligned}
$$

Inserting this into the expression for the max norm, we have

$$
\begin{aligned}
||B\,U_i - B\,U'_i)|| &= \max_s |(B\,U_i - B\,U'_i)(s)| \\
&\leq \gamma \max_s |\sum_{s'} P(s' \mid s, a^*(s)) (U_i(s') - U'_i(s'))| \\
&\leq \gamma \max_s |U_i(s) - U'_i(s)| = \gamma ||U_i - U'_i||
\end{aligned}
$$

17.7

a. For U_A we have

$$U_A(s) = R(s) + \max_a \sum_{s'} P(s'|a,s)U_B(s')$$

and for U_B we have

$$U_B(s) = R(s) + \min_a \sum_{s'} P(s'|a,s)U_A(s') \ .$$

b. To do value iteration, we simply turn each equation from part (a) into a Bellman update and apply them in alternation, applying each to all states simultaneously. The process terminates when the utility vector for one player is the same as the previous utility vector *for the same player* (i.e., two steps earlier). (Note that typically U_A and U_B are not the same in equilibrium.)

c. The state space is shown in Figure S17.2.

d. We mark the terminal state values in bold and initialize other values to 0. Value iteration proceeds as follows:

	(1,4)	(2,4)	(3,4)	(1,3)	(2,3)	(4,3)	(1,2)	(3,2)	(4,2)	(2,1)	(3,1)
U_A	0	0	0	0	0	**+1**	0	0	**+1**	**–1**	**–1**
U_B	0	0	0	0	–1	**+1**	0	–1	**+1**	**–1**	**–1**
U_A	0	0	0	–1	+1	**+1**	–1	+1	**+1**	**–1**	**–1**
U_B	–1	+1	+1	–1	–1	**+1**	–1	–1	**+1**	**–1**	**–1**
U_A	+1	+1	+1	–1	+1	**+1**	–1	+1	**+1**	**–1**	**–1**
U_B	–1	+1	+1	–1	–1	**+1**	–1	–1	**+1**	**–1**	**–1**

and the optimal policy for each player is as follows:

	(1,4)	(2,4)	(3,4)	(1,3)	(2,3)	(4,3)	(1,2)	(3,2)	(4,2)	(2,1)	(3,1)
π_A^*	(2,4)	(3,4)	(2,4)	(2,3)	(4,3)		(3,2)	(4,2)			
π_B^*	(1,3)	(2,3)	(3,2)	(1,2)	(2,1)		(1,3)	(3,1)			

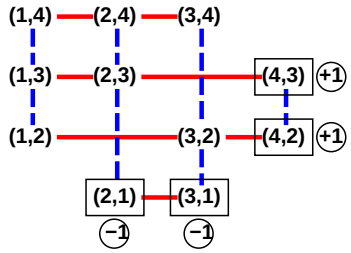


Figure S17.2 State-space graph for the game in Figure 5.17.

17.8

a. $r = 100$.

u	l	.
u	l	d
u	l	l

See the comments for part d. This should have been $r = -100$ to illustrate an alternative behavior:

r	r	.
d	r	u
r	r	u

Here, the agent tries to reach the goal quickly, subject to attempting to avoid the square $(1, 3)$ as much as possible. Note that the agent will choose to move Down in square $(1, 2)$ in order to actively avoid the possibility of "accidentally" moving into the square $(1, 3)$ if it tried to move Right instead, since the penalty for moving into square (1, 3) is so great.

b. $r = -3$.

r	r	.
r	r	u
r	r	u

Here, the agent again tries to reach the goal as fast as possible while attempting to avoid the square $(1, 3)$, but the penalty for square $(1, 3)$ is not so great that the agent will try to actively avoid it at all costs. Thus, the agent will choose to move Right in square $(1, 2)$ in order to try to get closer to the goal even if it occasionally will result in a transition to square $(1, 3)$.

c. $r = 0$.

r	r	.
u	u	u
u	u	u

Here, the agent again tries to reach the goal as fast as possible, but will try to do so via a path that includes square $(1, 3)$ if possible. This results from the fact that square $(1, 3)$ does not incur the reward of -1 in all other non-goal states, so it reaching the goal via a path through that square can potentially have slightly greater reward than another path of equal length that does not pass through $(1, 3)$.

d. $r = 3$.

u	l	.
u	l	d
u	l	l

17.9 The utility of Up is

$$50\gamma - \sum_{t=2}^{101} \gamma^t = 50\gamma - \gamma^2 \frac{1 - \gamma^{100}}{1 - \gamma},$$

while the utility of Down is

$$-50\gamma + \sum_{t=2}^{101} \gamma^t = -50\gamma + \gamma^2 \frac{1 - \gamma^{100}}{1 - \gamma}.$$

Solving numerically we find the indifference point to be $\gamma \approx 0.9844$: larger than this and we want to go Down to avoid the expensive long-term consequences, smaller than this and we want to go Up to get the immediate benefit.

17.10

a. Intuitively, the agent wants to get to state 3 as soon as possible, because it will pay a cost for each time step it spends in states 1 and 2. However, the only action that reaches state 3 (action b) succeeds with low probability, so the agent should minimize the cost it incurs while trying to reach the terminal state. This suggests that the agent should definitely try action b in state 1; in state 2, it might be better to try action a to get to state 1 (which is the better place to wait for admission to state 3), rather than aiming directly for state 3. The decision in state 2 involves a numerical tradeoff.

b. The application of policy iteration proceeds in alternating steps of value determination and policy update.

- *Initialization*: $U \leftarrow \langle -1,\ -2,\ 0\rangle$, $P \leftarrow \langle b,\ b\rangle$.
- *Value determination*:

$$\begin{aligned} u_1 &= -1 + 0.1u_3 + 0.9u_1 \\ u_2 &= -2 + 0.1u_3 + 0.9u_2 \\ u_3 &= 0 \end{aligned}$$

That is, $u_1 = -10$ and $u_2 = -20$.

Policy update: In state 1,

$$\sum_j T(1, a, j)u_j = 0.8 \times -20 + 0.2 \times -10 = -18$$

while

$$\sum_j T(1, b, j)u_j = 0.1 \times 0 \times 0.9 \times -10 = -9$$

so action b is still preferred for state 1.

In state 2,

$$\sum_j T(1, a, j)u_j = 0.8 \times -10 + 0.2 \times -20 = -12$$

while

$$\sum_j T(1, b, j)u_j = 0.1 \times 0 \times 0.9 \times -20 = -18$$

so action a is preferred for state 1. We set $unchanged? \leftarrow \text{false}$ and proceed.

- *Value determination*:

$$\begin{aligned} u_1 &= -1 + 0.1u_3 + 0.9u_1 \\ u_2 &= -2 + 0.8u_1 + 0.2u_2 \\ u_3 &= 0 \end{aligned}$$

Once more $u_1 = -10$; now, $u_2 = -15$. *Policy update*: In state 1,

$$\sum_j T(1, a, j)u_j = 0.8 \times -15 + 0.2 \times -10 = -14$$

while

$$\sum_j T(1,b,j)u_j = 0.1 \times 0 \times 0.9 \times -10 = -9$$

so action b is still preferred for state 1.

In state 2,

$$\sum_j T(1,a,j)u_j = 0.8 \times -10 + 0.2 \times -15 = -11$$

while

$$\sum_j T(1,b,j)u_j = 0.1 \times 0 \times 0.9 \times -15 = -13.5$$

so action a is still preferred for state 1. $unchanged?$ remains true, and we terminate.

Note that the resulting policy matches our intuition: when in state 2, try to move to state 1, and when in state 1, try to move to state 3.

c. An initial policy with action a in both states leads to an unsolvable problem. The initial value determination problem has the form

$$\begin{aligned} u_1 &= -1 + 0.2u_1 + 0.8u_2 \\ u_2 &= -2 + 0.8u_1 + 0.2u_2 \\ u_3 &= 0 \end{aligned}$$

and the first two equations are inconsistent. If we were to try to solve them iteratively, we would find the values tending to $-\infty$.

Discounting leads to well-defined solutions by bounding the penalty (expected discounted cost) an agent can incur at either state. However, the choice of discount factor will affect the policy that results. For γ small, the cost incurred in the distant future plays a negligible role in the value computation, because γ^n is near 0. As a result, an agent could choose action b in state 2 because the discounted short-term cost of remaining in the non-terminal states (states 1 and 2) outweighs the discounted long-term cost of action b failing repeatedly and leaving the agent in state 2. An additional exercise could ask the student to determine the value of γ at which the agent is indifferent between the two choices.

17.11 The framework for this problem is in `"uncertainty/domains/4x3-mdp.lisp"`. There is still some synthesis for the student to do for answer b. For c. some experimental design is necessary.

17.12 (Note: Early printings used "value determination," a term accidentally left over from the second edition, instead of "policy evaluation.") The policy evaluation algorithm calculates $U^\pi(s)$ for a given policy π. The policy for an agent that thinks $U$is the true utility and P is the true model would be based on Equation (17.4):

$$\pi(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s,a)U(s') \ .$$

Given this policy, the policy loss compared to the true optimal policy, starting in state s, is just $U^{\pi^*}(s) - U^{\pi}(s)$.

17.13 The belief state update is given by Equation (17.11), i.e.,

$$b'(s') = \alpha\, P(e \mid s') \sum_s P(s' \mid s, a) b(s) \ .$$

It may be helpful to compute this in two stages: update for the action, then update for the observation. The observation probabilities $P(e \mid s')$ are all either 0.9 (for squares that actually have one wall) or 0.1 (for squares with two walls). The following table shows the results. Note in particular how the probability mass concentrates on (3,2).

		Left	1 wall	*Left*	1 wall
(1,1)	.11111	.20000	.06569	.09197	.02090
(1,2)	.11111	.11111	.03650	.04234	.00962
(1,3)	.11111	.20000	.06569	.09197	.02090
(2,1)	.11111	.11111	.03650	.27007	.06136
(2,3)	.11111	.11111	.03650	.05985	.01360
(3,1)	.11111	.11111	.32847	.06861	.14030
(3,2)	.11111	.11111	.32847	.30219	.61791
(3,3)	.11111	.02222	.06569	.03942	.08060
(4,1)	.11111	.01111	.00365	.00036	.00008
(4,2)	0	.01111	.03285	.03321	.06791
(4,3)	0	0	0	0	0

17.14 In a sensorless environment, POMDP value iteration is essentially the same as ordinary state-space search—the branching ocurs only on acton choices, not observations. Hence the time complexity is $O(|A|^d)$.

17.15 Policies for the 2-state MDP all have a threshold belief p, such that if $b(0) > p$ then the optimal action is Go, otherwise it is $Stay$. The question is, what does this change do to the threshold? By making sensing more informative in state 0 and less informative in state 1, the change has made state 0 more desirable, hence the threshold value p *increases*.

17.16 This question is simple a matter of examining the definitions. In a dominant strategy equilibrium $[s_1, \ldots, s_n]$, it is the case that for every player i, s_i is optimal for every combination t_{-i} by the other players:

$$\forall\, i \ \ \forall\, t_{-i} \ \ \forall\, s'_i \ \ [s_i, t_{-i}] \succsim [s'_i, t_{-i}] \ .$$

In a Nash equilibrium, we simply require that s_i is optimal for the particular current combination s_{-i} by the other players:

$$\forall\, i \ \ \forall\, s'_i \ \ [s_i, s_{-i}] \succsim [s'_i, s_{-i}] \ .$$

Therefore, dominant strategy equilibrium is a special case of Nash equilibrium. The converse does not hold, as we can show simply by pointing to the CD/DVD game, where neither of the Nash equilibria is a dominant strategy equilibrium.

17.17 In the following table, the rows are labelled by A's move and the columns by B's move, and the table entries list the payoffs to A and B respectively.

	R	P	S	F	W
R	0,0	–1,1	1,-1	–1,1	1,-1
P	1,-1	0,0	–1,1	–1,1	1,-1
S	–1,1	1,-1	0,0	–1,1	1,-1
F	1,-1	1,-1	1,-1	0,0	–1,1
W	–1,1	–1,1	–1,1	1,-1	0,0

Suppose A chooses a mixed strategy $[r : R;\ p : P;\ s : S;\ f : F;\ w : W]$, where $r + p + s + f + w = 1$. The payoff *to A* of B's possible pure responses are as follows:

$$\begin{aligned} R &: +p - s + f - w \\ P &: -r + s + f - w \\ S &: +r - p + f - w \\ F &: -r - p - s + w \\ W &: +r + p + s - f \end{aligned}$$

It is easy to see that no option is dominated over the whole region. Solving for the intersection of the hyperplanes, we find $r = p = s = 1/9$ and $f = w = 1/3$. By symmetry, we will find the same solution when B chooses a mixed strategy first.

17.18 We apply iterated strict dominance to find the pure strategy. First, *Pol: do nothing* dominates *Pol: contract*, so we drop the *Pol: contract* row. Next, *Fed: contract* dominates *Fed: do nothing* and *Fed: expand* on the remaining rows, so we drop those columns. Finally, *Pol: expand* dominates *Pol: do nothing* on the one remaining column. Hence the only Nash equilibrium is a dominant strategy equilibrium with *Pol: expand* and *Fed: contract*. This is not Pareto optimal: it is worse for both players than the four strategy profiles in the top right quadrant.

17.19 This question really has two answers, depending on what assumption is made about the probability distribution over bidder's private valuations v_i for the item.

In a Dutch auction, just as in a first-price sealed-bid auction, bidders must estimate the likely private values of the other bidders. When the price is higher than v_i, agent i will not bid, but as soon as the price reaches v_i, he faces a dilemma: bid now and win the item at a higher price than necessary, or wait and risk losing the item to another bidder. In the standard models of auction theory, each bidder, in addition to a private value v_i, has a probability density $p_i(v_1, \ldots, v_n)$ over the private values of all n bidders for the item. In particular, we consider **independent private values**, so that the distribution over the other bidders' values is independent of v_i. Each bidder will choose a bid—i.e., the first price at which they will bid if that price is reached—through a bidding function $b_i(v_i)$.

INDEPENDENT PRIVATE VALUES

We are interested in finding a Nash equilibrium (technically a **Bayes–Nash equilibrium** in which each bidder's bidding function is optimal given the bidding functions of the other agents. Under risk-neutrality, optimality of a bid b is determined by the expected payoff, i.e., the probability of winning the auction with bid b times the profit when paying that amount

for the item. Now, agent i wins the auction with bid b if all the other bids are less than b; let the probability of this happening be $W_i(b)$ for whatever fixed bidding functions the other bidders use. ($W_i(b)$ is thus a cumulative probability distribution and nondecreasing in b; under independent private values, it does not depend on v_i.) Then we can write the expected payoff for agent i as

$$Q_i(v_i, b) = W_i(b)(v_i - b)$$

and the optimality condition in equilibrium is therefore

$$\forall\, i, b \;\; W_i(b_i(v_i))(v_i - b_i(v_i)) \geq W_i(b)(v_i - b)\ . \tag{17.1}$$

We now prove that the bidding functions $b_i(v_i)$ must be **monotonic**, i.e., *nondecreasing* in the private valuation v_i. Let v and v' be two different valuations, with $b = b_i(v)$ and $b' = b_i(v')$. Applying Equation (17.1) twice, first to say that (v, b) is better than (v, b') and then to say that (v', b') is better than (v', b), we obtain

$$\begin{aligned} W_i(b)(v - b) &\;\geq\; W_i(b')(v - b') \\ W_i(b')(v' - b') &\;\geq\; W_i(b)(v' - b) \end{aligned}$$

Rearranging, these become

$$\begin{aligned} v(W_i(b) - W_i(b')) &\;\geq\; W_i(b)b - W_i(b')b' \\ v'(W_i(b') - W_i(b)) &\;\geq\; W_i(b')b' - W_i(b)b \end{aligned}$$

Adding these equations, we have

$$(v' - v)(W_i(b') - W_i(b)) \geq 0$$

from which it follows that if $v' > v$, then $W_i(b') \geq W_i(b)$. Monotonicity does not follow immediately, however; we have to handle two cases:

- If $W_i(b') > W_i(b)$, or if W_i is strictly increasing, then $b' \geq b$ and $b_i(\cdot)$ is monotonic.
- Otherwise, $W_i(b') = W_i(b)$ and W_i is flat between b and b'. Now if W_i is flat in any interval $[x, y]$, then an optimal bidding function will prefer x over any other bid in the interval since that maximizes the profit on winning without affecting the probability of winning; hence, we must have $b' = b$ and again $b_i(\cdot)$ is monotonic.

Intuitively, the proof amounts to the following: if a higher valuation could result in a lower bid, then by swapping the two bids the agent could increase the *sum* of the payoffs for the two bids, which means that *at least one* of the two original bids is suboptimal.

Returning to the question of efficiency—the property that the item goes to the bidder with the highest valuation—we see that it follows immediately from monotonicity in the case where the bidders' prior distributions over valuations are **symmetric** or identically distributed.[1]

[1] According to Milgrom (1989), Vickrey (1961) proved that under this assumption, the Dutch auction is efficient. Vickrey's argument in Appendix III for the monotonicity of the bidding function is similar to the argument above but, as written, seems to apply only to the uniform-distribution case he was considering. Indeed, much of his analysis beginning with Appendix II is based on an inverse bidding function, which implicitly *assumes* monotonicity of the bidding function. Many other authors also begin by assuming monotonicity, then derive the form of the optimal bidding function, and then show it is monotonic. This proves the existence of an equilibrium with monotonic bidding functions, but not that all equilibria have this property.

Vickrey (1961) proves that the auction is *not* efficient in the asymmetric case where one player's distribution is uniform over $[0, 1]$ and the other's is uniform over $[a, b]$ for $a > 0$. Milgrom (1989) provides another, more transparent example of inefficiency: Suppose Alice has a known, fixed value of \$101 for an item, while Bob's value is \$50 with probability 0.8 and \$75 with probability 0.2. Given that Bob will never bid higher than his valuation, Alice can see that a bid of \$51 will win *at least* 80% of the time, giving an expected profit of *at least* $0.8 \times (\$101 - \$51) = \$40$. On the other hand, any bid of \$62 or more cannot yield an expected profit at most \$39, regardless of Bob's bid, and so is dominated by the bid of \$51. Hence, in any equilibrium, Alice's bid at most \$61. Knowing this, Bob can bid \$62 whenever his valuation is \$75 and be sure of winning. Thus, with 20% probability, the item goes to Bob, whose valuation for it is lower than Alice's. This violates efficiency.

Besides efficiency in the symmetric case, monotonicity has another important consequence for the analysis of the Dutch (and first-price) auction : it makes it possible to derive the exact form of the bidding function. As it stands, Equation (17.1) is difficult or impossible to solve because the cumulative distribution of the other bidders' bids, $W_i(b)$, depends on their bidding functions, so all the bidding functions are coupled together. (Note the similarity to the Bellman equations for an MDP.) With monotonicity, however, we can define W_i in terms of the known valuation distributions. Assuming independence and symmetry, and writing $b_i^{-1}(b)$ for the inverse of the (monotonic) bidding function, we have

$$Q_i(v_i, b) = (P(b_i^{-1}(b)))^{n-1}(v_i - b)$$

where $P(v)$ is the probability that an individual valuation is less than v. At equilibrium, where b maximizes Q_i, the first derivative must be zero:

$$\frac{\partial Q}{\partial b} = 0 = \frac{(n-1)(P(b_i^{-1}(b)))^{n-2}p(b_i^{-1}(b))(v_i - b)}{b_i'(b_i^{-1}(b))} - (P(b_i^{-1}(b)))^{n-1}$$

where we have used the fact that $df^{-1}(x)/dx = 1/f'(f^{-1}(x))$.

For an equilibrium bidding function, of course, $b_i^{-1}(b) = v_i$; substituting this and simplifying, we find the following differential equation for b_i:

$$b_i'(v_i) = (v_i - b_i(v_i)) \cdot (n-1)p(v_i)/P(v_i) \ .$$

To find concrete solutions we also need to establish a boundary condition. Suppose v_0 is the lowest possible valuation for the item; then we must have $b_i(v_0) = v_0$ (Milgrom and Weber, 1982). Then the solution, as shown by McAfee and McMillan (1987), is

$$b_i(v_i) = v_i - \frac{\int_{v_0}^{v_i}(P(v))^{n-1}dv}{(P(v_i))^{n-1}} \ .$$

For example, suppose p is uniform in $[0, 1]$; then $P(v) = v$ and $b_i(v_i) = v_i \cdot (n-1)/n$, which is the classical result obtained by Vickrey (1961).

17.20 In such an auction it is rational to continue bidding as long as winning the item would yield a profit, i.e., one is willing to bid up to $2v_i$. The auction will end at $2v_o + d$, so the winner will pay $v_o + d/2$, slightly less than in the regular version.

17.21 Every game is either a win for one side (and a loss for the other) or a tie. With 2 for a win, 1 for a tie, and 0 for a loss, 2 points are awarded for every game, so this is a constant-sum game.

If 1 point is awarded for a loss in overtime, then for some games 3 points are awarded in all. Therefore, the game is no longer constant-sum.

Suppose we assume that team A has probability r of winning in regular time and team B has probability s of winning in regular time (assuming normal play). Furthermore, assume team B has a probability q of winning in overtime (which occurs if there is a tie after regular time). Once overtime is reached (by any means), the expected utilities are as follows:

$$\begin{aligned} U_A^O &= 1 + p \\ U_B^O &= 1 + q \end{aligned}$$

In normal play, the expected utilities are derived from the probability of winning plus the probability of tying times the expected utility of overtime play:

$$\begin{aligned} U_A &= 2r + (1 - r - s)(1 + p) \\ U_B &= 2s + (1 - r - s)(1 + q) \end{aligned}$$

Hence A has an incentive to agree if $U_A^O > U_A$, or

$$1 + p > 2r + (1 - r - s)(1 + p) \quad \text{or} \quad rp - r + sp + s > 0 \quad \text{or} \quad p > \frac{r - s}{r + s}$$

and B has an incentive to agree if $U_B^O > U_B$, or

$$1 + q > 2s + (1 - r - s)(1 + q) \quad \text{or} \quad sq - s + rq + r > 0 \quad \text{or} \quad q > \frac{s - r}{r + s}$$

When both of these inequalities hold, there is an incentive to tie in regulation play. For any values of r and s, there will be values of p and q such that both inequalities hold.

For an in-depth statistical analysis of the actual effects of the rule change and a more sophisticated treatment of the utility functions, see "Overtime! Rules and Incentives in the National Hockey League" by Stephen T. Easton and Duane W. Rockerbie, available at `http://people.uleth.ca/~rockerbie/OVERTIME.PDF`.

Solutions for Chapter 18
Learning from Examples

18.1 The aim here is to couch language learning in the framework of the chapter, not to solve the problem! This is a very interesting topic for class discussion, raising issues of nature vs. nurture, the indeterminacy of meaning and reference, and so on. Basic references include Chomsky (1957) and Quine (1960).

The first step is to appreciate the variety of knowledge that goes under the heading "language." The infant must learn to recognize and produce speech, learn vocabulary, learn grammar, learn the semantic and pragmatic interpretation of a speech act, and learn strategies for disambiguation, among other things. The performance elements for this (in humans) and their associated learning mechanisms are obviously very complex and as yet little is known about them.

A naive model of the learning environment considers just the exchange of speech sounds. In reality, the physical context of each utterance is crucial: a child must see the context in which "watermelon" is uttered in order to learn to associate "watermelon" with watermelons. Thus, the environment consists not just of other humans but also the physical objects and events about which discourse takes place. Auditory sensors detect speech sounds, while other senses (primarily visual) provide information on the physical context. The relevant effectors are the speech organs and the motor capacities that allow the infant to respond to speech or that elicit verbal feedback.

The performance standard could simply be the infant's general utility function, however that is realized, so that the infant performs reinforcement learning to perform and respond to speech acts so as to improve its well-being—for example, by obtaining food and attention. However, humans' built-in capacity for mimicry suggests that the production of sounds similar to those produced by other humans is a goal in itself. The child (once he or she learns to differentiate sounds and learn about pointing or other means of indicating salient objects) is also exposed to examples of supervised learning: an adult says "shoe" or "belly button" while indicating the appropriate object. So sentences produced by adults provide labelled positive examples, and the response of adults to the infant's speech acts provides further classification feedback.

Mostly, it seems that adults do not correct the child's speech, so there are very few negative classifications of the child's attempted sentences. This is significant because early work on language learning (such as the work of Gold, 1967) concentrated just on identifying the set of strings that are grammatical, assuming a particular grammatical formalism. If there are

only positive examples, then there is nothing to rule out the grammar $S \rightarrow Word^*$. Some theorists (notably Chomsky and Fodor) used what they call the "poverty of the stimulus" argument to say that the basic universal grammar of languages must be innate, because otherwise (given the lack of negative examples) there would be no way that a child could learn a language (under the assumptions of language learning as learning a set of grammatical strings). Critics have called this the "poverty of the imagination" argument—I can't think of a learning mechanism that would work, so it must be innate. Indeed, if we go to probabilistic context free grammars, then it is possible to learn a language without negative examples.

18.2 Learning tennis is much simpler than learning to speak. The requisite skills can be divided into movement, playing strokes, and strategy. The environment consists of the court, ball, opponent, and one's own body. The relevant sensors are the visual system and proprioception (the sense of forces on and position of one's own body parts). The effectors are the muscles involved in moving to the ball and hitting the stroke. The learning process involves both supervised learning and reinforcement learning. Supervised learning occurs in acquiring the predictive transition models, e.g., where the opponent will hit the ball, where the ball will land, and what trajectory the ball will have after one's own stroke (e.g., if I hit a half-volley *this* way, it goes into the net, but if I hit it *that* way, it clears the net). Reinforcement learning occurs when points are won and lost—this is particularly important for strategic aspects of play such as shot placement and positioning (e.g., in 60% of the points where I hit a lob in response to a cross-court shot, I end up losing the point). In the early stages, reinforcement also occurs when a shot succeeds in clearing the net and landing in the opponent's court. Achieving this small success is itself a sequential process involving many motor control commands, and there is no teacher available to tell the learner's motor cortex which motor control commands to issue.

18.3 The algorithm may not return the "correct" tree, but it will return a tree that is logically equivalent, assuming that the method for generating examples eventually generates all possible combinations of input attributes. This is true because any two decision tree defined on the same set of attributes that agree on all possible examples are, by definition, logically equivalent. The actually form of the tree may differ because there are many different ways to represent the same function. (For example, with two attributes A and B we can have one tree with A at the root and another with B at the root.) The root attribute of the original tree may not in fact be the one that will be chosen by the information gain geuristic when applied to the training examples.

18.4 This question brings a little bit of mathematics to bear on the analysis of the learning problem, preparing the ground for Chapter 20. Error minimization is a basic technique in both statistics and neural nets. The main thing is to see that the error on a given training set can be written as a mathematical expression and viewed as a function of the hypothesis chosen. Here, the hypothesis in question is a single number $\alpha \in [0, 1]$ returned at the leaf.

a. If α is returned, the absolute error is

$$\begin{aligned} E = p(1-\alpha) + n\alpha = \alpha(n-p) + p &= n \text{ when } \alpha = 1 \\ &= p \text{ when } \alpha = 0 \end{aligned}$$

This is minimized by setting

$\alpha = 1$ if $p > n$
$\alpha = 0$ if $p < n$

That is, α is the majority value.

b. First calculate the sum of squared errors, and its derivative:

$$E = p(1-\alpha)^2 + n\alpha^2$$
$$\frac{dE}{d\alpha} = 2\alpha n - 2p(1-\alpha) = 2\alpha(p+n) - 2p$$

The fact that the second derivative, $\frac{d^2E}{d\alpha^2} = 2(p+n)$, is greater than zero means that E is minimized (not maximized) where $\frac{dE}{d\alpha} = 0$, i.e., when $\alpha = \frac{p}{p+n}$.

18.5 This result emphasizes the fact that any statistical fluctuations caused by the random sampling process will result in an apparent information gain.

The easy part is showing that the gain is zero when each subset has the same ratio of positive examples. The gain is defined as

$$B\left(\frac{p}{p+n}\right) - \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$$

Since $p = \sum p_k$ and $n = \sum n_k$, if $p_k/(p_k+n_k)$ is the same for all k we must have $p_k/(p_k+n_k) = p/(p+n)$ for all k. From this, we obtain

$$\begin{aligned} Gain &= B\left(\frac{p}{p+n}\right) - B\left(\frac{p}{p+n}\right)\frac{1}{p+n}\sum_{k=1}^{d} p_k + n_k \\ &= B\left(\frac{p}{p+n}\right) - B\left(\frac{p}{p+n}\right)\frac{1}{p+n}(p+n) = 0 \end{aligned}$$

Note that this holds for all values of $p_k + n_k$. To prove that the value is positive elsewhere, we can apply the method of Lagrange multipliers to show that this is the only stationary point; the gain is clearly positive at the extreme values, so it is positive everywhere but the stationary point. In detail, we have constraints $\sum_k p_k = p$ and $\sum_k n_k = n$, and the Lagrange function is

$$\Lambda = B\left(\frac{p}{p+n}\right) - \sum_k \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right) + \lambda_1\left(p - \sum_k p_k\right) + \lambda_2\left(n - \sum_k n_k\right) .$$

Setting its derivatives to zero, we obtain, for each k,

$$\begin{aligned} \frac{\partial\Lambda}{\partial p_k} &= -\frac{1}{p+n}B\left(\frac{p_k}{p_k+n_k}\right) - \frac{p_k+n_k}{p+n}\log\frac{p_k}{n_k}\left(\frac{1}{p_k+n_k} - \frac{p_k}{(p_k+n_k)^2}\right) - \lambda_1 = 0 \\ \frac{\partial\Lambda}{\partial n_k} &= -\frac{1}{p+n}B\left(\frac{p_k}{p_k+n_k}\right) - \frac{p_k+n_k}{p+n}\log\frac{p_k}{n_k}\left(\frac{-p_k}{(p_k+n_k)^2}\right) - \lambda_2 = 0 . \end{aligned}$$

Subtracting these two, we obtain $\log(p_k/n_k) = (p+n)(\lambda_2 - \lambda_1)$ for all k, implying that at any stationary point the ratios p_k/n_k must be the same for all k. Given the two summation constraints, the only solution is the one given in the question.

18.6 Note that to compute each split, we need to compute $Remainder(A_i)$ for each attribute A_i, and select the attribute the provides the minimal remaining information, since the existing information prior to the split is the same for all attributes we may choose to split on.

Computations for first split: remainders for $A1$, $A2$, and $A3$ are

$$(4/5)(-2/4\log(2/4) - 2/4\log(2/4)) + (1/5)(-0 - 1/1\log(1/1)) = 0.800$$
$$(3/5)(-2/3\log(2/3) - 1/3\log(1/3)) + (2/5)(-0 - 2/2\log(2/2)) \approx 0.551$$
$$(2/5)(-1/2\log(1/2) - 1/2\log(1/2)) + (3/5)(-1/3\log(1/3) - 2/3\log(2/3)) \approx 0.951$$

Choose A_2 for first split since it minimizes the remaining information needed to classify all examples. Note that all examples with $A_2 = 0$, are correctly classified as $B = 0$. So we only need to consider the three remaining examples (x_3, x_4, x_5) for which $A_2 = 1$.

After splitting on A_2, we compute the remaining information for the other two attributes on the three remaining examples (x_3, x_4, x_5) that have $A_2 = 1$. The remainders for $A1$ and $A3$ are

$$(2/3)(-2/2\log(2/2) - 0) + (1/3)(-0 - 1/1\log(1/1)) = 0$$
$$(1/3)(-1/1\log(1/1) - 0) + (2/3)(-1/2\log(1/2) - 1/2\log(1/2)) \approx 0.667.$$

So, we select attribute A1 to split on, which correctly classifies all remaining examples.

18.7 See Figure S18.1, where nodes on successive rows measure attributes A_1, A_2, and A_3. (Any fixed ordering works.)

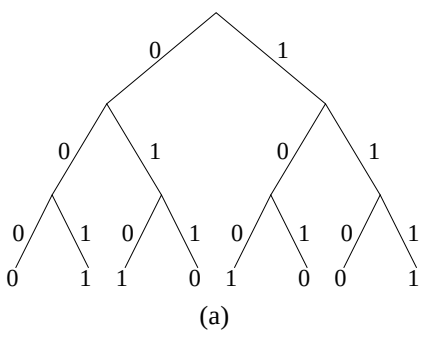


Figure S18.1 XOR function representations: (a) decision tree, and (b) decision graph.

18.8 This is a fairly small, straightforward programming exercise. The only hard part is the actual χ^2 computation; you might want to provide your students with a library function to do this.

18.9 This is another straightforward programming exercise. The follow-up exercise is to run tests to see if the modified algorithm actually does better.

18.10 Let the prior probabilities of each attribute value be $P(v_1), \ldots, P(v_n)$. (These probabilities are estimated by the empirical fractions among the examples at the current node.)

From page 540, the intrinsic information content of the attribute is

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log v_i$$

Given this formula and the empirical estimates of $P(v_i)$, the modification to the code is straightforward.

18.11 If we leave out an example of one class, then the majority of the remaining examples are of the other class, so the majority classifier will always predict the wrong answer.

18.12

Test	If yes	If no
$A_1 = 1$	1	next test
$A_3 = 1 \wedge A_4 = 0$	0	next test
$A_2 = 0$	0	1

18.13

Proof (sketch): Each path from the root to a leaf in a decision tree represents a logical conjunction that results in a classification at the leaf node. We can simply create a decision list by producing one rule to correspond to each such path through the decision tree where the rule in the decision list has the test given by the logical conjunction in the path and the output for the rule is the corresponding classification at the leaf of the path. Thus we produce one rule for each leaf in the decision tree (since each leaf determines a unique path), constructing a decision list that captures the same function represented in the decision tree.

A simple example of a function that can be represented with strictly fewer rules in a decision list than the number of leaves in a minimal sized decision tree is the logical conjunction of two boolean attributes: $A_1 \wedge A_2 \Rightarrow T$.

The decision list has the form:

Test	If yes	If no
$A_1 = T \wedge A_2 = T$	T	F

Note: one could consider this either one rule, or at most two rules if we were to represent it as follows:

Test	If yes	If no
$A_1 = T \wedge A_2 = T$	T	next test
T	F	

In either case, the corresponding decision tree has three leaves.

18.14 Note: this is the only exercise to cover the material in section 18.6. Although the basic ideas of computational learning theory are both important and elegant, it is not easy to find good exercises that are suitable for an AI class as opposed to a theory class. If you are teaching a graduate class, or an undergraduate class with a strong emphasis on learning, it might be a good idea to use some of the exercises from Kearns and Vazirani (1994).

a. If each test is an arbitrary conjunction of literals, then a decision list can represent an arbitrary DNF (disjunctive normal form) formula directly. The DNF expression $C_1 \vee C_2 \vee \cdots \vee C_n$, where C_i is a conjunction of literals, can be represented by a

decision list in which C_i is the ith test and returns $True$ if successful. That is:

$C_1 \rightarrow True$;
$C_2 \rightarrow True$;
$\ldots$
$C_n \rightarrow True$;
$True \rightarrow False$

Since any Boolean function can be written as a DNF formula, then any Boolean function can be represented by a decision list.

b. A decision tree of depth k can be translated into a decision list whose tests have at most k literals simply by encoding each path as a test. The test returns the corresponding leaf value if it succeeds. Since the decision tree has depth k, no path contains more than k literals.

18.15 The L_1 loss is minimized by the median, in this case 7, and the L_2 loss by the mean, in this case $143/7$.

For the first, suppose we have an odd number $2n+1$ of elements $y_{-n} < \ldots < y_0 < \ldots < y_n$. For $n = 0$, $\hat{y} = y_0$ is the median and minimizes the loss. Then, observe that the L_1 loss for $n+1$ is

$$\frac{1}{2n+3}\sum_{i=-(n+1)}^{n+1}|\hat{y}-y_i| = \frac{1}{2n+3}\left(|\hat{y}-y_{n+1}| + |\hat{y}-y_{-(n+1)}|\right) + \frac{1}{2n+3}\sum_{i=-n}^{n}|\hat{y}-y_i|$$

The first term is equal to $|y_{n+1} - y_{-(n+1)}|$ whenever $y_{n+1} \leq \hat{y} \leq y_{-(n+1)}$, e.g. for $\hat{y} = y_0$, and is strictly larger otherwise. But by inductive hypothesis the second term also is minimized by $\hat{y} = y_0$, the median.

For the second, notice that as the L_2 loss of $\hat{y}$ given data $y_1, \ldots, y_n$

$$\frac{1}{n}\sum_i(\hat{y}-y_i)^2$$

is differentiable we can find critical points:

$$0 = \frac{2}{n}\sum_i(\hat{y}-y_i)$$

or $\hat{y} = (1/n)\sum_i y_i$. Taking the second derivative we see this is the unique local minimum, and thus the global minimum as the loss is infinite when $\hat{y}$ tends to either infinity.

18.16

a. The circle equation expands into five terms

$$0 = x_1^2 + x_2^2 - 2ax_1 - 2bx_2 + (a^2 + b^2 - r^2)$$

corresponding to weights $w = (2a, 2b, 1, 1)$ and intercept $a^2 + b^2 - r^2$. This shows that a circular boundary is linear in this feature space, allowing linear separability.

In fact, the three features $x_1, x_2, x_1^2 + x_2^2$ suffice.

b. The (axis-aligned) ellipse equation expands into six terms

$$0 = cx_1^2 + dx_2^2 - 2acx_1 - 2bdx_2 + (a^2c + b^2d - 1)$$

corresponding to weights $w = (2ac, 2bd, c, d, 0)$ and intercept $a^2 + b^2 - r^2$. This shows that an elliptical boundary is linear in this feature space, allowing linear separability.

In fact, the four features x_1, x_2, x_1^2, x_2^2 suffice for any axis-aligned ellipse.

18.17 The examples map from $[x_1, x_2]$ to $[x_1, x_1, x_2]$ coordinates as follows:
$[-1, -1]$ (negative) maps to $[-1, +1]$
$[-1, +1]$ (positive) maps to $[-1, -1]$
$[+1, -1]$ (positive) maps to $[+1, -1]$
$[+1, +1]$ (negative) maps to $[+1, +1]$
Thus, the positive examples have $x_1 x_2 = -1$ and the negative examples have $x_1 x_2 = +1$. The maximum margin separator is the line $x_1 x_2 = 0$, with a margin of 1. The separator corresponds to the $x_1 = 0$ *and* $x_2 = 0$ axes in the original space—this can be thought of as the limit of a hyperbolic separator with two branches.

18.18

18.19 XOR (in fact any Boolean function) is easiest to construct using step-function units. Because XOR is not linearly separable, we will need a hidden layer. It turns out that just one hidden node suffices. To design the network, we can think of the XOR function as OR with the AND case (both inputs on) ruled out. Thus the hidden layer computes AND, while the output layer computes OR but weights the output of the hidden node negatively. The network shown in Figure S18.2 does the trick.

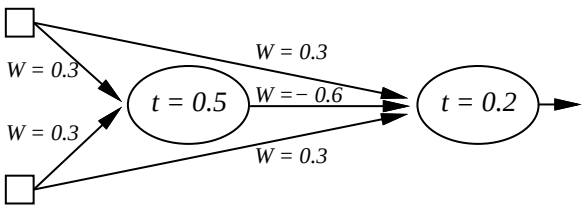


Figure S18.2 A network of step-function neurons that computes the XOR function.

18.20 According to Rojas (1996), the number of linearly separable Boolean functions with n inputs is

$$s_n = 2\sum_{i=0}^{n} \binom{2^n - 1}{i}$$

For $n \geq 2$ we have

$$s_n \leq 2(n+1)\binom{2^n - 1}{n} = 2(n+1) \cdot \frac{(2^n - 1)!}{n!(2^n - n - 1)!} \leq \frac{2(n+1)(2^n)^n}{n!} \leq 2^{n^2}$$

so the fraction of representable functions vanishes as n increases.

18.21 This question introduces some of the concepts that are studied in depth in Chapter 20; it could be used as an exercise for that chapter too, but is interesting to see at this stage also.

The logistic output is

$$p = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}} = \frac{1}{1+e^{-\sum_j w_j x_j}} \ .$$

Taking the log and differentiating, we have

$$\begin{aligned} \log p &= -\log\left(1+e^{-\mathbf{w}\cdot\mathbf{x}}\right) \\ \frac{\partial \log p}{\partial w_i} &= -\left(\frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}\frac{\partial}{\partial w_i}\left(1+e^{-\mathbf{w}\cdot\mathbf{x}}\right)\right) \\ &= -p\cdot(-x_i)\cdot e^{-\mathbf{w}\cdot\mathbf{x}} = (1-p)x_i \ . \end{aligned}$$

For a negative example, we have

$$\begin{aligned} \log(1-p) &= -\log 1/(1-p) = -\log\left(1+e^{\mathbf{w}\cdot\mathbf{x}}\right) \\ \frac{\partial \log p}{\partial w_i} &= -\left(\frac{1}{1+e^{\mathbf{w}\cdot\mathbf{x}}}\frac{\partial}{\partial w_i}\left(1+e^{\mathbf{w}\cdot\mathbf{x}}\right)\right) \\ &= -(1-p)\cdot x_i \cdot e^{\mathbf{w}\cdot\mathbf{x}} = -(1-p)\cdot x_i \cdot p/(1-p) = -p x_i \ . \end{aligned}$$

The loss function is $L = -\log p$ for a positive example ($y=1$) and $L = -\log(1-p)$ for a negative example ($y=0$). We can write this as a single rule:

$$L = -\log p^y (1-p)^{(1-y)} = -y\log p - (1-y)\log(1-p) \ .$$

Using the above results, we obtain

$$\frac{\partial L}{\partial w_i} = -y(1-p)x_i + (1-y)p x_i = -x_i(y-p) = -x_i(y - h_{\mathbf{w}}(\mathbf{x}))$$

which has the same form as the linear regression and perceptron learning rules.

18.22 This exercise reinforces the student's understanding of neural networks as mathematical functions that can be analyzed at a level of abstraction above their implementation as a network of computing elements. For simplicity, we will assume that the activation function is the same linear function at each node: $g(x) = cx + d$. (The argument is the same (only messier) if we allow different c_i and d_i for each node.)

a. The outputs of the hidden layer are

$$H_j = g\left(\sum_k w_{k,j} I_k\right) = c\sum_k w_{k,j} I_k + d$$

The final outputs are

$$O_i = g\left(\sum_j w_{j,i} H_j\right) = c\left(\sum_j w_{j,i}\left(c\sum_k w_{k,j} I_k + d\right)\right) + d$$

Now we just have to see that this is linear in the inputs:

$$O_i = c^2 \sum_k I_k \sum_j w_{k,j} w_{j,i} + d\left(1 + c\sum_j w_{j,i}\right)$$

Thus we can compute the same function as the two-layer network using just a one-layer perceptron that has weights $w_{k,i} = \sum_j w_{k,j} w_{j,i}$ and an activation function $g(x) = c^2 x + d\left(1 + c\sum_j w_{j,i}\right)$.

b. The above reduction can be used straightforwardly to reduce an n-layer network to an $(n-1)$-layer network. By induction, the n-layer network can be reduced to a single-layer network. Thus, linear activation function restrict neural networks to represent only linearly functions.

c. The original network with n input and outout nodes and h hidden nodes has $2hn$ weights, whereas the "reduced" network has n^2 weights. When $h \ll n$, the original network has far fewer weights and thus represents the i/o mapping more concisely. Such networks are known to learn much faster than the reduced network; so the idea of using linear activation functions is not without merit.

18.23 This question is especially important for students who are not expected to implement or use a neural network system. Together with 20.15 and 20.17, it gives the student a concrete (if slender) grasp of what the network actually does. Many other similar questions can be devised.

Intuitively, the data suggest that a probabilistic prediction $P(Output = 1) = 0.8$ is appropriate. The network will adjust its weights to minimize the error function. The error is

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2 = \frac{1}{2}[80(1-a_1)^2 + 20(0-a_1)^2] = 50O_1^2 - 80O_1 + 50$$

The derivative of the error with respect to the single output a_1 is

$$\frac{\partial E}{\partial a_1} = 100a_1 - 80$$

Setting the derivative to zero, we find that indeed $a_1 = 0.8$. The student should spot the connection to Ex. 18.8.

18.24 This is just a simple example of the general cross-validation model-selection method described in the chapter. For each possible size of hidden layer up to some reasonable bound, the k-fold cross-validation score is obtained given the existing training data and the best hidden layer size is chosen. This can be done using the AIMA code or with any of several public-domain machine learning toolboxes such as WEKA.

18.25 The main purpose of this exercise is to make concrete the notion of the *capacity* of a function class (in this case, linear halfspaces). It can be hard to internalize this concept, but the examples really help.

a. Three points in general position on a plane form a triangle. Any subset of the points can be separated from the rest by a line, as can be seen from the two examples in Figure S18.3(a).

b. Figure S18.3(b) shows two cases where the positive and negative examples cannot be separated by a line.

c. Four points in general position on a plane form a tetrahedron. Any subset of the points can be separated from the rest by a plane, as can be seen from the two examples in Figure S18.3(c).

d. Figure S18.3(d) shows a case where a negative point is inside the tetrahedron formed by four positive points; clearly no plane can separate the two sets.

e. Proof omitted.

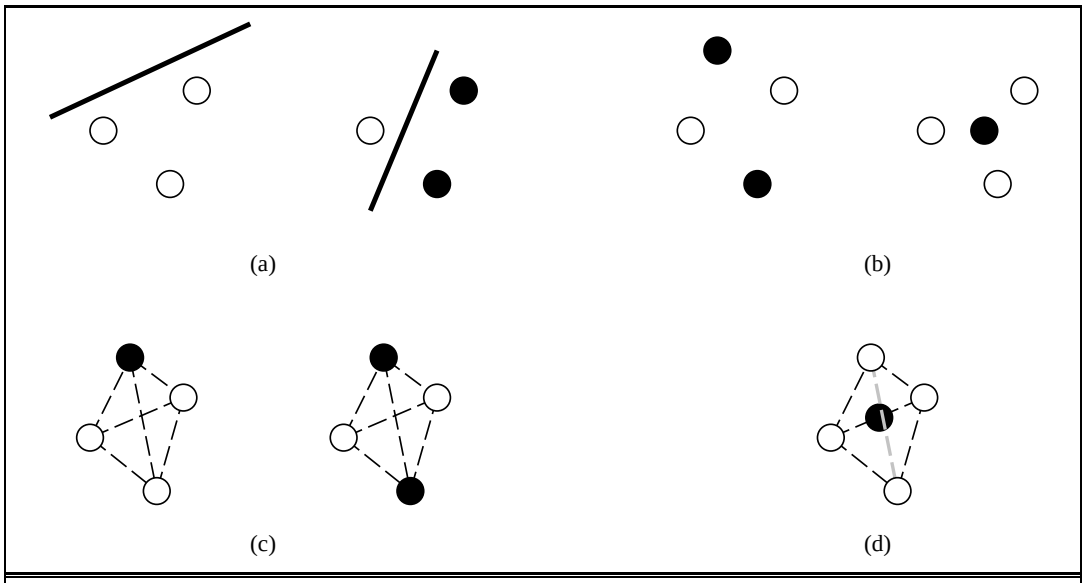


Figure S18.3 Illustrative examples for VC dimensions.

Solutions for Chapter 19
Knowledge in Learning

19.1 In CNF, the premises are as follows:

$\neg Nationality(x, n) \vee \neg Nationality(y, n) \vee \neg Language(x, l) \vee Language(y, l)$
$Nationality(Fernando, Brazil)$
$Language(Fernando, Portuguese)$

We can prove the desired conclusion directly rather than by refutation. Resolve the first two premises with $\{x/Fernando\}$ to obtain

$$\neg Nationality(y, Brazil) \vee \neg Language(Fernando, l) \vee Language(y, l)$$

Resolve this with $Language(Fernando, Portuguese)$ to obtain

$$\neg Nationality(y, Brazil) \vee Language(y, Portuguese)$$

which is the desired conclusion $Nationality(y, Brazil) \Rightarrow Language(y, Portuguese)$.

19.2 This question is tricky in places. It is important to see the distinction between the shared and unshared variables on the LHS and RHS of the determination. The shared variables will be instantiated to the objects to be compared in an analogical inference, while the unshared variables are instantiated with the objects' observed and inferred properties.

a. Here the objects being reasoned about are coins, and design, denomination, and mass are properties of coins. So we have

$$Coin(c) \Rightarrow (Design(c, d) \wedge Denomination(c, a) \succ Mass(c, m))$$

This is (very nearly exactly) true because coins of a given denomination and design are stamped from the same original die using the same material; size and shape determine volume; and volume and material determine mass.

b. Here we have to be careful. The objects being reasoned about are not programs but *runs of a given program*. (This determination is also one often forgotten by novice programmers.) We can use situation calculus to refer to the runs:

$$\forall p \;\; Input(p, i, s) \succ Output(p, o, s)$$

Here the $\forall p$ captures the p variable so that it does not participate in the determination as one of the shared or unshared variables. The situation is the shared variable. The determination expands out to the following Horn clause:

$$Input(p, i, s_1) \wedge Input(p, i, s_2) \wedge Output(p, o, s_1) \Rightarrow Output(p, o, s_2)$$

That is, if p has the same input in two different situations it will have the same output in those situations. This is generally true because computers operate on programs and inputs deterministically; however, it is important that "input" include the entire state of the computer's memory, file system, and so on. Notice that the "naive" choice

$$Input(p, i) \succ Output(p, o)$$

expands out to

$$Input(p_1, i) \wedge Input(p_2, i) \wedge Output(p_1, o) \Rightarrow Output(p_2, o)$$

which says that if any two programs have the same input they produce the same output!

c. Here the objects being reasoned are people in specific time intervals. (The intervals could be the same in each case, or different but of the same kind such as days, weeks, etc. We will stick to the same interval for simplicity. As above, we need to quantify the interval to "precapture" the variable.) We will use $Climate(x, c, i)$ to mean that person x experiences climate c in interval i, and we will assume for the sake of variety that a person's metabolism is constant.

$$\forall i \ \ Climate(x, c, i) \wedge Diet(x, d, i) \wedge Exercise(x, e, i) \wedge Metabolism(x, m) \succ Gain(x, w, i)$$

While the determinations seems plausible, it leaves out such factors as water intake, clothing, disease, etc. The qualification problem arises with determinations just as with implications.

d. Let $Baldness(x, b)$ mean that person x has baldness b (which might be $Bald$, $Partial$, or $Hairy$, say). A first stab at the determination might be

$$Mother(m, x) \wedge Father(g, m) \wedge Baldness(g, b) \succ Baldness(x, b)$$

but this would only allow an inference when two people have the same mother and maternal grandfather because the m and g are the unshared variables on the LHS. Also, the RHS has no unshared variable. Notice that the determination does not say specifically that baldness is inherited without modification; it allows, for example, for a hypothetical world in which the maternal grandchildren of a bald man are all hairy, or vice versa. This might not seem particularly natural, but consider other determinations such as "Whether or not I file a tax return determines whether or not my spouse must file a tax return."

The baldness of the maternal grandfather is the relevant value for prediction, so that should be the unshared variable on the LHS. The mother and maternal grandfather are designated by skolem functions:

$$Mother(M(x), x) \wedge Father(F(M(x)), M(x)) \wedge Baldness(F(M(x)), b_1) \succ Baldness(x, b_2)$$

If we use $Father$ and $Mother$ as function symbols, then the meaning becomes clearer:

$$Baldness(Father(Mother(x)), b_1) \succ Baldness(x, b_2)$$

Just to check, this expands into

$$Baldness(Father(Mother(x)), b_1) \wedge Baldness(Father(Mother(y)), b_1) \wedge Baldness(x, b_2) \Rightarrow Baldness(y, b_2)$$

which has the intended meaning.

19.3 Because of the qualification problem, it is not usually possible in most real-world applications to list on the LHS of a determination *all* the relevant factors that determine the RHS. Determinations will usually therefore be true to an extent—that is, if two objects agree on the LHS there is some probability (preferably greater than the prior) that the two objects will agree on the RHS. An appropriate definition for probabilistic determinations simply includes this conditional probability of matching on the RHS given a match on the LHS. For example, we could define $Nationality(x, n) \succ Language(x, l)(0.90)$ to mean that if two people have the same nationality, then there is a 90% chance that they have the same language.

19.4 This exercise test the student's understanding of resolution and unification, as well as stressing the nondeterminism of the inverse resolution process. It should help a lot in making the inverse resolution operation less mysterious and more amenable to mathematical analysis. It is helpful first to draw out the resolution "V" when doing these problems, and then to do a careful case analysis.

a. There is no possible value for C_2 here. The resolution step would have to resolve away both the $P(x, y)$ on the LHS of C_1 and the $Q(x, y)$ on the right, which is not possible. (Resolution *can* remove more than one literal from a clause, but only if those literals are redundant—i.e., one subsumes the other.)

b. Without loss of generality, let C_1 contain the negative (LHS) literal to be resolved away. The LHS of C_1 therefore contains one literal l, while the LHS of C_2 must be empty. The RHS of C_2 must contain l' such that l and l' unify with some unifier θ. Now we have a choice: $P(A, B)$ on the RHS of C could come from the RHS of C_1 or of C_2. Thus the two basic solution templates are

$$C_1 = l \;\Rightarrow\; False \;\;;\;\; C_2 = True \;\Rightarrow\; l' \vee P(A, B)\theta^{-1}$$
$$C_1 = l \;\Rightarrow\; P(A, B)\theta^{-1} \;\;;\;\; C_2 = True \;\Rightarrow\; l'$$

Within these templates, the choice of l is entirely unconstrained. Suppose l is $Q(x, y)$ and l' is $Q(A, B)$. Then $P(A, B)\theta^{-1}$ could be $P(x, y)$ (or $P(A, y)$ or $P(x, B)$) and the solutions are

$$C_1 = Q(x, y) \;\Rightarrow\; False \;\;;\;\; C_2 = True \;\Rightarrow\; Q(A, B) \vee P(x, y)$$
$$C_1 = Q(x, y) \;\Rightarrow\; P(x, y) \;\;;\;\; C_2 = True \;\Rightarrow\; Q(A, B)$$

c. As before, let C_1 contain the negative (LHS) literal to be resolved away, with l' on the RHS of C_2. We now have four possible templates because each of the two literals in C could have come from either C_1 or C_2:

$$C_1 = l \;\Rightarrow\; False \;\;;\;\; C_2 = P(x, y)\theta^{-1} \;\Rightarrow\; l' \vee P(x, f(y))\theta^{-1}$$
$$C_1 = l \;\Rightarrow\; P(x, f(y))\theta^{-1} \;\;;\;\; C_2 = P(x, y)\theta^{-1} \;\Rightarrow\; l'$$
$$C_1 = l \wedge P(x, y)\theta^{-1} \;\Rightarrow\; False \;\;;\;\; C_2 = True \;\Rightarrow\; l' \vee P(x, f(y))\theta^{-1}$$
$$C_1 = l \wedge P(x, y)\theta^{-1} \;\Rightarrow\; P(x, f(y))\theta^{-1} \;\;;\;\; C_2 = True \;\Rightarrow\; l'$$

Again, we have a fairly free choice for l. However, since C contains x and y, θ cannot bind those variables (else they would not appear in C). Thus, if l is $Q(x, y)$, then l' must be $Q(x, y)$ also and θ will be empty.

19.5 We will assume that Prolog is the logic programming language. It is certainly true that any solution returned by the call to $Resolve$ will be a correct inverse resolvent. Unfortunately, it is quite possible that the call will fail to return because of Prolog's depth-first search. If the clauses in $Resolve$ and $Unify$ are infelicitously arranged, the proof tree might go down the branch corresponding to indefinitely nested function symbols in the solution and never return. This can be alleviated by redesigning the Prolog inference engine so that it works using breadth-first search or iterative deepening, although the infinitely deep branches will still be a problem. Note that any cuts used in the Prolog program will also be a problem for the inverse resolution.

19.6 This exercise gives some idea of the rather large branching factor facing top-down ILP systems.

a. It is important to note that position is significant—$P(A, B)$ is very different from $P(B, A)$! The first argument position can contain one of the five existing variables or a new variable. For each of these six choices, the second position can contain one of the five existing variables or a new variable, *except* that the literal with two new variables is disallowed. Hence there are 35 choices. With negated literals too, the total branching factor is 70.

b. This seems to be quite a tricky combinatorial problem. The easiest way to solve it seems to be to start by including the multiple possibilities that are equivalent under renaming of the new variables as well as those that contain only new variables. Then these redundant or illegal choices can be removed later. Now, we can use up to $r - 1$ new variables. If we use $\leq i$ new variables, we can write $(n + i)^r$ literals, so using exactly $i > 0$ variables we can write $(n + i)^r - (n + i - 1)^r$ literals. Each of these is functionally isomorphic under any renaming of the new variables. With i variables, there are are i renamings. Hence the total number of distinct literals (including those illegal ones with no old variables) is

$$n^r + \sum_{i=1}^{r-1} \frac{(n + i)^r - (n + i - 1)^r}{i!}$$

Now we just subtract off the number of distinct all-new literals. With $\leq i$ new variables, the number of (not necessarily distinct) all-new literals is i^r, so the number with exactly $i > 0$ is $i^r - (i - 1)^r$. Each of these has $i!$ equivalent literals in the set. This gives us the final total for distinct, legal literals:

$$n^r + \sum_{i=1}^{r-1} \frac{(n + i)^r - (n + i - 1)^r}{i!} - \sum_{i=1}^{r-1} \frac{i^r - (i - 1)^r}{i!}$$

which can doubtless be simplified. One can check that for $r = 2$ and $n = 5$ this gives 35.

c. If a literal contains only new variables, then either a subsequent literal in the clause body connects one or more of those variables to one or more of the "old" variables, or it doesn't. If it does, then the same clause will be generated with those two literals reversed, such that the restriction is not violated. If it doesn't, then the literal is either always true (if the predicate is satisfiable) or always false (if it is unsatisfiable), independent of the "input" variables in the head. Thus, the literal would either be redundant or would render the clause body equivalent to $False$.

19.7 FOIL is available on the web at http://www-2.cs.cmu.edu/afs/cs/project/ai-repository-/ai/areas/learning/systems/foil/0.html (and possibly other places). It is worthwhile to experiment with it.

Solutions for Chapter 20
Learning Probabilistic Models

20.1 The code for this exercise is a straightforward implementation of Equations 20.1 and 20.2. Figure S20.1 shows the results for data sequences generated from h_3 and h_4. (Plots for h_1 and h_2 are essentially identical to those for h_5 and h_4.) Results obtained by students may vary because the data sequences are generated randomly from the specified candy distribution. In (a), the samples very closely reflect the true probabilities and the hypotheses other than h_3 are effectively ruled out very quickly. In (c), the early sample proportions are somewhere between 50/50 and 25/75; furthermore, h_3 has a higher prior than h_4. As a result, h_3 and h_4 vie for supremacy. Between 50 and 60 samples, a preponderance of limes ensures the defeat of h_3 and the prediction quickly converges to 0.75.

20.2 This is a nontrivial sequential decision problem, but can be solved using the tools developed in the book. It leads into general issues of statistical decision theory, stopping rules, etc. Here, we sketch the "straightforward" solution.

We can think of this problem as a simplified form of POMDP (see Chapter 17). The "belief states" are defined by the numbers of cherry and lime candies observed so far in the sampling process. Let these be C and L, and let $U(C, L)$ be the utility of the corresponding belief state. In any given state, there are two possible decisions: $sell$ and $sample$. There is a simple Bellman equation relating Q and U for the sampling case:

$$Q(C, L, sample) = P(cherry|C, L)U(C + 1, L) + P(lime|C, L)U(C, L + 1)$$

Let the posterior probability of each h_i be $P(h_i|C, L)$, the size of the bag be N, and the fraction of cherries in a bag of type i be f_i. Then the value obtained by selling is given by the value of the sampled candies (which Ann gets to keep) plus the price paid by Bob (which equals the expected utility of the remaining candies for Bob):

$$Q(C, L, sell) = Cc_A + L\ell_A + \sum_i P(h_i|C, L)[(f_iN - C)c_B + ((1 - f_i)N - L)\ell_B]$$

and of course we have

$$U(C, L) = \max\{Q(C, L, sell), Q(C, L, sample)\} .$$

Thus we can set up a dynamic program to compute Q given the obvious boundary conditions for the case where $C+L=N$. The solution of this dynamic program gives the optimal policy for Ann. It will have the property that if she should sell at (C, L), then she should also sell at $(C, L + k)$ for all positive k. Thus, the problem is to determine, for each C, the threshold

value of L at or above which she should sell. A minor complication is that the formula for $P(h_i|C, L)$ should take into account the non-replacement of candies and the finiteness of N, otherwise odd things will happen when $C + L$ is close to N.

20.3 The Bayesian approach would be to take both drugs. The maximum likelihood approach would be to take the anti-B drug. In the case where there are two versions of B, the Bayesian still recommends taking both drugs, while the maximum likelihood approach is now to take the anti-A drug, since it has a 40% chance of being correct, versus 30% for each of the B cases. This is of course a caricature, and you would be hard-pressed to find a doctor, even a rabid maximum-likelihood advocate who would prescribe like this. But you can find ones who do research like this.

20.4 Boosted naive Bayes learning is discussed by Elkan (1997). The application of boosting to naive Bayes is straightforward. The naive Bayes learner uses maximum-likelihood

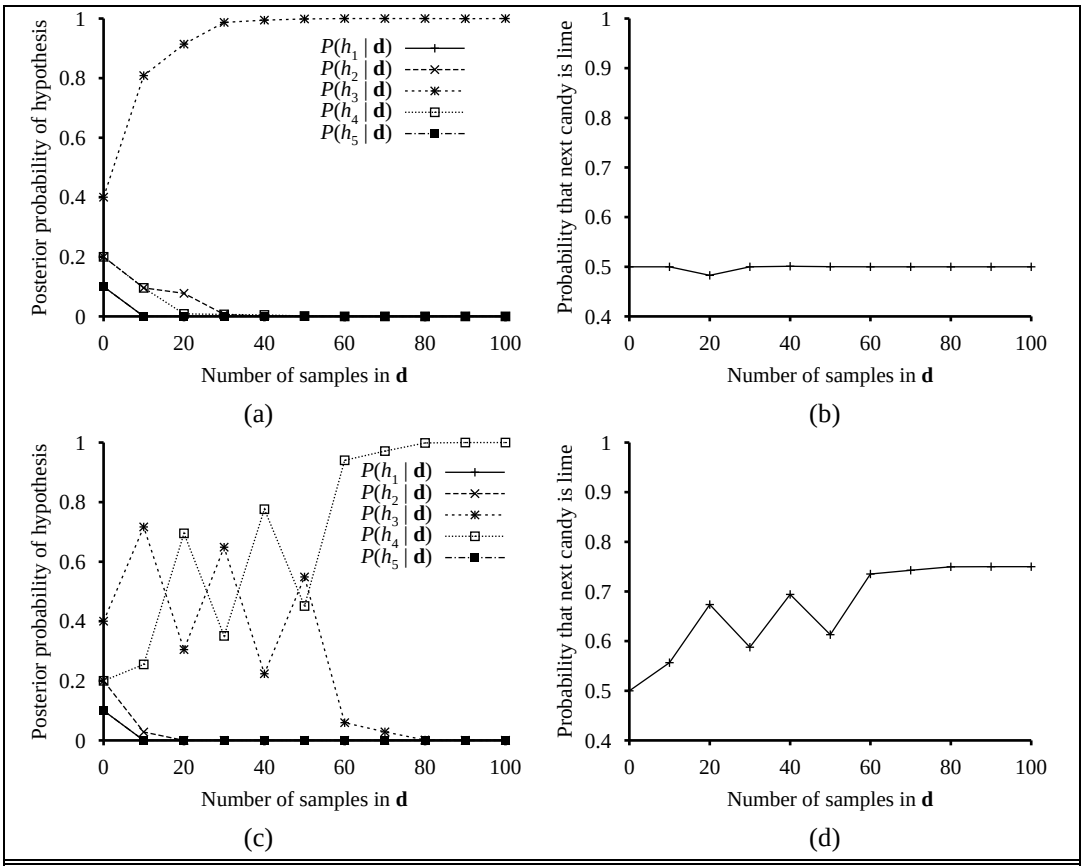


Figure S20.1 Graphs for Ex. 20.1. (a) Posterior probabilities $P(h_i|d_1, \ldots, d_N)$ over a sample sequence of length 100 generated from h_3 (50% cherry + 50% lime). (b) Bayesian prediction $P(d_{N+1} = lime|d_1, \ldots, d_N)$ given the data in (a). (c) Posterior probabilities $P(h_i|d_1, \ldots, d_N)$ over a sample sequence of length 100 generated from h_4 (25% cherry + 75% lime). (d) Bayesian prediction $P(d_{N+1} = lime|d_1, \ldots, d_N)$ given the data in (c).

parameter estimation based on counts, so using a weighted training set simply means adding weights rather than counting. Each naive Bayes model is treated as a deterministic classifier that picks the most likely class for each example.

20.5 We have

$$L = -m(\log\sigma + \log\sqrt{2\pi}) - \sum_j \frac{(y_j - (\theta_1 x_j + \theta_2))^2}{2\sigma^2}$$

hence the equations for the derivatives at the optimum are

$$\begin{aligned}
\frac{\partial L}{\partial \theta_1} &= -\sum_j \frac{x_j(y_j - (\theta_1 x_j + \theta_2))}{\sigma^2} = 0 \\
\frac{\partial L}{\partial \theta_2} &= -\sum_j \frac{(y_j - (\theta_1 x_j + \theta_2))}{\sigma^2} = 0 \\
\frac{\partial L}{\partial \sigma} &= -\frac{m}{\sigma} + \sum_j \frac{(y_j - (\theta_1 x_j + \theta_2))^2}{\sigma^3} = 0
\end{aligned}$$

and the solutions can be computed as

$$\begin{aligned}
\theta_1 &= \frac{m\left(\sum_j x_j y_j\right) - \left(\sum_j y_j\right)\left(\sum_j x_j\right)}{m\left(\sum_j x_j^2\right) - \left(\sum_j x_j\right)^2} \\
\theta_2 &= \frac{1}{m}\sum_j (y_j - \theta_1 x_j) \\
\sigma^2 &= \frac{1}{m}\sum_j (y_j - (\theta_1 x_j + \theta_2))^2
\end{aligned}$$

20.6 There are a couple of ways to solve this problem. Here, we show the indicator variable method described on page 743. Assume we have a child variable Y with parents $X_1, \ldots, X_k$ and let the range of each variable be $\{0, 1\}$. Let the noisy-OR parameters be $q_i = P(Y = 0 | X_i = 1, X_{-i} = 0)$. The noisy-OR model then asserts that

$$P(Y = 1 | x_1, \ldots, x_k) = 1 - \prod_{i=1}^{k} q_i^{x_i} \ .$$

Assume we have m complete-data samples with values y_j for Y and x_{ij} for each X_i. The conditional log likelihood for $P(Y|X_1, \ldots, X_k)$ is given by

$$\begin{aligned}
L &= \sum_j \log\left(1 - \prod_i q_i^{x_i j}\right)^{y_j} \left(\prod_i q_i^{x_i j}\right)^{1-y_j} \\
&= \sum_j y_j \log\left(1 - \prod_i q_i^{x_i j}\right) + (1 - y_j)\sum_i x_{ij} \log q_i
\end{aligned}$$

The gradient with respect to each noisy-OR parameter is

$$\begin{aligned}\frac{\partial L}{\partial q_i} &= \sum_j -\frac{y_j x_{ij} \prod_i q_i^{x_i j}}{q_i \left(1 - \prod_i q_i^{x_i j}\right)} + \frac{(1-y_j)x_{ij}}{q_i} \\ &= \sum_j \frac{x_{ij}\left(1 - y_j - \prod_i q_i^{x_i j}\right)}{q_i \left(1 - \prod_i q_i^{x_i j}\right)}\end{aligned}$$

20.7

a. By integrating over the range $[0, 1]$, show that the normalization constant for the distribution $\text{beta}[a, b]$ is given by $\alpha = \Gamma(a+b)/\Gamma(a)\Gamma(b)$ where $\Gamma(x)$ is the **Gamma function**, defined by $\Gamma(x+1) = x \cdot \Gamma(x)$ and $\Gamma(1) = 1$. (For integer x, $\Gamma(x+1) = x!$.)

GAMMA FUNCTION

We will solve this for positive integer a and b by induction over a. Let $\alpha(a, b)$ be the normalization constant. For the base cases, we have

$$\alpha(1, b) = 1/\int_0^1 \theta^0 (1-\theta)^{b-1} d\theta = -1/[\frac{1}{b}(1-\theta)^b]_0^1 = b$$

and

$$\frac{\Gamma(1+b)}{\Gamma(1)\Gamma(b)} = \frac{b \cdot \Gamma(b)}{1 \cdot \Gamma(b)} = b\,.$$

For the inductive step, we assume for all b that

$$\alpha(a-1, b+1) = \frac{\Gamma(a+b)}{\Gamma(a-1)\Gamma(b+1)} = \frac{a-1}{b} \cdot \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$$

Now we evaluate $\alpha(a, b)$ using integration by parts. We have

$$\begin{aligned}1/\alpha(a, b) &= \int_0^1 \theta^{a-1}(1-\theta)^{b-1} d\theta \\ &= [\theta^{a-1} \cdot \frac{1}{b}(1-\theta)^b]_1^0 + \frac{a-1}{b}\int_0^1 \theta^{a-2}(1-\theta)^b d\theta \\ &= 0 + \frac{a-1}{b}\frac{1}{\alpha(a-1, b+1)}\end{aligned}$$

Hence

$$\alpha(a, b) = \frac{b}{a-1}\alpha(a-1, b+1) = \frac{b}{a-1}\frac{a-1}{b} \cdot \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$$

as required.

b. The mean is given by the following integral:

$$\begin{aligned}\mu(a, b) &= \alpha(a, b)\int_0^1 \theta \cdot \theta^{a-1}(1-\theta)^{b-1} d\theta \\ &= \alpha(a, b)\int_0^1 \theta^a (1-\theta)^{b-1} d\theta \\ &= \alpha(a, b)/\alpha(a+1, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \cdot \frac{\Gamma(a+1)\Gamma(b)}{\Gamma(a+b+1)}\end{aligned}$$

$$= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \cdot \frac{a\Gamma(a)\Gamma(b)}{(a+b)\Gamma(a+b+1)} = \frac{a}{a+b} .$$

c. The mode is found by solving for $d\mathrm{beta}[a,b](\theta)/d\theta = 0$:

$$\begin{aligned} &\frac{d}{d\theta}(\alpha(a,b)\theta^{a-1}(1-\theta)^{b-1}) \\ &\quad = \quad \alpha(a,b)[(a-1)\theta^{a-2}(1-\theta)^{b-1} - (b-1)\theta^{a-1}(1-\theta)^{b-2}] = 0 \\ &\quad \Rightarrow \quad (a-1)(1-\theta) = (b-1)\theta \\ &\quad \Rightarrow \quad \theta = \frac{a-1}{a+b-2} \end{aligned}$$

d. $\mathrm{beta}[\epsilon,\epsilon] = \alpha(\epsilon,\epsilon)\theta^{\epsilon-1}(1-\theta)^{\epsilon-1}$ tends to very large values close to $\theta = 0$ and $\theta = 1$, i.e., it expresses the prior belief that the distribution characterized by θ is nearly deterministic (either positively or negatively). After updating with a positive example we obtain the distribution $\mathrm{beta}[1+\epsilon,\epsilon]$, which has nearly all its mass near $\theta = 1$ (and the converse for a negative example), i.e., we have learned that the distribution characterized by θ is deterministic in the positive sense. If we see a "counterexample", e.g., a positive and a negative example, we obtain $\mathrm{beta}[1+\epsilon,1+\epsilon]$, which is close to uniform, i.e., the hypothesis of near-determinism is abandoned.

20.8 Consider the maximum-likelihood parameter values for the CPT of node Y in the original network, where an extra parent X_{k+1} will be added to Y. If we set the parameters for $P(y|x_1,\ldots,x_k,x_{k+1})$ in the new network to be identical to $P(y|x_1,\ldots,x_k)$ in the original netowrk, regardless of the value x_{k+1}, then the likelihood of the data is unchanged. Maximizing the likelihood by altering the parameters can then only *increase* the likelihood.

20.9

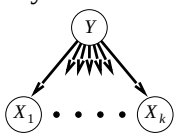
a. The probability of a positive example is π and of a negative example is $(1-\pi)$, and the data are independent, so the probability of the data is $\pi^p(1-\pi)^n$

b. We have $L = p\log\pi + n\log(1-\pi)$; if the derivative is zero, we have

$$\frac{\partial L}{\partial \pi} = \frac{p}{\pi} - \frac{n}{1-\pi} = 0$$

so the ML value is $\pi = p/(p+n)$, i.e., the proportion of positive examples in the data.

c. This is the "naive Bayes" probability model.



d. The likelihood of a single instance is a product of terms. For a positive example, π times α_i for each true attribute and $(1-\alpha_i)$ for each negative attribute; for a negative example, $(1-\pi)$ times β_i for each true attribute and $(1-\beta_i)$ for each negative attribute. Over the whole data set, the likelihood is $\pi^p(1-\pi)^n \prod_i \alpha_i^{p_i^+}(1-\alpha_i)^{n_i^+}\beta_i^{p_i^-}(1-\beta_i)^{n_i^-}$.

e. The log likelihood is

$L = p\log\pi + n\log(1-\pi) + \sum_i p_i^+ \log\alpha_i + n_i^+ \log(1-\alpha_i) + p_i^- \log\beta_i + n_i^- \log(1-\beta_i)$.
Setting the derivatives w.r.t. α_i and β_i to zero, we have

$$\frac{\partial L}{\partial \alpha_i} = \frac{p_i^+}{\alpha_i} - \frac{n_i^+}{1-\alpha_i} = 0 \quad \text{and} \quad \frac{\partial L}{\partial \beta_i} = \frac{p_i^-}{\beta_i} - \frac{n_i^-}{1-\beta_i} = 0$$

giving $\alpha_i = p_i^+/(p_i^+ + n_i^+)$, i.e., the fraction of cases where X_i is true given Y is true, and $\beta_i = p_i^-/(p_i^- + n_i^-)$, i.e., the fraction of cases where X_i is true given Y is false.

f. In the data set we have $p = 2$, $n = 2$, $p_i^+ = 1$, $n_i^+ = 1$, $p_i^- = 1$, $n_i^- = 1$. From our formulæ, we obtain $\pi = \alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0.5$.

g. Each example is predicted to be positive with probability 0.5.

20.10

a. Consider the ideal case in which the bags were infinitely large so there is no statistical fluctuation in the sample. With two attributes (say, $Flavor$ and $Wrapper$), we have five unknowns: θ gives the the relative sizes of the bags, θ_{F1} and θ_{F2} give the proportion of cherry candies in each bag, and θ_{W1} and θ_{W2} give the proportion of red wrappers in each bag. In the data, we observe just the flavor and wrapper for each candy; there are four combinations, so three independent numbers can be obtained. This is not enough to recover five unknowns. With three attributes, there are eight combinations and seven numbers can be obtained, enough to recover the seven parameters.

b. The computation for $\theta^{(1)}$ has eight nearly identical expressions and calculations, one of which is shown. The symbolic expression for $\theta_{F1}^{(1)}$ is shown, but not its evaluation; it would be reasonable to ask students to write out the expression in terms of the parameters, as was done for $\theta^{(1)}$, and calculate the value. The final answers are given in the chapter.

c. Consider the contribution to the update for θ from the 273 red-wrapped cherry candies with holes:

$$\frac{273}{1000} \cdot \frac{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)}}{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)} + \theta_{F2}^{(0)}\theta_{W2}^{(0)}\theta_{H2}^{(0)}(1-\theta^{(0)})}$$

If all of the seven named parameters have value p, this reduces to

$$\frac{273}{1000} \cdot \frac{p^4}{p^4 + p^3(1-p)} = \frac{273p}{1000}$$

with similar results for the other candy categories. Thus, the new value for $theta^{(1)}$ just ends up being $1000p/1000 = p$.

We can check the expression for θ_{F1} too; for example, the 273 red-wrapped cherry candies with holes contribute an expected count of

$$\begin{aligned} &273P(Bag = 1 \mid Flavor_j = cherry, Wrapper = red, Holes = 1) \\ &= 273\frac{\theta_{F1}\theta_{W1}\theta_{H1}\theta}{\theta_{F1}\theta_{W1}\theta_{H1}\theta + \theta_{F2}\theta_{W2}\theta_{H2}(1-\theta)} = 273p \end{aligned}$$

and the 90 green-wrapped cherry candies with no holes contribute an expected count of

$$\begin{aligned} & 90P(Bag = 1 \mid Flavor_j = cherry, Wrapper = green, Holes = 0) \\ = \; & 90\frac{\theta_{F1}(1-\theta_{W1})(1-\theta_{H1})\theta}{\theta_{F1}(1-\theta_{W1})(1-\theta_{H1})\theta + \theta_{F2}(1-\theta_{W2})(1-\theta_{H2})(1-\theta)} \\ = \; & 90p^2(1-p)^2/p(1-p)^2 = 90p \; . \end{aligned}$$

Continuing, we find that the new value for θ_{F1} is $560p/1000p = 0.56$, the proportion of cherry candies in the entire sample.

For θ_{F2}, the 273 red-wrapped cherry candies with holes contribute an expected count of

$$\begin{aligned} & 273P(Bag = 2 \mid Flavor_j = cherry, Wrapper = red, Holes = 1) \\ = \; & 273\frac{\theta_{F2}\theta_{W2}\theta_{H2}(1-\theta)}{\theta_{F1}\theta_{W1}\theta_{H1}\theta + \theta_{F2}\theta_{W2}\theta_{H2}(1-\theta)} = 273(1-p) \end{aligned}$$

with similar contributions from the other cherry categories, so the new value is $560(1-p)/1000(1-p) = 0.56$, as for θ_{F1}. Similarly, $\theta^{(1)}_{W1} = \theta^{(1)}_{W2} = 0.545$, the proportion of red wrappers in the sample, and $\theta^{(1)}_{H1} = \theta^{(1)}_{H2} = 0.550$, the proportion of candies with holes in the sample.

Intuitively, this makes sense: because the bag label is invisible, labels 1 and 2 are *a priori* indistinguishable; initializing all the conditional parameters to the same value (regardless of the bag) provides no means of breaking the symmetry. Thus, the symmetry remains.

On the next iteration, we no longer have all the parameters set to p, but we do know that, for example,

$$\theta_{F1}\theta_{W1}\theta_{H1} = \theta_{F2}\theta_{W2}\theta_{H2}$$

so those terms cancel top and bottom in the expression for the contribution of the 273 candies to θ_{F1}, and once again the contribution is $273p$.

To cut a long story short, all the parameters remain fixed after the first iteration, with θ at its initial value p and the other parameters at the corresponding empirical frequencies as indicated above.

d. This part takes some time but makes the abstract mathematical expressions in the chapter very concrete! The one concession to abstraction will be the use of symbols for the empirical counts, e.g.,

$$N_{cr1} = N(Flavor = cherry, Wrapper = red, Holes = 1) = 273 \; .$$

with marginal counts N_c, N_{r1}, etc. Thus we have $\theta^{(1)}_{F1} = N_c/N = 560/1000$.

The log likelihood is given by

$$\begin{aligned} L(\mathbf{d}) \; & = \; \log P(\mathbf{d}) = \log \prod_j P(d_j) = \sum_j \log P(d_j) \\ & = \; N_{cr1} \log P(F = cherry, W = red, H = 1) + \\ & \quad \; N_{lr1} \log P(F = lime, W = red, H = 1) + \end{aligned}$$

$$
\begin{aligned}
&N_{cr0} \log P(F = cherry, W = red, H = 0) + \\
&N_{lr0} \log P(F = lime, W = red, H = 0) + \\
&N_{cg1} \log P(F = cherry, W = green, H = 1) + \\
&N_{lg1} \log P(F = lime, W = green, H = 1) + \\
&N_{cg0} \log P(F = cherry, W = green, H = 0) + \\
&N_{lg0} \log P(F = lime, W = green, H = 0)
\end{aligned}
$$

Each of these probabilities can be expressed in terms of the network parameters, giving the following expression for $L(\mathbf{d})$:

$$
\begin{aligned}
&N_{cr1} \log(\theta_{F1}\theta_{W1}\theta_{H1}\theta + \theta_{F2}\theta_{W2}\theta_{H2}(1-\theta)) + \\
&N_{lr1} \log((1-\theta_{F1})\theta_{W1}\theta_{H1}\theta + (1-\theta_{F2})\theta_{W2}\theta_{H2}(1-\theta)) + \\
&N_{cr0} \log(\theta_{F1}\theta_{W1}(1-\theta_{H1})\theta + \theta_{F2}\theta_{W2}(1-\theta_{H2})(1-\theta)) + \\
&N_{lr0} \log((1-\theta_{F1})\theta_{W1}(1-\theta_{H1})\theta + (1-\theta_{F2})\theta_{W2}(1-\theta_{H2})(1-\theta)) + \\
&N_{cg1} \log(\theta_{F1}(1-\theta_{W1})\theta_{H1}\theta + \theta_{F2}(1-\theta_{W2})\theta_{H2}(1-\theta)) + \\
&N_{lg1} \log((1-\theta_{F1})(1-\theta_{W1})\theta_{H1}\theta + (1-\theta_{F2})(1-\theta_{W2})\theta_{H2}(1-\theta)) + \\
&N_{cg0} \log(\theta_{F1}(1-\theta_{W1})(1-\theta_{H1})\theta + \theta_{F2}(1-\theta_{W2})(1-\theta_{H2})(1-\theta)) + \\
&N_{lg0} \log((1-\theta_{F1})(1-\theta_{W1})(1-\theta_{H1})\theta + (1-\theta_{F2})(1-\theta_{W2})(1-\theta_{H2})(1-\theta))
\end{aligned}
$$

Hence $\partial L/\partial\theta$ is given by

$$
\begin{aligned}
&N_{cr1} \frac{\theta_{F1}\theta_{W1}\theta_{H1} - \theta_{F2}\theta_{W2}\theta_{H2}}{\theta_{F1}\theta_{W1}\theta_{H1}\theta + \theta_{F2}\theta_{W2}\theta_{H2}(1-\theta)} \\
&-N_{lr1} \frac{(1-\theta_{F1})\theta_{W1}\theta_{H1} - (1-\theta_{F2})\theta_{W2}\theta_{H2}}{(1-\theta_{F1})\theta_{W1}\theta_{H1}\theta + (1-\theta_{F2})\theta_{W2}\theta_{H2}(1-\theta)} \\
&+N_{cr0} \frac{\theta_{F1}\theta_{W1}(1-\theta_{H1}) - \theta_{F2}\theta_{W2}(1-\theta_{H2})}{\theta_{F1}\theta_{W1}(1-\theta_{H1})\theta + \theta_{F2}\theta_{W2}(1-\theta_{H2})(1-\theta)} \\
&-N_{lr0} \frac{(1-\theta_{F1})\theta_{W1}(1-\theta_{H1}) - (1-\theta_{F2})\theta_{W2}(1-\theta_{H2})}{(1-\theta_{F1})\theta_{W1}(1-\theta_{H1})\theta + (1-\theta_{F2})\theta_{W2}(1-\theta_{H2})(1-\theta)} \\
&+N_{cg1} \frac{\theta_{F1}(1-\theta_{W1})\theta_{H1} - \theta_{F2}(1-\theta_{W2})\theta_{H2}}{\theta_{F1}(1-\theta_{W1})\theta_{H1}\theta + \theta_{F2}(1-\theta_{W2})\theta_{H2}(1-\theta)} \\
&-N_{lg1} \frac{(1-\theta_{F1})(1-\theta_{W1})\theta_{H1} - (1-\theta_{F2})(1-\theta_{W2})\theta_{H2}}{(1-\theta_{F1})(1-\theta_{W1})\theta_{H1}\theta + (1-\theta_{F2})(1-\theta_{W2})\theta_{H2}(1-\theta)} \\
&+N_{cg0} \frac{\theta_{F1}(1-\theta_{W1})(1-\theta_{H1}) - \theta_{F2}(1-\theta_{W2})(1-\theta_{H2})}{\theta_{F1}(1-\theta_{W1})(1-\theta_{H1})\theta + \theta_{F2}(1-\theta_{W2})(1-\theta_{H2})(1-\theta)} \\
&-N_{lg0} \frac{(1-\theta_{F1})(1-\theta_{W1})(1-\theta_{H1}) - (1-\theta_{F2})(1-\theta_{W2})(1-\theta_{H2})}{(1-\theta_{F1})(1-\theta_{W1})(1-\theta_{H1})\theta + (1-\theta_{F2})(1-\theta_{W2})(1-\theta_{H2})(1-\theta)}
\end{aligned}
$$

By inspection, we can see that whenever $\theta_{F1} = \theta_{F2}$, $\theta_{W1} = \theta_{W2}$, and $\theta_{H1} = \theta_{H2}$, the derivative is identically zero. Moreover, each term in the above expression has the form $k/f(\theta)$ where k does not contain θ and $f'(\theta)$ evaluates to zero under these conditions. Thus the second derivative $\partial^2 L/\partial\theta^2$ is a collection of terms of the form $-kf'(\theta)/(f(\theta))^2$, all of which evaluate to zero. In fact, all derivatives evaluate to

zero under these conditions, so the likelihood is completely flat with respect to θ in the subspace defined by $\theta_{F1} = \theta_{F2}$, $\theta_{W1} = \theta_{W2}$, and $\theta_{H1} = \theta_{H2}$. Another way to see this is to note that, in this subspace, the terms within the logs in the expression for $L(\mathbf{d})$ simplify to terms of the form $\phi_F\phi_W\phi_H\theta + \phi_F\phi_W\phi_H(1-\theta) = \phi_F\phi_W\phi_H$, so that the likelihood is in fact independent of θ!

A representative partial derivative $\partial L/\partial\theta_{F1}$ is given by

$$
\begin{aligned}
&N_{cr1}\,\frac{\theta_{W1}\theta_{H1}\theta}{\theta_{F1}\theta_{W1}\theta_{H1}\theta + \theta_{F2}\theta_{W2}\theta_{H2}(1-\theta)} \\
&-N_{lr1}\,\frac{\theta_{W1}\theta_{H1}\theta}{(1-\theta_{F1})\theta_{W1}\theta_{H1}\theta + (1-\theta_{F2})\theta_{W2}\theta_{H2}(1-\theta)} \\
&+N_{cr0}\,\frac{\theta_{W1}(1-\theta_{H1})\theta}{\theta_{F1}\theta_{W1}(1-\theta_{H1})\theta + \theta_{F2}\theta_{W2}(1-\theta_{H2})(1-\theta)} \\
&-N_{lr0}\,\frac{\theta_{W1}(1-\theta_{H1})\theta}{(1-\theta_{F1})\theta_{W1}(1-\theta_{H1})\theta + (1-\theta_{F2})\theta_{W2}(1-\theta_{H2})(1-\theta)} \\
&+N_{cg1}\,\frac{(1-\theta_{W1})\theta_{H1}\theta}{\theta_{F1}(1-\theta_{W1})\theta_{H1}\theta + \theta_{F2}(1-\theta_{W2})\theta_{H2}(1-\theta)} \\
&-N_{lg1}\,\frac{(1-\theta_{W1})\theta_{H1}\theta}{(1-\theta_{F1})(1-\theta_{W1})\theta_{H1}\theta + (1-\theta_{F2})(1-\theta_{W2})\theta_{H2}(1-\theta)} \\
&+N_{cg0}\,\frac{(1-\theta_{W1})(1-\theta_{H1})\theta}{\theta_{F1}(1-\theta_{W1})(1-\theta_{H1})\theta + \theta_{F2}(1-\theta_{W2})(1-\theta_{H2})(1-\theta)} \\
&-N_{lg0}\,\frac{(1-\theta_{W1})(1-\theta_{H1})\theta}{(1-\theta_{F1})(1-\theta_{W1})(1-\theta_{H1})\theta + (1-\theta_{F2})(1-\theta_{W2})(1-\theta_{H2})(1-\theta)}
\end{aligned}
$$

Unlike the previous case, here the individual terms do not evaluate to zero. Writing $\theta_{F1} = \theta_{F2} = N_c/N$, etc., the expression for $\partial L/\partial\theta_{F1}$ becomes

$$
\begin{aligned}
&N_{cr1}\,\frac{N\,N_r N_1\theta}{N_c N_r N_1\theta + N_c N_r N_1(1-\theta)} \\
&-N_{lr1}\,\frac{N\,N_r N_1\theta}{(N-N_c)N_r N_1\theta + (N-N_c)N_r N_1(1-\theta)} \\
&+N_{cr0}\,\frac{N\,N_r(N-N_1)\theta}{N_c N_r(N-N_1)\theta + N_c N_r(N-N_1)(1-\theta)} \\
&-N_{lr0}\,\frac{N\,N_r(N-N_1)\theta}{(N-N_c)N_r(N-N_1)\theta + (N-N_c)N_r(N-N_1)(1-\theta)} \\
&+N_{cg1}\,\frac{N\,(N-N_r)N_1\theta}{N_c(N-N_r)N_1\theta + N_c(N-N_r)N_1(1-\theta)} \\
&-N_{lg1}\,\frac{N\,(N-N_r)N_1\theta}{(N-N_c)(N-N_r)N_1\theta + (N-N_c)(N-N_r)N_1(1-\theta)} \\
&+N_{cg0}\,\frac{N\,(N-N_r)(N-N_1)\theta}{N_c(N-N_r)(N-N_1)\theta + N_c(N-N_r)(N-N_1)(1-\theta)} \\
&-N_{lg0}\,\frac{N\,(N-N_r)(N-N_1)\theta}{(N-N_c)(N-N_r)(N-N_1)\theta + (N-N_c)(N-N_r)(N-N_1)(1-\theta)}
\end{aligned}
$$

This in turn simplifies to

$$\begin{aligned}\frac{\partial L}{\partial \theta_{F1}} &= \frac{(N_{cr1}+N_{cr0}+N_{cg1}+N_{cg0})N\theta}{N_c} - \frac{(N_{lr1}+N_{lr0}+N_{lg1}+N_{lg0})N\theta}{N-N_c} \\ &= \frac{N_c N\theta}{N_c} - \frac{(N-N_c)N\theta}{N-N_c} = 0\,.\end{aligned}$$

Thus, we have a stationary point as expected.

To identify the nature of the stationary point, we need to examine the second derivatives. We will not do this exhaustively, but will note that

$$\begin{aligned}\partial^2 L/\partial\theta_{F1}^2 &= -N_{cr1}\,\frac{(\theta_{W1}\theta_{H1}\theta)^2}{(\theta_{F1}\theta_{W1}\theta_{H1}\theta + \theta_{F2}\theta_{W2}\theta_{H2}(1-\theta))^2} \\ &\quad -N_{lr1}\,\frac{(\theta_{W1}\theta_{H1}\theta)^2}{((1-\theta_{F1})\theta_{W1}\theta_{H1}\theta + (1-\theta_{F2})\theta_{W2}\theta_{H2}(1-\theta))^2}\cdots\end{aligned}$$

with all terms negative, suggesting (possibly) a local maximum in the likelihood surface. A full analysis requires evaluating the Hessian matrix of second derivatives and calculating its eigenvalues.

Solutions for Chapter 21
Reinforcement Learning

21.1 The code repository shows an example of this, implemented in the passive 4×3 environment. The agents are found under `lisp/learning/agents/passive*.lisp` and the environment is in `lisp/learning/domains/4x3-passive-mdp.lisp`. (The MDP is converted to a full-blown environment using the function `mdp->environment` which can be found in `lisp/uncertainty/environments/mdp.lisp`.)

21.2 Consider a world with two states, S_0 and S_1, with two actions in each state: stay still or move to the other state. Assume the move action is non-deterministic—it sometimes fails, leaving the agent in the same state. Furthermore, assume the agent starts in S_0 and that S_1 is a terminal state. If the agent tries several move actions and they all fail, the agent may conclude that $T(S_0, Move, S_1)$ is 0, and thus may choose a policy with $\pi(S_0) = Stay$, which is an improper policy. If we wait until the agent reaches S_1 before updating, we won't fall victim to this problem.

21.3 This question essentially asks for a reimplementation of a general scheme for asynchronous dynamic programming of which the prioritized sweeping algorithm is an example (Moore and Atkeson, 1993). For **a.**, there is code for a priority queue in both the Lisp and Python code repositories. So most of the work is the experimentation called for in **b**.

21.4 This utility estimation function is similar to equation (21.9), but adds a term to represent Euclidean distance on a grid. Using equation (21.10), the update equations are the same for θ_0 through θ_2, and the new parameter θ_3 can be calculated by taking the derivative with respect to θ_3:

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha\,(u_j(s) - \hat{U}_\theta(s))\ , \\
\theta_1 &\leftarrow \theta_1 + \alpha\,(u_j(s) - \hat{U}_\theta(s))x\ , \\
\theta_2 &\leftarrow \theta_2 + \alpha\,(u_j(s) - \hat{U}_\theta(s))y\ , \\
\theta_3 &\leftarrow \theta_3 + \alpha\,(u_j(s) - \hat{U}_\theta(s))\sqrt{(x - x_g)^2 + (y - y_g)^2}\ .
\end{aligned}$$

21.5 Code not shown. Several reinforcement learning agents are given in the directory `lisp/learning/agents`.

21.6 Possible features include:

- Distance to the nearest $+1$ terminal state.

- Distance to the nearest -1 terminal state.
- Number of adjacent $+1$ terminal states.
- Number of adjacent -1 terminal states.
- Number of adjacent obstacles.
- Number of obstacles that intersect with a path to the nearest $+1$ terminal state.

21.7 The modification involves combining elements of the environment converter for games (`game->environment` in `lisp/search/games.lisp`) with elements of the function `mdp->environment`. The reward signal is just the utility of winning/drawing/losing and occurs only at the end of the game. The evaluation function used by each agent is the utility function it learns through the TD process. It is important to keep the TD learning process (which is entirely independent of the fact that a game is being played) distinct from the game-playing algorithm. Using the evaluation function with a deep search is probably better because it will help the agents to focus on relevant portions of the search space by improving the quality of play. There is, however, a tradeoff: the deeper the search, the more computer time is used in playing each training game.

21.8 This is a relatively time-consuming exercise. Code not shown to compute three-dimensional plots. The utility functions are:

a. $U(x, y) = 1 - \gamma((10 - x) + (10 - y))$ is the true utility, and is linear.

b. Same as in a, except that $U(10, 1) = -1$.

c. The exact utility depends on the exact placement of the obstacles. The best approximation is the same as in a. The features in exercise 21.9 might improve the approximation.

d. The optimal policy is to head straight for the goal from any point on the right side of the wall, and to head for (5, 10) first (and then for the goal) from any point on the left of the wall. Thus, the exact utility function is:

$$\begin{aligned} U(x, y) &= 1 - \gamma((10 - x) + (10 - y)) \qquad \text{(if } x \geq 5\text{)} \\ &= 1 - \gamma((5 - x) + (10 - y)) - 5\gamma \qquad \text{(if } x < 5\text{)} \end{aligned}$$

Unfortunately, this is not linear in x and y, as stated. Fortunately, we can restate the optimal policy as "head straight up to row 10 first, then head right until column 10." This gives us the same exact utility as in a, and the same linear approximation.

e. $U(x, y) = 1 - \gamma(|5 - x| + |5 - y|)$ is the true utility. This is also not linear in x and y, because of the absolute value signs. All can be fixed by introducing the features $|5 - x|$ and $|5 - y|$.

21.9 Code not shown.

21.10 To map evolutionary processes onto the formal model of reinforcement learning, one must find evolutionary analogs for the reward signal, learning process, and the learned policy. Let us start with a simple animal that does not learn during its own lifetime. This animal's genotype, to the extent that it determines animal's behavior over its lifetime, can be thought of as the parameters θ of a policy pi_θ. Mutations, crossover, and related processes are the

part of the learning algorithm—like an empirical gradient neighborhood generator in policy search—that creates new values of θ. One can also imagine a reinforcement learning process that works on many different copies of π simultaneously, as evolution does; evolution adds the complication that each copy of π modifies the environment for other copies of π, whereas in RL the environment dynamics are assumed fixed, independent of the policy chosen by the agent. The most difficult issue, as the question indicates, is the reward function and the underlying objective function of the learning process. In RL, the objective function is to find policies that maximize the expected sum of rewards over time. Biologists usually talk about evolution as maximizing "reproductive fitness," i.e., the ability of individuals of a given genotype to reproduce and thereby propagate the genotype to the next generation. In this simple view, evolution's "objective function" is to find the π that generates the most copies of itself over infinite time. Thus, the "reward signal" is positive for creation of new individuals; death, *per se*, seems to be irrelevant.

Of course, the real story is much more complex. Natural selection operates not just at the genotype level but also at the level of individual genes and groups of genes; the environment is certainly multiagent rather than single-agent; and, as noted in the case of Baldwinian evolution in Chapter 4, evolution may result in organisms that have hardwired reward signals that are related to the fitness reward and may use those signals to learn during their lifetimes.

As far as we know there has been no careful and philosophically valid attempt to map evolution onto the formal model of reinforcement learning; any such attempt must be careful not to *assume* that such a mapping is possible or to *ascribe* a goal to evolution; at best, one may be able to interpret what evolution tends to do *as if* it were the result of some maximizing process, and ask what it is that is being maximized.

Solutions for Chapter 22
Natural Language Processing

22.1 Code not shown. The distribution of words should fall along a Zipfian distribution: a straight line on a log-log scale. The generated language should be similar to the examples in the chapter.

22.2 Using a unigram language model, the probability of a segmentation of a string $s_{1:N}$ into k nonempty words $s = w_1 \dots w_k$ is $\prod_{i=1}^{k} P_{lm}(w_i)$ where P_{lm} is the unigram language model. This is not normalized without a distribution over the number of words k, but let's ignore this for now.

To see that we can find the most probable segmentation of a string by dynamic programming, let $p(i)$ be the maximum probability of any segmentation of $s_{i:N}$ into words. Then $p(N+1) = 1$ and

$$p(i) = \max_{j=i,\dots,N} P_{lm}(s_{i:j})\, p(j+1)$$

because any segmentation of $s_{i:N}$ starts with a single word spanning $s_{i:j}$ and a segmentation of the rest of the string $s_{j+1:N}$. Because we are using a unigram model, the optimal segmentation of $s_{j+1:N}$ does not depend on the earlier parts of the string.

Using the techniques of this chapter to form a unigram model accessed by the function `prob_word(word)`, the following Python code solves the above dynamic program to output an optimal segmentation:

```
def segment(text):
    length = len(text)
    max_prob = [0] * (length+1)
    max_prob[length] = 1
    split_idx = [-1] * (length+1)
    for start in range(length,-1,-1):
        for split in range(start+1,length+1):
            p = max_prob[split] * prob_word(text[start:split])
            if p > max_prob[start]:
                max_prob[start] = p
                split_idx[start] = split
    i = 0
    words = []
    while i < length:
        words.append(text[i:split_idx[i]])
        i = split_idx[i]
        if i == -1:
```

```
            return None  # for text with zero probability
    return words
```

One caveat is the language model must assign probabilities to unknown words based on their length, otherwise sufficiently long strings will be segmented as single unknown words. One natural option is to fit an exponential distribution to the words lengths of a corpus. Alternatively, one could learn a distribution over the number of words in a string based on its length, add a $P(k)$ term to the probability of a segmentation, and modify the dynamic program to handle this (i.e., to compute $p(i, k)$ the maximum probability of segmenting $s_{i:N}$ into k words).

22.3 Code not shown. The approach suggested here will work in some cases, for authors with distinct vocabularies. For more similar authors, other features such as bigrams, average word and sentence length, parts of speech, and punctuation might help. Accuracy will also depend on how many authors are being distinguished. One interesting way to make the task easier is to group authors into male and female, and try to distinguish the sex of an author not previously seen. This was suggested by the work of Shlomo Argamon.

22.4 Code not shown. There are now several open-source projects to do Bayesian spam filtering, so beware if you assign this exercise.

22.5 Doing the evaluation is easy, if a bit tedious (requiring 150 page evaluations for the complete 10 documents $\times$ 3 engines $\times$ 5 queries). Explaining the differences is more difficult. Some things to check are whether the good results in one engine are even in the other engines at all (by searching for unique phrases on the page); check whether the results are commercially sponsored, are produced by human editors, or are algorithmically determined by a search ranking algorithm; check whether each engine does the features mentioned in the next exercise.

22.6 One good way to do this is to first find a search that yields a single page (or a few pages) by searching for rare words or phrases on the page. Then make the search more difficult by adding a variant of one of the words on the page—a word with different case, different suffix, different spelling, or a synonym for one of the words on the page, and see if the page is still returned. (Make sure that the search engine requires all terms to match for this technique to work.)

22.7 Code not shown. The simplest approach is to look for a string of capitalized words, followed by "Inc" or "Co." or "Ltd." or similar markers. A more complex approach is to get a list of company names (e.g. from an online stock service), look for those names as exact matches, and also extract patterns from them. Reporting recall and precision requires a clearly-defined corpus.

22.8

A. Use the precision on the first 20 documents returned.
B. Use the reciprocal rank of the first relevant document. Or just the rank, considered as a cost function (large is bad).
C. Use the recall.

D. Score this as 1 if the first 100 documents retrieved contain at least one relevant to the query and 0 otherwise.

E. Score this as \$($A(R + I) + BR - NC$) where R is the number of relevant documents retrieved, I is the number of irrelevant documents retrieved, and C is the number of relevant documents not retrieved.

F. One model would be a probabilistic one, in which, if the user has seen R relevant documents and I irrelevant ones, she will continue searching with probability $p(R, I)$ for some function p, to be specified. The measure of quality is then the expected number of relevant documents examined.

Solutions for Chapter 23
Natural Language for Communication

23.1 No answer required; just read the passage.

23.2 The prior is represented by rules such as

$$P(N_0 = A): \qquad S \rightarrow A\ S_A$$

where S_A means "rest of sentence after an A." Transitions are represented as, for example,

$$P(N_{t+1} = B \mid N_t = A): \qquad S_A \rightarrow B\ S_B$$

and the sensor model is just the lexical rules such as

$$P(W_t = \text{is} \mid N_t = A): \qquad A \rightarrow \text{is}\ .$$

23.3

a. (i).

b. This has two parses. The first uses $VP \rightarrow VP\ Adverb$, $VP \rightarrow Copula\ Adjective$, $Copula \rightarrow is$, $Adjective \rightarrow$ **well**, $Adverb \rightarrow$ **well**. Its probability is

$$0.2 \times 0.2 \times 0.8 \times 0.5 \times 0.5 = 0.008\ .$$

The second uses $VP \rightarrow VP\ Adverb$ twice, $VP \rightarrow Verb$, $Verb \rightarrow is$, and $Adverb \rightarrow$ **well** twice. Its probability is

$$0.2 \times 0.2 \times 0.1 \times 0.5 \times 0.5 \times 0.5 = 0.0005\ .$$

The total probability is 0.0085.

c. It exhibits both lexical and syntactic ambiguity.

d. True. There can only be finitely many ways to generate the finitely many strings of 10 words.

23.4 The purpose of this exercise is to get the student thinking about the properties of natural language. There is a wide variety of acceptable answers. Here are ours:

- **Grammar and Syntax** Java: formally defined in a reference book. Grammaticality is crucial; ungrammatical programs are not accepted. English: unknown, never formally defined, constantly changing. Most communication is made with "ungrammatical" utterances. There is a notion of graded acceptability: some utterances are judged slightly ungrammatical or a little odd, while others are clearly right or wrong.

- **Semantics** Java: the semantics of a program is formally defined by the language specification. More pragmatically, one can say that the meaning of a particular program is the JVM code emitted by the compiler. English: no formal semantics, meaning is context dependent.
- **Pragmatics and Context-Dependence** Java: some small parts of a program are left undefined in the language specification, and are dependent on the computer on which the program is run. English: almost everything about an utterance is dependent on the situation of use.
- **Compositionality** Java: almost all compositional. The meaning of "A + B" is clearly derived from the meaning of "A" and the meaning of "B" in isolation. English: some compositional parts, but many non-compositional dependencies.
- **Lexical Ambiguity** Java: a symbol such as "Avg" can be locally ambiguous as it might refer to a variable, a class, or a function. The ambiguity can be resolved simply by checking the declaration; declarations therefore fulfill in a very exact way the role played by background knowledge and grammatical context in English. English: much lexical ambiguity.
- **Syntactic Ambiguity** Java: the syntax of the language resolves ambiguity. For example, in "if (X) if (Y) A; else B;" one might think it is ambiguous whether the "else" belongs to the first or second "if," but the language is specified so that it always belongs to the second. English: much syntactic ambiguity.
- **Reference** Java: there is a pronoun "this" to refer to the object on which a method was invoked. Other than that, there are no pronouns or other means of indexical reference; no "it," no "that." (Compare this to stack-based languages such as Forth, where the stack pointer operates as a sort of implicit "it.") There is reference by name, however. Note that ambiguities are determined by scope—if there are two or more declarations of the variable "X", then a use of X refers to the one in the innermost scope surrounding the use. English: many techniques for reference.
- **Background Knowledge** Java: none needed to interpret a program, although a local "context" is built up as declarations are processed. English: much needed to do disambiguation.
- **Understanding** Java: understanding a program means translating it to JVM byte code. English: understanding an utterance means (among other things) responding to it appropriately; participating in a dialog (or choosing not to participate, but having the potential ability to do so).

As a follow-up question, you might want to compare different languages, for example: English, Java, Morse code, the SQL database query language, the Postscript document description language, mathematics, etc.

23.5 The purpose of this exercise is to get some experience with simple grammars, and to see how context-sensitive grammars are more complicated than context-free. One approach to writing grammars is to write down the strings of the language in an orderly fashion, and then see how a progression from one string to the next could be created by recursive application of rules. For example:

a. The language $a^n b^n$: The strings are ϵ, ab, $aabb$, ...(where ϵ indicates the null string). Each member of this sequence can be derived from the previous by wrapping an a at the start and a b at the end. Therefore a grammar is:

$$S \rightarrow \epsilon$$
$$S \rightarrow a\ S\ b$$

b. The palindrome language: Let's assume the alphabet is just a, b and c. (In general, the size of the grammar will be proportional to the size of the alphabet. There is no way to write a context-free grammar without specifying the alphabet/lexicon.) The strings of the language include ϵ, *a, b, c, aa, bb, cc, aaa, aba, aca, bab, bbb, bcb, ...*. In general, a string can be formed by bracketing any previous string with two copies of any member of the alphabet. So a grammar is:

$$S \rightarrow \epsilon \mid a \mid b \mid c \mid a\ S\ a \mid b\ S\ b \mid c\ S\ c$$

c. The duplicate language: For the moment, assume that the alphabet is just ab. (It is straightforward to extend to a larger alphabet.) The duplicate language consists of the strings: ϵ, aa, bb, $aaaa$, $abab$, $bbbb$, $baba$, ... Note that all strings are of even length.

One strategy for creating strings in this language is this:

- Start with markers for the front and middle of the string: we can use the non-terminal F for the front and M for the middle. So at this point we have the string FM.
- Generate items at the front of the string: generate an a followed by an A, or a b followed by a B. Eventually we get, say, $FaAaAbBM$. Then we no longer need the F marker and can delete it, leaving $aAaAbBM$.
- Move the non-terminals A and B down the line until just before the M. We end up with $aabAABM$.
- Hop the As and Bs over the M, converting each to a terminal (a or b) as we go. Then we delete the M, and are left with the end result: $aabaab$.

Here is a grammar to implement this strategy:

$S \rightarrow F\ M$	(starting markers)
$F \rightarrow F\ a\ A$	(introduce symbols)
$F \rightarrow F\ b\ B$	
$F \rightarrow \epsilon$	(delete the F marker)
$A\ a \rightarrow a\ A$	(move non-terminals down to the M)
$A\ b \rightarrow b\ A$	
$B\ a \rightarrow a\ B$	
$B\ b \rightarrow b\ B$	
$A\ M \rightarrow M\ a$	(hop over M and convert to terminal)
$B\ M \rightarrow M\ b$	
$M \rightarrow \epsilon$	(delete the M marker)

Here is a trace of the grammar deriving $aabaab$:

$\underline{S}$

$\underline{F}M$
$\underline{F}bBM$
$\underline{F}aAbBM$
$\underline{F}aAaAbBM$
$a\underline{Aa}AbBM$
$aaA\underline{Ab}BM$
$aa\underline{Ab}ABM$
$aabAA\underline{BM}$
$aabA\underline{AM}b$
$aab\underline{AM}ab$
$aab\underline{M}aab$
$aabaab$

23.6 Grammar (A) does not work, because there is no way for the verb "walked" followed by the adverb "slowly" and the prepositional phrase "to the supermarket" to be parsed as a verb phrase. A verb phrase in (A) must have either two adverbs or be just a verb. Here is the parse under grammar (B):

```
S---NP-+-Pro---Someone
       |
       |-VP-+-V---walked
            |
            |-Vmod-+-Adv---slowly
                   |
                   |-Vmod---Adv---PP---Prep-+-to
                                            |
                                            |-NP-+-Det---the
                                                 |
                                                 |-NP---Noun---supermarket
```

Here is the parse under grammar (C):

```
S---NP-+-Pro---Someone
       |
       |-VP-+-V---walked
            |
            |-Adv-+-Adv---slowly
                  |
                  |-Adv---PP---Prep-+-to
                                    |
                                    |-NP-+-Det---the
                                         |
                                         |-NP---Noun---supermarket
```

23.7 Here is a start of a grammar:

```
Time => DigitHour ":" DigitMinute
    | "midnight" | "noon" | "12 midnight" | "12 noon''
    | ClockHour "o'clock"
```

$$
\begin{array}{rcl}
S & \rightarrow & NP\ VP \\
 & | & S'\ Conj\ S' \\
S' & \rightarrow & NP\ VP \\
 & | & SConj\ S' \\
SConj & \rightarrow & S'\ Conj \\
 & & \\
NP & \rightarrow & \textbf{me} \mid \textbf{you} \mid \textbf{I} \mid \textbf{it} \mid \ldots \\
 & | & \textbf{John} \mid \textbf{Mary} \mid \textbf{Boston} \mid \ldots \\
 & | & \textbf{stench} \mid \textbf{breeze} \mid \textbf{wumpus} \mid \textbf{pits} \mid \ldots \\
 & | & Article\ Noun \\
 & | & ArticleAdjs\ Noun \\
 & | & Digit\ Digit \\
 & | & NP\ PP \\
 & | & NP\ RelClause \\
ArticleAdjs & \rightarrow & Article\ Adjs \\
 & & \\
VP & \rightarrow & \textbf{is} \mid \textbf{feel} \mid \textbf{smells} \mid \textbf{stinks} \mid \ldots \\
 & | & VP\ NP \\
 & | & VP\ Adjective \\
 & | & VP\ PP \\
 & | & VP\ Adverb \\
 & & \\
Adjs & \rightarrow & \textbf{right} \mid \textbf{dead} \mid \textbf{smelly} \mid \textbf{breezy} \ldots \\
 & | & Adjective\ Adjs \\
PP & \rightarrow & Prep\ NP \\
RelClause & \rightarrow & RelPro\ VP
\end{array}
$$

Figure S23.1 The final result after turning $\mathcal{E}_0$ into CNF (omitting probabilities).

```
    | Difference BeforeAfter ExtendedHour

DigitHour => 0 | 1 | ... | 23
DigitMinute => 1 | 2 | ... | 60
HalfDigitMinute => 1 | 2 | ... | 29
ClockHour => ClockDigitHour | ClockWordHour
ClockDigitHour => 1 | 2 | ... | 12
ClockWordHour => "one" | ... | "twelve"
BeforeAfter =>  "to" | "past" | "before" | "after"
Difference => HalfDigitMinute "minutes" | ShortDifference
ShortDifference => "five" | "ten" | "twenty" | "twenty-five" | "quarter" | "half"
ExtendedHour => ClockHour | "midnight" | "noon"
```

The grammar is not perfect; for example, it allows “ten before six” and “quarter past noon,” which are a little odd-sounding, and “half before six,” which is not really OK.

23.8 The final grammar is shown in Figure S23.1. (Note that in early printings, the question asked for the rule $S' \rightarrow S$ to be added.) In step **d**, students may be tempted to drop the rules ($Y \rightarrow \ldots$), which fails immediately.

$S \rightarrow NP(Subjective, number, person)\ VP(number, person) \mid \ldots$
$NP(case, number, person) \rightarrow Pronoun(case, number, person)$
$NP(case, number, Third) \rightarrow Name(number) \mid Noun(number) \mid \ldots$
$VP(number, person) \rightarrow VP(number, person)\ NP(Objective, \_, \_) \mid \ldots$
$PP \rightarrow Preposition\ NP(Objective, \_, \_)$
$Pronoun(Subjective, Singular, First) \rightarrow$ **I**
$Pronoun(Subjective, Singular, Second) \rightarrow$ **you**
$Pronoun(Subjective, Singular, Third) \rightarrow$ **he** | **she** | **it**
$Pronoun(Subjective, Plural, First) \rightarrow$ **we**
$Pronoun(Subjective, Plural, Second) \rightarrow$ **you**
$Pronoun(Subjective, Plural, Third) \rightarrow$ **they**
$Pronoun(Objective, Singular, First) \rightarrow$ **me**
$Pronoun(Objective, Singular, Second) \rightarrow$ **you**
$Pronoun(Objective, Singular, Third) \rightarrow$ **him** | **her** | **it**
$Pronoun(Objective, Plural, First) \rightarrow$ **us**
$Pronoun(Objective, Plural, Second) \rightarrow$ **you**
$Pronoun(Objective, Plural, Third) \rightarrow$ **them**
$Verb(Singular, First) \rightarrow$ **smell**
$Verb(Singular, Second) \rightarrow$ **smell**
$Verb(Singular, Third) \rightarrow$ **smells**
$Verb(Plural, \_) \rightarrow$ **smell**

Figure S23.2 A partial DCG for $\mathcal{E}_1$, modified to handle subject–verb number/person agreement as in Ex. 22.2.

23.9 See Figure S23.2 for a partial DCG. We include both person and number annotation although English really only differentiates the third person singular for verb agreement (except for the verb *be*).

23.10 One parse captures the meaning "I am able to fish" and the other "I put fish in cans." Both have the left branch $NP \rightarrow Pronoun \rightarrow$ **I**, which has probability 0.16.

- The first has the right branch $VP \rightarrow Modal\,Verb$ (0.2) with $Modal \rightarrow$ **can** (0.3) and $Verb \rightarrow$ **fish** (0.1), so its prior probability is

$$0.16 \times 0.2 \times 0.3 \times 0.1 = 0.00096\,.$$

- The second has the right branch $VP \rightarrow VerbNP$ (0.8) with $Verb \rightarrow$ **can** (0.1) and $NP \rightarrow Noun \rightarrow$ **fish** (0.6×0.3), so its prior probability is

$$0.16 \times 0.8 \times 0.1 \times 0.6 \times 0.3 = 0.002304\,.$$

As these are the only two parses, and the conditional probability of the string given the parse is 1, their conditional probabilities given the string are proportional to their priors and sum to 1: 0.294 and 0.706.

23.11 The rule for A is

$$\begin{aligned} A(n') &\rightarrow aA(n)\ \{n' = \textsc{Successor}(n)\} \\ A(1) &\rightarrow a \end{aligned}$$

The rules for B and C are similar.

$$\begin{aligned} &NP(case, number, Third) \rightarrow Name(number) \\ &NP(case, Plural, Third) \rightarrow Noun(Plural) \\ &NP(case, number, Third) \rightarrow Article(number)Noun(number) \\ &Article(Singular) \rightarrow \textbf{a} \mid \textbf{an} \mid \textbf{the} \\ &Article(Plural) \rightarrow \textbf{the} \mid \textbf{some} \mid \textbf{many} \end{aligned}$$

Figure S23.3 A partial DCG for $\mathcal{E}_1$, modified to handle article–noun agreement as in Ex. 22.3.

23.12 See Figure S23.3

23.13

a. Webster's New Collegiate Dictionary (9th edn.) lists multiple meaning for all these words except "multibillion" and "curtailing".

b. The attachment of all the propositional phrases is ambiguous, e.g. does "from . . . loans" attach to "struggling" or "recover"? Does "of money" attach to "depriving" or "companies"? The coordination of "and hiring" is also ambiguous; is it coordinated with "expansion" or with "curtailing" and "depriving" (using British punctuation).

c. The most clear-cut case is "healthy companies" as an example of HEALTH for IN A GOOD FINANCIAL STATE. Other possible metaphors include "Banks . . . recover" (same metaphor as "healthy"), "banks struggling" (PHYSICAL EFFORT for WORK), and "expansion" (SPATIAL VOLUME for AMOUNT OF ACTIVITY); in these cases, the line between metaphor and polysemy is vague.

23.14 This is a very difficult exercise—most readers have no idea how to answer the questions (except perhaps to remember that "too few" is better than "too many"). This is the whole point of the exercise, as we will see in exercise 23.14.

23.15 The main point of this exercise is to show that current translation software is far from perfect. The mistakes made are often amusing for students.

23.16 It's not true in general. With two phrases of length 1 which are inverted f_2, f_1, we have $d_1 = 0$ and $d_2 = 1 - 2 - 1 = -2$ which don't sum to zero.

23.17

a. "I have never seen a better programming language" is easy for most people to see.

b. "John loves mary" seems to be prefered to "Mary loves John" (on Google, by a margin of 2240 to 499, and by a similar margin on a small sample of respondents), but both are of course acceptable.

c. This one is quite difficult. The first sentence of the second paragraph of Chapter 22 is "Communication is the intentional exchange of information brought about by the production and perception of signs drawn from a shared system of conventional signs." However, this cannot be reliably recovered from the string of words given here. Code not shown for testing the probabilities of permutations.

d. This one is easy for students of US history, being the beginning of the second sentence of the Declaration of Independence: "We hold these truths to be self-evident, that all men are created equal . . ."

23.18

To solve questions like this more generally one can use the Viterbi algorithm. However, observe that the first two states must be onset, as onset is the only state which can output C_1 and C_2. Similarly the last two state must be end. The third state is either onset or mid, and the fourth and fifth are either mid or end. Having reduced to eight possibilities, we can exhaustively enumerate to find the most likely sequence and its probability.

First we compute the joint probabilities of the hidden states and output sequence:

$$\begin{aligned}
P(1234466, OOOMMEE) &= 0.5 \times 0.2 \times 0.3 \times 0.7 \times 0.7 \times 0.5 \times 0.5 \\
&\quad \times 0.3 \times 0.3.7 \times 0.9 \times 0.1 \times 0.4 \\
&= 8.335 \times 10^{-6} \\
P(1234466, OOOMEEE) &= 5.292 \times 10^{-7} \\
P(1234466, OOOEMEE) &= 0 \\
P(1234466, OOOEEEE) &= 0 \\
P(1234466, OOMMMEE) &= 1.667 \times 10^{-5} \\
P(1234466, OOMMEEE) &= 1.058 \times 10^{-6} \\
P(1234466, OOMEMEE) &= 0 \\
P(1234466, OOMEEEE) &= 6.720 \times 10^{-8}
\end{aligned}$$

We find the most likely sequence was O, O, M, M, M, E, E. Normalizing, we find this has probability 0.6253.

23.19 Now we can answer the difficult questions of 22.7:

- The steps are sorting the clothes into piles (e.g., white vs. colored); going to the washing machine (optional); taking the clothes out and sorting into piles (e.g., socks versus shirts); putting the piles away in the closet or bureau.
- The actual running of the washing machine is never explicitly mentioned, so that is one possible answer. One could also say that drying the clothes is a missing step.
- The material is clothes and perhaps other washables.

- Putting too many clothes together can cause some colors to run onto other clothes.
- It is better to do too few.
- So they won’t run; so they get thoroughly cleaned; so they don’t cause the machine to become unbalanced.

Solutions for Chapter 24
Perception

24.1 The small spaces between leaves act as pinhole cameras. That means that the circular light spots you see are actually images of the circular sun. You can test this theory next time there is a solar eclipse: the circular light spots will have a crescent bite taken out of them as the eclipse progresses. (Eclipse or not, the light spots are easier to see on a sheet of paper than on the rough forest floor.)

24.2 Consider the set of light rays passing through the center of projection (the pinhole or the lens center), and tangent to the surface of the sphere. These define a double cone whose apex is the center of projection. Note that the outline of the sphere on the image plane is just the cross section corresponding to the intersection of this cone with the image plane of the camera. We know from geometry that such a conic section will typically be an ellipse. It is a circle in the special case that the sphere is directly in front of the camera (its center lies on the optical axis).

While on a planar retina, the image of an off-axis sphere would indeed be an ellipse, the human visual system tries to infer what is in the three-dimensional scene, and here the most likely solution is that one is looking at a sphere.

Some students might note that the eye's retina is not planar but closer to spherical. On a perfectly spherical retina the image of a sphere will be circular. The point of the question remains valid, however.

24.3 Recall that the image brightness of a Lambertian surface (page 743) is given by $I(x, y) = k\mathbf{n}(x, y).\mathbf{s}$. Here the light source direction **s** is along the x-axis. It is sufficient to consider a horizontal cross-section (in the x–z plane) of the cylinder as shown in Figure S24.1(a). Then, the brightness $I(x) = k \cos\theta(x)$ for all the points on the right half of the cylinder. The left half is in shadow. As $x = r\cos\theta$, we can rewrite the brightness function as $I(x) = \frac{kx}{r}$ which reveals that the isobrightness contours in the lit part of the cylinder must be equally spaced. The view from the z-axis is shown in Figure S24.1(b).

24.4 We list the four classes and give two or three examples of each:

a. *depth*: Between the top of the computer monitor and the wall behind it. Between the side of the clock tower and the sky behind it. Between the white sheets of paper in the foreground and the book and keyboard behind them.

b. *surface normal*: At the near corner of the pages of the book on the desk. At the sides of the keys on the keyboard.

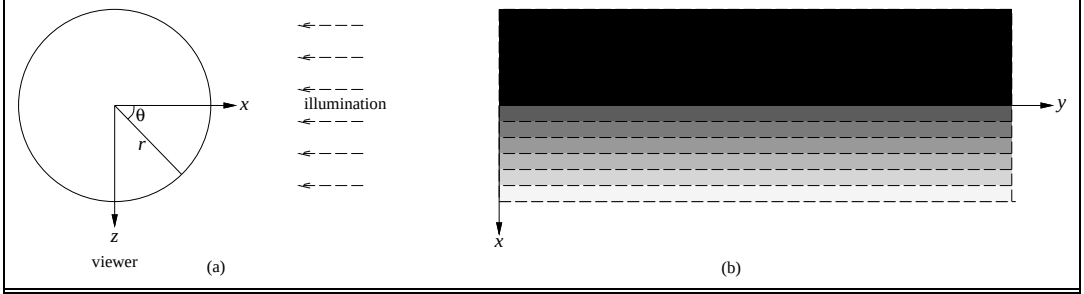


Figure S24.1 (a) Geometry of the scene as viewed from along the y-axis. (b) The scene from the z-axis, showing the evenly spaced isobrightness contours.

c. *reflectance*: Between the white paper and the black lines on it. Between the "golden" bridge in the picture and the blue sky behind it.

d. *illumination*: On the windowsill, the shadow from the center glass pane divider. On the paper with Greek text, the shadow along the left from the paper on top of it. On the computer monitor, the edge between the white window and the blue window is caused by different illumination by the CRT.

24.5 Before answering this exercise, we draw a diagram of the apparatus (top view), shown in Figure S24.2. Notice that we make the approximation that the focal length is the distance from the lens to the image plane; this is valid for objects that are far away. Notice that this question asks nothing about the y coordinates of points; we might as well have a single line of 512 pixels in each camera.

a. Solve this by constructing similar triangles: whose hypotenuse is the dotted line from object to lens, and whose height is 0.5 meters and width 16 meters. This is similar to a triangle of width 16cm whose hypotenuse projects onto the image plane; we can compute that its height must be 0.5cm; this is the offset from the center of the image plane. The other camera will have an offset of 0.5cm in the opposite direction. Thus the total disparity is 1.0cm, or, at 512 pixels/10cm, a disparity of 51.2 pixels, or 51, since there are no fractional pixels. Objects that are farther away will have smaller disparity. Writing this as an equation, where d is the disparity in pixels and Z is the distance to the object, we have:

$$d = 2 \times \frac{512 \text{ pixels}}{10 \text{ cm}} \times 16 \text{ cm} \times \frac{0.5 \text{ m}}{Z}$$

b. In other words, this question is asking how much further than 16m could an object be, and still occupy the same pixels in the image plane? Rearranging the formula above by swapping d and Z, and plugging in values of 51 and 52 pixels for d, we get values of Z of 16.06 and 15.75 meters, for a difference of 31cm (a little over a foot). This is the range resolution at 16 meters.

c. In other words, this question is asking how far away would an object be to generate a disparity of one pixel? Objects farther than this are in effect out of range; we can't say

where they are located. Rearranging the formula above by swapping d and Z we get 51.2 meters.

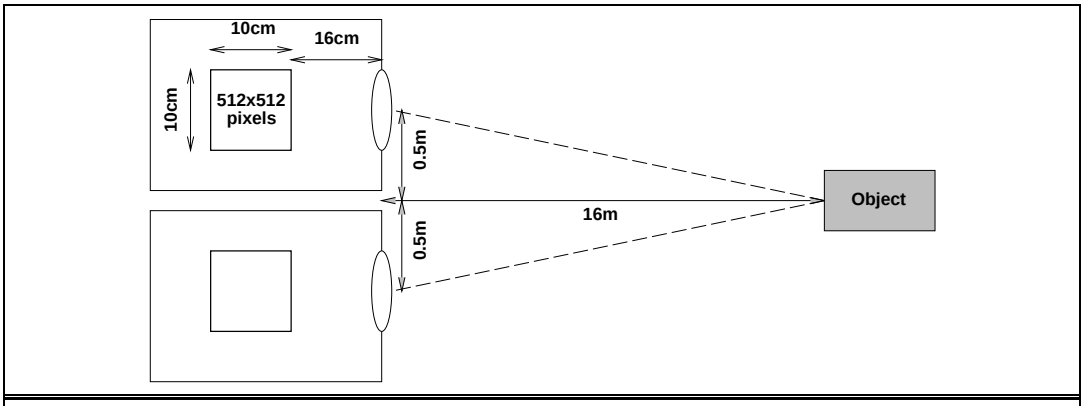


Figure S24.2 Top view of the setup for stereo viewing (Exercise 24.6).

24.6

a. False. This can be quite difficult, particularly when some point are occluded from one eye but not the other.

b. True. The grid creates an apparent texture whose distortion gives good information as to surface orientation.

c. False.

d. False. A disk viewed edge-on appears as a straight line.

24.7 A, B, C can be viewed in stereo and hence their depths can be measured, allowing the viewer to determine that B is nearest, A and C are equidistant and slightly further away. Neither D nor E can be seen by both cameras, so stereo cannot be used. Looking at the figure, it appears that the bottle *occludes* D from Y and E from X, so D and E must be further away than A, B, C, but their relative depths cannot be determined. There is, however, another possibility (noticed by Alex Fabrikant). Remember that each camera sees the camera's-eye view not the bird's-eye view. X sees DABC and Y sees ABCE. It is possible that D is very close to camera X, so close that it falls outside the field of view of camera Y; similarly, E might be very close to Y and be outside the field of view of X. Hence, unless the cameras have a 180-degree field of view—probably impossible—there is no way to determine whether D and E are in front of or behind the bottle.

Solutions for Chapter 25
Robotics

25.1 To answer this question, consider all possibilities for the initial samples before and after resampling. This can be done because there are only finitely many states. The following C++ program calculates the results for finite N. The result for $N = \infty$ is simply the posterior, calculated using Bayes rule.

```
int
main(int argc, char *argv[])
{
  // parse command line argument
  if (argc != 3){
    cerr << "Usage: " << argv[0] << " <number of samples>"
 << " <number of states>" << endl;
    exit(0);
  }

  int numSamples = atoi(argv[1]);
  int numStates = atoi(argv[2]);
  cerr << "number of samples: " << numSamples << endl
       << "number of states:  " << numStates  << endl;
  assert(numSamples >= 1);
  assert(numStates >= 1);

  // generate counter
  int samples[numSamples];
  for (int i = 0; i < numSamples; i++)
    samples[i] = 0;

  // set up probability tables
  assert(numStates == 4); // presently defined for 4 states
  double condProbOfZ[4] = {0.8, 0.4, 0.1, 0.1};
  double posteriorProb[numStates];
  for (int i = 0; i < numStates; i++)
    posteriorProb[i] = 0.0;
  double eventProb = 1.0 / pow(numStates, numSamples);

  //loop through all possibilities
  for (int done = 0; !done; ){

    // compute importance weights (is probability distribution)
    double weight[numSamples], totalWeight = 0.0;
    for (int i = 0; i < numSamples; i++)
      totalWeight += weight[i] = condProbOfZ[samples[i]];
    // normalize them
    for (int i = 0; i < numSamples; i++)
      weight[i] /= totalWeight;

    // calculate contribution to posterior probability
    for (int i = 0; i < numSamples; i++)
      posteriorProb[samples[i]] += eventProb * weight[i];
```

(a)

```
    // increment counter
    for (int i = 0; i < numSamples && i != -1;){
      samples[i]++;
      if (samples[i] >= numStates)
samples[i++] = 0;
      else
i = -1;
      if (i == numSamples)
done = 1;
    }
  }

  // print result
  cout << "Result:  ";
  for (int i = 0; i < numStates; i++)
    cout << " " << posteriorProb[i];
  cout << endl;


  // calculate asymptotic expectation
  double totalWeight = 0.0;
  for (int i = 0; i < numStates; i++)
    totalWeight += condProbOfZ[i];

  cout << "Unbiased:";
  for (int i = 0; i < numStates; i++)
    cout << " " << condProbOfZ[i] / totalWeight;
  cout << endl;

  // calculate KL divergence
  double kl = 0.0;
  for (int i = 0; i < numStates; i++)
    kl += posteriorProb[i] * (log(posteriorProb[i]) -
      log(condProbOfZ[i] / totalWeight));
  cout << "KL divergence: " << kl << endl;
}
```

(b)

Figure S25.1 Code to calculate answer to exercise 25.1.

a. The program (correctly) calculates the following posterior distributions for the four states, as a function of the number of samples N. Note that for $N = 1$, the measurement is ignored entirely! The correct posterior for $N = \infty$ is calculated using Bayes rule.

N	p(sample at s_1)	p(sample at s_2)	p(sample at s_3)	p(sample at s_4)
$N = 1$	0.25	0.25	0.25	0.25
$N = 2$	0.368056	0.304167	0.163889	0.163889
$N = 3$	0.430182	0.314463	0.127677	0.127677
$N = 4$	0.466106	0.314147	0.109874	0.109874
$N = 5$	0.488602	0.311471	0.0999636	0.0999636
$N = 6$	0.503652	0.308591	0.0938788	0.0938788
$N = 7$	0.514279	0.306032	0.0898447	0.0898447
$N = 8$	0.522118	0.303872	0.0870047	0.0870047
$N = 9$	0.528112	0.30207	0.0849091	0.0849091
$N = 10$	0.532829	0.300562	0.0833042	0.0833042
$N = \infty$	0.571429	0.285714	0.0714286	0.0714286

b. Plugging the posterior for $N = \infty$ into the definition of the Kullback Liebler Divergence gives us:

N	$KL(\hat{p}, p)$	N	$KL(\hat{p}, p)$
$N = 1$	0.386329	$N = 7$	0.00804982
$N = 2$	0.129343	$N = 8$	0.00593024
$N = 3$	0.056319	$N = 9$	0.00454205
$N = 4$	0.029475	$N = 10$	0.00358663
$N = 5$	0.0175705	$N = \infty$	0

c. The proof for $N = 1$ is trivial, since the re-weighting ignores the measurement probability entirely. Therefore, the probability for generating a sample in any of the locations in S is given by the initial distribution, which is uniform.

For $N = 2$, a proof is easily obtained by considering all $2^4 = 16$ ways in which initial samples are generated:

number	samples		probability of sample set	$p(z\|s)$ for each sample		weights for each sample		probability of resampling for each location in S			
1	0	0	$\frac{1}{16}$	$\frac{4}{5}$	$\frac{4}{5}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{16}$	0	0	0
2	0	1	$\frac{1}{16}$	$\frac{2}{5}$	$\frac{4}{5}$	$\frac{3}{9}$	$\frac{6}{9}$	$\frac{1}{24}$	$\frac{1}{48}$	0	0
3	0	2	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{4}{5}$	$\frac{1}{9}$	$\frac{8}{9}$	$\frac{1}{18}$	0	$\frac{1}{144}$	0
4	0	3	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{4}{5}$	$\frac{1}{9}$	$\frac{8}{9}$	$\frac{1}{18}$	0	0	$\frac{1}{144}$
5	1	0	$\frac{1}{16}$	$\frac{4}{5}$	$\frac{2}{5}$	$\frac{6}{9}$	$\frac{3}{9}$	$\frac{1}{24}$	$\frac{1}{48}$	0	0
6	1	1	$\frac{1}{16}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{16}$	0	0
7	1	2	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{4}{5}$	0	$\frac{1}{20}$	$\frac{1}{80}$	0
8	1	3	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{2}{5}$	$\frac{1}{5}$	$\frac{4}{5}$	0	$\frac{1}{20}$	0	$\frac{1}{80}$
9	2	0	$\frac{1}{16}$	$\frac{4}{5}$	$\frac{1}{10}$	$\frac{8}{9}$	$\frac{1}{9}$	$\frac{1}{18}$	0	$\frac{1}{144}$	0
10	2	1	$\frac{1}{16}$	$\frac{2}{5}$	$\frac{1}{10}$	$\frac{4}{5}$	$\frac{1}{5}$	0	$\frac{1}{20}$	$\frac{1}{80}$	0
11	2	2	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{1}{16}$	0
12	2	3	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{1}{32}$	$\frac{1}{32}$
13	3	0	$\frac{1}{16}$	$\frac{4}{5}$	$\frac{1}{10}$	$\frac{8}{9}$	$\frac{1}{9}$	$\frac{1}{18}$	0	0	$\frac{1}{144}$
14	3	1	$\frac{1}{16}$	$\frac{2}{5}$	$\frac{1}{10}$	$\frac{4}{5}$	$\frac{1}{5}$	0	$\frac{1}{20}$	0	$\frac{1}{80}$
15	3	2	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{1}{32}$	$\frac{1}{32}$
16	3	3	$\frac{1}{16}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	$\frac{1}{16}$
sum of all probabilities								$\frac{53}{144}$	$\frac{73}{240}$	$\frac{59}{360}$	$\frac{59}{360}$

A quick check should convince you that these numbers are the same as above. Placing this into the definition of the Kullback Liebler divergence with the correct posterior distribution, gives us 0.129343.

For $N = \infty$ we know that the sampler is unbiased. Hence, the probability of generating a sample is the same as the posterior distribution calculated by Bayes filters. Those are given above as well.

d. Here are two possible modifications. First, if the initial robot location is known with absolute certainty, the sampler above will always be unbiased. Second, if the sensor measurement z is equally likely for all states, that is $p(z|s_1) = p(z|s_2) = p(z|s_3) = p(z|s_4)$, it will also be unbiased. An *invalid* answer, which we frequently encountered in class, pertains to the algorithm (instead of the problem formulation). For example, replacing particle filters by the exact discrete Bayes filer remedies the problem but is not a legitimate answer to this question. Neither is the use of infinitely many particles.

25.2 Implementing Monte Carlo localization requires a lot of work but is a premiere way to gain insights into the basic workings of probabilistic algorithms in robotics, and the intricacies inherent in real data. We have used this exercise in many courses, and students consistently expressed having learned a lot. We strongly recommend this exercise!

The implementation is not as straightforward as it may appear at first glance. Common problems include:

- The sensor model models too little noise, or the wrong type of noise. For example, a simple Gaussian will not work here.
- The motion model assumes too little or too much noise, or the wrong type of noise. Here a Gaussian will work fine though.
- The implementation may introduce unnecessarily high variance in the resulting sampling set, by sampling too often, or by sampling in the wrong way. This problem manifests itself by diversity disappearing prematurely, often with the wrong samples surviving. While the basic MCL algorithm, as stated in the book, suggests that sampling should occur after each motion update, implementations that sample less frequently tend to yield superior results. Further, drawing samples independently of each other is inferior to so-called low variance samplers. Here is a version of low variance sampling, in which $\mathcal{X}$ denotes the particles and W their importance weights. The resulting resampled particles reside in the set S'.

function LOW-VARIANCE-WEIGHTED-SAMPLE-WITH-REPLACEMENT*(S, W):*
  $S' = \{\}$
  $b = \sum_{i=1}^{N} W[i]$
  $r = \text{rand}(0; b)$
  for $n = 1$ ***to*** N ***do***
    $i = \text{argmin}_j \sum_{m=1}^{j} W[m] \geq r$
    add $S[i]$ *to* S'
    $r = (r + \text{rand}(0; c))$ *modulo* b
  return S'

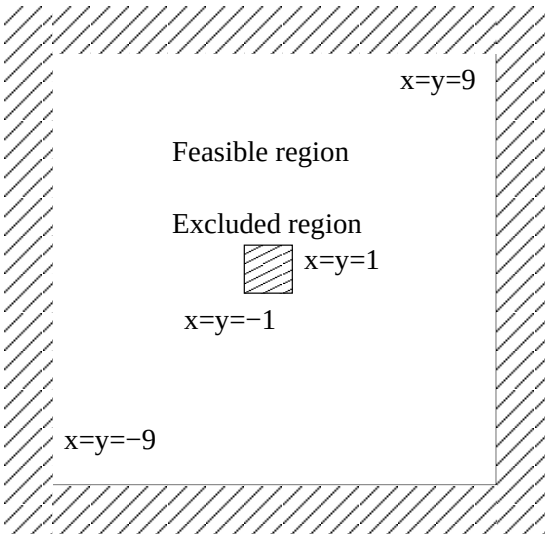


Figure S25.2 Robot configuration.

The parameter c determines the speed at which we cycle through the sample set. While each sample's probability remains the same as if it were sampled independently, the resulting samples are dependent, and the variance of the sample set S' is lower (assuming $c < b$). As a pleasant side effect, the low-variance samples is also easily implemented in $O(N)$ time, which is more difficult for the independent sampler.

- Samples are started in the occupied or unknown parts of the map, or are allowed into those parts during the forward sampling (motion prediction) step of the MCL algorithm.
- Too few samples are used. A few thousand should do the job, a few hundred will probably not.

The algorithm can be sped up by pre-caching all noise-free measurements, for all x-y-θ poses that the robot might assume. For that, it is convenient to define a grid over the space of all poses, with 10 centimeters spatial and 2 degrees angular resolution. One might then compute the noise-free measurements for the centers of those grid cells. The sensor model is clearly just a function of those correct measurements; and computing those takes the bulk of time in MCL.

25.3 See Figure S25.2.

25.4

A. Hill climbing down the potential moves manipulator B down the rod to the point where the derivative of the term "square of distance from current position of B to goal position" is exactly the negative of the derivative of the term "1/square of distance from A to B". This is a local minimum of the potential function, because it is a minimum of the sum of those two terms, with A held fixed, and small movements of A do not change the

value of the term "1/square of distance from A to B", and only increase the value of the term "square of distance from current position of A to goal position"

B. Add a term of the form "1/square of distance between the center of A and the center of B." Now the stopping configuration of part A is no longer a local minimum because moving A to the left decreases this term. (Moving A to the left does also increase the value of the term "square of distance from current position of A to goal position", but that term is at a local minimum, so its derivative is zero, so the gain outweighs the loss, at least for a while.) For the right combination of linear coefficient, hill climbing will find its way to a correct solution.

25.5 Let α be the shoulder and β be the elbow angle. The coordinates of the end effector are then given by the following expression. Here z is the height and x the horizontal displacement between the end effector and the robot's base (origin of the coordinate system):

$$\begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} 0cm \\ 60cm \end{pmatrix} + \begin{pmatrix} \sin\alpha \\ \cos\alpha \end{pmatrix} \cdot 40cm + \begin{pmatrix} \sin(\alpha+\beta) \\ \cos(\alpha+\beta) \end{pmatrix} \cdot 40cm$$

Notice that this is only one way to define the kinematics. The zero-positions of the angles α and β can be anywhere, and the motors may turn clockwise or counterclockwise. Here we chose define these angles in a way that the arm points straight up at $\alpha = \beta = 0$; furthermore, increasing α and β makes the corresponding joint rotate counterclockwise.

Inverse kinematics is the problem of computing α and β from the end effector coordinates x and z. For that, we observe that the elbow angle β is uniquely determined by the Euclidean distance between the shoulder joint and the end effector. Let us call this distance d. The shoulder joint is located $60cm$ above the origin of the coordinate system; hence, the distance d is given by $d = \sqrt{x^2 + (z - 60cm)^2}$. An alternative way to calculate d is by recovering it from the elbow angle β and the two connected joints (each of which is $40cm$ long): $d = 2 \cdot 40cm \cdot \cos\frac{\beta}{2}$. The reader can easily derive this from basic trigonometry, exploiting the fact that both the elbow and the shoulder are of equal length. Equating these two different derivations of d with each other gives us

$$\sqrt{x^2 + (z - 60cm)^2} = 80cm \cdot \cos\frac{\beta}{2} \tag{25.1}$$

or

$$\beta = \pm 2 \cdot \arccos\frac{\sqrt{x^2 + (z - 60cm)^2}}{80cm} \tag{25.2}$$

In most cases, β can assume two symmetric configurations, one pointing down and one pointing up. We will discuss exceptions below.

To recover the angle α, we note that the angle between the shoulder (the base) and the end effector is given by $\arctan 2(x, z - 60cm)$. Here $\arctan 2$ is the common generalization of the arcus tangens to all four quadrants (check it out—it is a function in C). The angle α is now obtained by adding $\frac{\beta}{2}$, again exploiting that the shoulder and the elbow are of equal length:

$$\alpha = \arctan 2(x, z - 60cm) - \frac{\beta}{2} \tag{25.3}$$

Of course, the actual value of α depends on the actual choice of the value of β. With the exception of singularities, β can take on exactly two values.

The inverse kinematics is *unique* if β assumes a single value; as a consequence, so does alpha. For this to be the case, we need that

$$\arccos \frac{\sqrt{x^2 + (z - 60cm)^2}}{80cm} = 0 \tag{25.4}$$

This is the case exactly when the argument of the $\arccos$ is 1, that is, when the distance $d = 80cm$ and the arm is fully stretched. The end points x, z then lie on a circle defined by $\sqrt{x^2 + (z - 60cm)^2} = 80cm$. If the distance $d > 80cm$, there is no solution to the inverse kinematic problem: the point is simply too far away to be reachable by the robot arm.

Unfortunately, configurations like these are numerically unstable, as the quotient may be slightly larger than one (due to truncation errors). Such points are commonly called *singularities*, and they can cause major problems for robot motion planning algorithms. A second singularity occurs when the robot is "folded up," that is, $\beta = 180°$. Here the end effector's position is identical with that of the robot elbow, regardless of the angle α: $x = 0cm$ and $z = 60cm$. This is an important singularity, as there are *infinitely* many solutions to the inverse kinematics. As long as $\beta = 180°$, the value of α can be arbitrary. Thus, this simple robot arm gives us an example where the inverse kinematics can yield zero, one, two, or infinitely many solutions.

25.6 Code not shown.

25.7

a. The configurations of the robots are shown by the black dots in Figure S25.3.

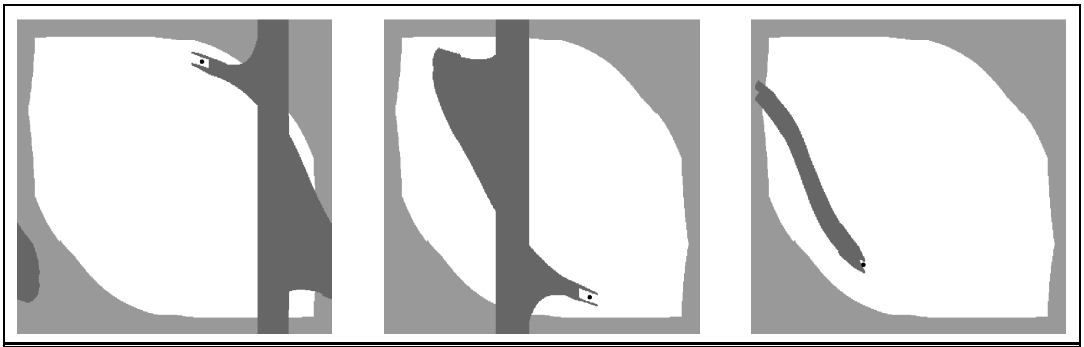


Figure S25.3 Configuration of the robots.

b. Figure S25.3 also answers the second part of this exercise: it shows the configuration space of the robot arm constrained by the self-collision constraint and the constraint imposed by the obstacle.

c. The three workspace obstacles are shown in Figure S25.4.

d. This question is a great mind teaser that illustrates the difficulty of robot motion planning! Unfortunately, for an arbitrary robot, a planar obstacle can decompose the workspace into *any* number of disconnected subspaces. To see, imagine a 1-DOF rigid robot that moves on a horizontal rod, and possesses N upward-pointing fingers, like a giant fork.

A single planar obstacle protruding vertically into one of the free-spaces between the fingers could effectively separate the configuration space into $N+1$ disjoint subspaces. A second DOF will not change this.

More interesting is the robot arm used as an example throughout this book. By slightly extending the vertical obstacles protruding into the robot's workspace we can decompose the configuration space into five disjoint regions. The following figures show the configuration space along with representative configurations for each of the five regions.

Is five the maximum for any planar object that protrudes into the workspace of this

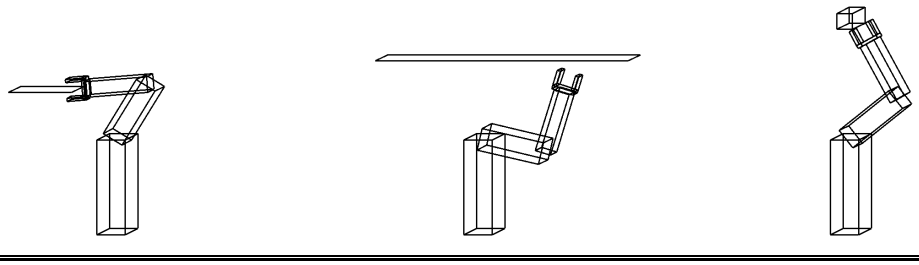


Figure S25.4 Workspace obstacles.

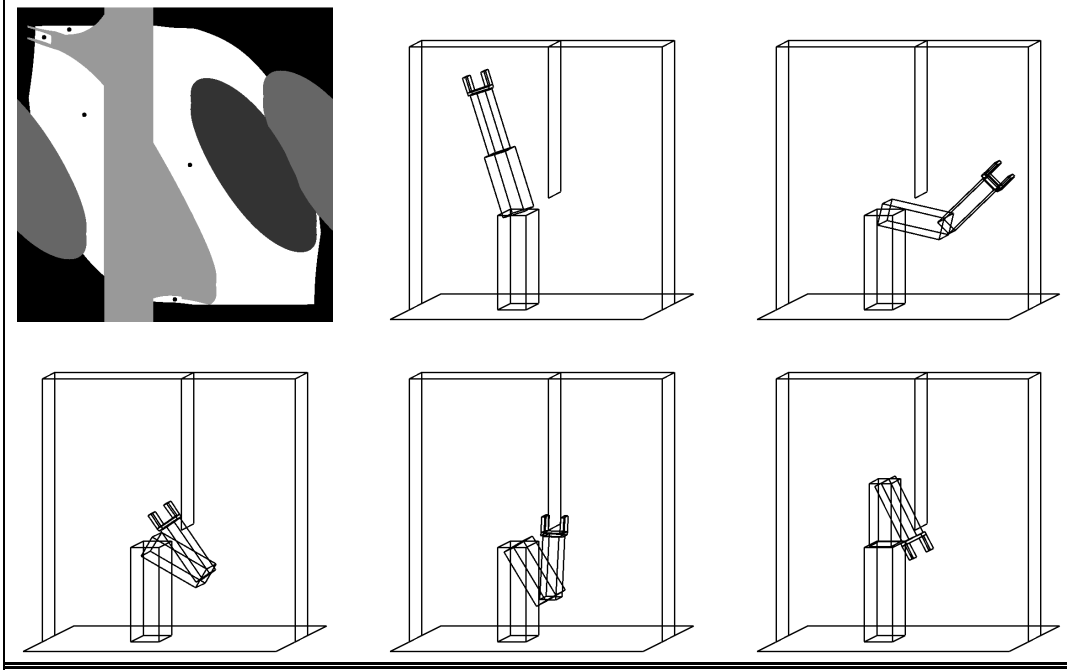


Figure S25.5 Configuration space for each of the five regions.

particular robot arm? We honestly do not know; but we offer a $1 reward for the first person who presents to us a solution that decomposes the configuration space into six, seven, eight, nine, or ten disjoint regions. For the reward to be claimed, all these regions must be clearly disjoint, and they must be a two-dimensional manifold in the robot's configuration space.

For non-planar objects, the configuration space is easily decomposed into any number of regions. A circular object may force the elbow to be just about maximally bent; the resulting workspace would then be a very narrow pipe that leave the shoulder largely unconstrained, but confines the elbow to a narrow range. This pipe is then easily chopped into pieces by small dents in the circular object; the number of such dents can be increased without bounds.

25.8

A. $x = 1 \cdot cos(60°) + 2 \cdot cos(85°) =$.
$y = 1 \cdot sin(60°) + 2 \cdot sin(85°) =$.
$\phi = 90°$

B. The minimal value of x is $1 \cdot \cos(70°) + 2 \cdot \cos(105°) = -0.176$
achieved when the first rotation is actually $70°$ and the second is actually $35°$.
The maximal value of x is $1 \cdot \cos(50°) + 2 \cdot \cos(65°) = 1.488$
achieved when the first rotation is actually $50°$ and the second is actually $15°$.
The minimal value of y is $1 \cdot \sin(50°) + 2 \cdot \sin(65°) = 2.579$
achieved when the first rotation is actually $50°$ and the second is actually $15°$.
The maximal value of y is $1 \cdot \sin(70°) + 2 \cdot \sin(90°) = 2.94$
achieved when the first rotation is actually $70°$ and the second is actually $20°$.
The minimal value of ϕ is $65°$ achieved when the first rotation is actually $50°$ and the second is actually $15°$.
The maximal value of ϕ is $105°$ achieved when the first rotation is actually $70°$ and the second is actually $35°$.

C. The maximal possible y-coordinate (1.0) is achieved when the rotation is executed at exactly $90°$. Since it is the maximal possible value, it cannot be the mean value. Since there is a maximal possible value, the distribution cannot be a Gaussian, which has non-zero (though small) probabilities for all values.

25.9 A simple deliberate controller might work as follows: Initialize the robot's map with an empty map, in which all states are assumed to be navigable, or free. Then iterate the following loop: Find the shortest path from the current position to the goal position in the map using A*; execute the first step of this path; sense; and modify the map in accordance with the sensed obstacles. If the robot reaches the goal, declare success. The robot declares failure when A* fails to find a path to the goal. It is easy to see that this approach is both complete and correct. The robot always find a path to a goal if one exists. If no such path exists, the approach detects this through failure of the path planner. When it declares failure, it is indeed correct in that no path exists.

A common reactive algorithm, which has the same correctness and completeness property as the deliberate approach, is known as the BUG algorithm. The BUG algorithm distinguishes two modes, the boundary-following and the go-to-goal mode. The robot starts in go-to-goal mode. In this mode, the robot always advances to the adjacent grid cell closest to the goal. If this is impossible because the cell is blocked by an obstacle, the robot switches to the boundary-following mode. In this mode, the robot follows the boundary of the obstacle until it reaches a point on the boundary that is a local minimum to the straight-line distance to the goal. If such a point is reached, the robot returns to the go-to-goal mode. If the robot reaches the goal, it declares success. It declares failure when the same point is reached twice, which can only occur in the boundary-following mode. It is easy to see that the BUG algorithm is correct and complete. If a path to the goal exists, the robot will find it. When the robot declares failure, no path to the goal may exist. If no such path exists, the robot will ultimately reach the same location twice and detect its failure.

Both algorithms can cope with continuous state spaces provides that they can accurately perceive obstacles, plan paths around them (deliberative algorithm) or follow their boundary (reactive algorithm). Noise in motion can cause failures for both algorithms, especially if the robot has to move through a narrow opening to reach the goal. Similarly, noise in perception destroys both completeness and correctness: In both cases the robot may erroneously conclude a goal cannot be reached, just because its perception was noise. However, a deliberate algorithm might build a probabilistic map, accommodating the uncertainty that arises from the noisy sensors. Neither algorithm as stated can cope with unknown goal locations; however, the deliberate algorithm is easily converted into an *exploration* algorithm by which the robot always moves to the nearest unexplored location. Such an algorithm would be complete and correct (in the noise-free case). In particular, it would be guaranteed to find and reach the goal when reachable. The BUG algorithm, however, would not be applicable. A common reactive technique for finding a goal whose location is unknown is random motion; this algorithm will with probability one find a goal if it is reachable; however, it is unable to determine when to give up, and it may be highly inefficient. Moving obstacles will cause problems for both the deliberate and the reactive approach; in fact, it is easy to design an adversarial case where the obstacle always moves into the robot's way. For slow-moving obstacles, a common deliberate technique is to attach a timer to obstacles in the grid, and erase them after a certain number of time steps. Such an approach often has a good chance of succeeding.

25.10 There are a number of ways to extend the single-leg AFSM in Figure 25.22(b) into a set of AFSMs for controlling a hexapod. A straightforward extension—though not necessarily the most efficient one—is shown in the following diagram. Here the set of legs is divided into two, named A and B, and legs are assigned to these sets in alternating sequence. The top level controller, shown on the left, goes through six stages. Each stage lifts a set of legs, pushes the ones still on the ground backwards, and then lowers the legs that have previously been lifted. The same sequence is then repeated for the other set of legs. The corresponding single-leg controller is essentially the same as in Figure 25.22(b), but with added wait-steps for synchronization with the coordinating AFSM. The low-level AFSM is replicated six times, once for each leg.

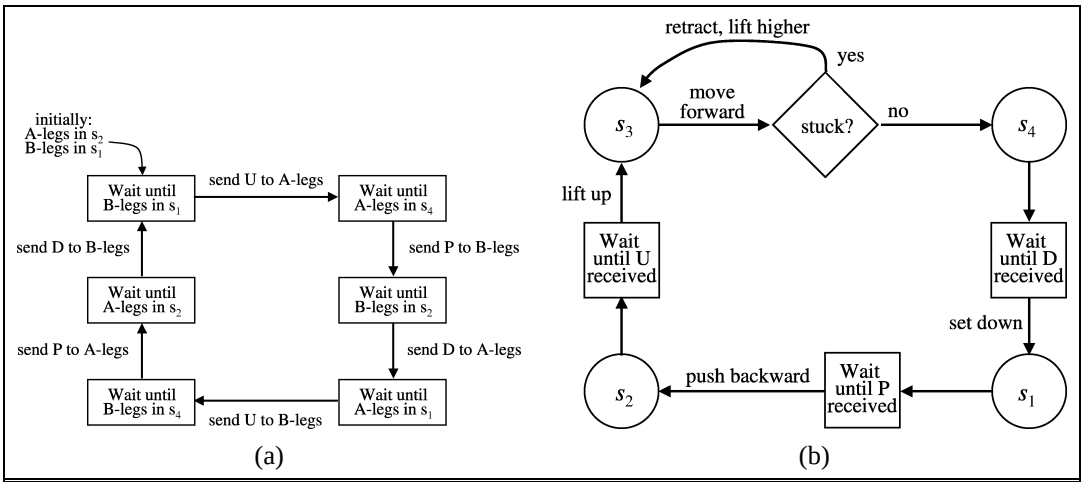


Figure S25.6 Controller for a hexapod robot.

For showing that this controller is stable, we show that at least one leg group is on the ground at all times. If this condition is fulfilled, the robot's center of gravity will always be above the imaginary triangle defined by the three legs on the ground. The condition is easily proven by analyzing the top level AFSM. When one group of legs in s_4 (or on the way to s_4 from s_3), the other is either in s_2 or s_1, both of which are on the ground. However, this proof only establishes that the robot does not fall over when on flat ground; it makes no assertions about the robot's performance on non-flat terrain. Our result is also restricted to *static stability*, that is, it ignores all dynamic effects such as inertia. For a fast-moving hexapod, asking that its center of gravity be enclosed in the triangle of support may be insufficient.

25.11 We have used this exercise in class to great effect. The students get a clearer picture of why it is hard to do robotics. The only drawback is that it is a lot of fun to play, and thus the students want to spend a lot of time on it, and the ones who are just observing feel like they are missing out. If you have laboratory or TA sections, you can do the exercise there.

Bear in mind that being the Brain is a very stressful job. It can take an hour just to stack three boxes. Choose someone who is not likely to panic or be crushed by student derision. Help the Brain out by suggesting useful strategies such as defining a mutually agreed Hand-centric coordinate system so that commands are unambiguous. Almost certainly, the Brain will start by issuing absolute commands such as "Move the Left Hand 12 inches positive y direction" or "Move the Left Hand to (24,36)." Such actions will never work. The most useful "invention" that students will suggest is the guarded motion discussed in Section 25.5—that is, macro-operators such as "Move the Left Hand in the positive y direction until the eyes say the red and green boxes are level." This gets the Brain out of the loop, so to speak, and speeds things up enormously.

We have also used a related exercise to show why robotics in particular and algorithm design in general is difficult. The instructor uses as props a doll, a table, a diaper and some safety pins, and asks the class to come up with an algorithm for putting the diaper on the

baby. The instructor then follows the algorithm, but interpreting it in the least cooperative way possible: putting the diaper on the doll's head unless told otherwise, dropping the doll on the floor if possible, and so on.

Solutions for Chapter 26
Philosophical Foundations

26.1 We will take the disabilities (see page 949) one at a time. Note that this exercise might be better as a class discussion rather than written work.

a. *be kind*: Certainly there are programs that are polite and helpful, but to be kind requires an intentional state, so this one is problematic.

b. *resourceful*: Resourceful means "clever at finding ways of doing things." Many programs meet this criteria to some degree: a compiler can be clever making an optimization that the programmer might not ever have thought of; a database program might cleverly create an index to make retrievals faster; a checkers or backgammon program learns to play as well as any human. One could argue whether the machines are "really" clever or just seem to be, but most people would agree this requirement has been achieved.

c. *beautiful*: Its not clear if Turing meant to be beautiful or to create beauty, nor is it clear whether he meant physical or inner beauty. Certainly the many industrial artifacts in the New York Museum of Modern Art, for example, are evidence that a machine can be beautiful. There are also programs that have created art. The best known of these is chronicled in *Aaron's code: Meta-art, artificial intelligence, and the work of Harold Cohen* (McCorduck, 1991).

d. *friendly* This appears to fall under the same category as *kind*.

e. *have initiative* Interestingly, there is now a serious debate whether software should take initiative. The whole field of software agents says that it should; critics such as Ben Schneiderman say that to achieve predictability, software should only be an assistant, not an autonomous agent. Notice that the debate over whether software *should* have initiative presupposes that it *has* initiative.

f. *have a sense of humor* We know of no major effort to produce humorous works. However, this seems to be achievable in principle. All it would take is someone like Harold Cohen who is willing to spend a long time tuning a humor-producing machine. We note that humorous text is probably easier to produce than other media.

g. *tell right from wrong* There is considerable research in applying AI to legal reasoning, and there are now tools that assist the lawyer in deciding a case and doing research. One could argue whether following legal precedents is the same as telling right from wrong, and in any case this has a problematic conscious aspect to it.

h. *make mistakes* At this stage, every computer user is familiar with software that makes mistakes! It is interesting to think back to what the world was like in Turing's day, when some people thought it would be difficult or impossible for a machine to make mistakes.

i. *fall in love* This is one of the cases that clearly requires consciousness. Note that while some people claim that their pets love them, and some claim that pets are not conscious, I don't know of anybody who makes both claims.

j. *enjoy strawberries and cream* There are two parts to this. First, there has been little to no work on taste perception in AI (although there has been related work in the food and perfume industries; see http://198.80.36.88/popmech/tech/U045O.html for one such artificial nose), so we're nowhere near a breakthrough on this. Second, the "enjoy" part clearly requires consciousness.

k. *make someone fall in love with it* This criteria is actually not too hard to achieve; machines such as dolls and teddy bears have been doing it to children for centuries. Machines that talk and have more sophisticated behaviors just have a larger advantage in achieving this.

l. *learn from experience* Part VI shows that this has been achieved many times in AI.

m. *use words properly* No program uses words perfectly, but there have been many natural language programs that use words properly and effectively within a limited domain (see Chapters 22-23).

n. *be the subject of its own thought* The problematic word here is "thought." Many programs can process themselves, as when a compiler compiles itself. Perhaps closer to human self-examination is the case where a program has an imperfect representation of itself. One anecdote of this involves Doug Lenat's Eurisko program. It used to run for long periods of time, and periodically needed to gather information from outside sources. It "knew" that if a person were available, it could type out a question at the console, and wait for a reply. Late one night it saw that no person was logged on, so it couldn't ask the question it needed to know. But it knew that Eurisko itself was up and running, and decided it would modify the representation of Eurisko so that it inherits from "Person," and then proceeded to ask itself the question!

o. *have as much diversity of behavior as man* Clearly, no machine has achieved this, although there is no principled reason why one could not.

p. *do something really new* This seems to be just an extension of the idea of learning from experience: if you learn enough, you can do something really new. "Really" is subjective, and some would say that no machine has achieved this yet. On the other hand, professional backgammon players seem unanimous in their belief that TDGammon (Tesauro, 1992), an entirely self-taught backgammon program, has revolutionized the opening theory of the game with its discoveries.

26.2 This exercise depends on what happens to have been published lately. The NEWS and MAGS databases, available on many online library catalog systems, can be searched for keywords such as Penrose, Searle, Chinese Room, Dreyfus, etc. We found about 90

reviews of Penrose's books. Here are some excerpts from a fairly typical one, by Adam Schulman (1995).

> Roger Penrose, the distinguished mathematical physicist, has again entered the lists to rid the world of a terrible dragon. The name of this dragon is "strong artificial intelligence."
>
> Strong Al, as its defenders call it, is both a widely held scientific thesis and an ongoing technological program. The thesis holds that the human mind is nothing but a fancy calculating machine-"-a computer made of meat"–and that all thinking is merely computation; the program is to build faster and more powerful computers that will eventually be able to do everything the human mind can do and more. Penrose believes that the thesis is false and the program unrealizable, and he is confident that he can prove these assertions. . . .
>
> In Part I of Shadows of the Mind Penrose makes his rigorous case that human consciousness cannot be fully understood in computational terms. . . . How does Penrose prove that there is more to consciousness than mere computation? Most people will already find it inherently implausible that the diverse faculties of human consciousness–self-awareness, understanding, willing, imagining, feeling–differ only in complexity from the workings of, say, an IBM PC.

Students should have no problem finding things in this and other articles with which to disagree. The `comp.ai` Newsnet group is also a good source of rash opinions.

Dubious claims also emerge from the interaction between journalists' desire to write entertaining and controversial articles and academics' desire to achieve prominence and to be viewed as ahead of the curve. Here's one typical result— *Is Nature's Way The Best Way?*, Omni, February 1995, p. 62:

> Artificial intelligence has been one of the least successful research areas in computer science. That's because in the past, researchers tried to apply conventional computer programming to abstract human problems, such as recognizing shapes or speaking in sentences. But researchers at MIT's Media Lab and Boston University's Center for Adaptive Systems focus on applying paradigms of intelligence closer to what nature designed for humans, which include evolution, feedback, and adaptation, are used to produce computer programs that communicate among themselves and in turn learn from their mistakes. *Profiles In Artificial Intelligence, David Freedman*.

This is not an argument that AI is impossible, just that it has been unsuccessful. The full text of the article is not given, but it is implied that the argument is that evolution worked for humans, therefore it is a better approach for programs than is "conventional computer programming." This is a common argument, but one that ignores the fact that (a) there are many possible solutions to a problem; one that has worked in the past may not be the best in the present (b) we don't have a good theory of evolution, so we may not be able to duplicate human evolution, (c) natural evolution takes millions of years and for almost all animals does not result in intelligence; there is no guarantee that artificial evolution will do better (d) artificial evolution (or genetic algorithms, ALife, neural nets, etc.) is not the only approach that involves feedback, adaptation and learning. "Conventional" AI does this as well.

26.3 Yes, this is a legitimate objection. Remember, the point of restoring the brain to normal (page 957) is to be able to ask "What was it like during the operation?" and be sure of

getting a "human" answer, not a mechanical one. But the skeptic can point out that it will not do to replace each electronic device with the corresponding neuron that has been carefully kept aside, because this neuron will not have been modified to reflect the experiences that occurred while the electronic device was in the loop. One could fix the argument by saying, for example, that each neuron has a single activation energy that represents its "memory," and that we set this level in the electronic device when we insert it, and then when we remove it, we read off the new activation energy, and somehow set the energy in the neuron that we put back in. The details, of course, depend on your theory of what is important in the functional and conscious functioning of neurons and the brain; a theory that is not well-developed so far.

26.4 To some extent this question illustrates the slipperiness of many of the concepts used in philosophical discussions of AI. Here is our best guess as to how a philosopher would answer this question. Remember that "wide content" refers to meaning ascribed by an outside observer with access to both brain and world, while narrow content refers to the brain state only. So the obvious answer seems to be that under wide content the states of the running program correspond to "having the goal of proving citizenship of the user," "having the goal of establishing the country of birth of the user," "knowing the user was born in the Isle of Man," and so on; and under narrow content, the program states are just arbitrary collections of bits with no obvious semantics in commonsense terms. (After all, the same compiled program might arise from an isomorphic set of rules about whether mushrooms are poisonous.) Many philosophers might object, however, that even under wide content the program has no such semantics because it never had the right kinds of causal connections to experience of the world that underpins concepts such as birth and citizenship.

26.5 The progress that has been made so far — a limited class of restricted cognitive activities can be carried out on a computer, some much better than humans, most much worse than humans — is very little evidence. If all cognitive activities can be explained in computational terms, then that would at least establish that cognition does not *require* the involvement of anything beyond physical processes. Of course, it would still be possible that something of the kind is *actually* involved in human cognition, but this would certainly increase the burden of proof on those who claim that it is.

26.6 The impact of AI has thus far been extremely small, by comparison. In fact, the social impact of *all* technological advances between 1958 and 2008 has been considerably smaller than the technological advances between 1890 and 1940. The common idea that we live in a world where technological change advances ever more rapidly is out-dated.

26.7 This question asks whether our obsession with intelligence merely reflects our view of ourselves as distinct due to our intelligence. One may respond in two ways. First, note that we already have ultrafast and ultrastrong machines (for example, aircraft and cranes) but they have not changed everything—only those aspects of life for which raw speed and strength are important. Good's argument is based on the view that intelligence is important in all aspects of life, since all aspects involve choosing how to act. Second, note that ultraintelligent machines have the special property that they can easily create ultrafast and ultrastrong machines

as needed, whereas the converse is not true.

26.8 It is hard to give a definitive answer to this question, but it can provoke some interesting essays. Many of the threats are actually problems of computer technology or industrial society in general, with some components that can be magnified by AI—examples include loss of privacy to surveillance, and the concentration of power and wealth in the hands of the most powerful. As discussed in the text, the prospect of robots taking over the world does not appear to be a serious threat in the foreseeable future.

26.9 Biological and nuclear technologies provide mush more immediate threats of weapons, yielded either by states or by small groups. Nanotechnlogy threatens to produce rapidly reproducing threats, either as weapons or accidently, but the feasibility of this technology is still quite hypothetical. As discussed in the text and in the previous exercise, computer technology such as centralized databases, network-attached cameras, and GPS-guided weapons seem to pose a more serious portfolio of threats than AI technology, at least as of today.

26.10 To decide if AI is impossible, we must first define it. In this book, we've chosen a definition that makes it easy to show it is possible in theory—for a given architecture, we just enumerate all programs and choose the best. In practice, this might still be infeasible, but recent history shows steady progress at a wide variety of tasks. Now if we define AI as the production of agents that act indistinguishably form (or at least as intellgiently as) human beings on any task, then one would have to say that little progress has been made, and some, such as Marvin Minsky, bemoan the fact that few attempts are even being made. Others think it is quite appropriate to address component tasks rather than the "whole agent" problem. Our feeling is that AI is neither impossible nor a ooming threat. But it would be perfectly consistent for someone to ffel that AI is most likely doomed to failure, but still that the risks of possible success are so great that it should not be persued for fear of success.

Bibliography

Andersson, R. L. (1988). *A robot ping-pong player: Experiment in real-time intelligent control*. MIT Press.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Bertoli, P., Cimatti, A., Roveri, M., and Traverso, P. (2001). Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI-01*, pp. 473–478.

Binder, J., Murphy, K., and Russell, S. J. (1997). Space-efficient inference in dynamic probabilistic networks. In *IJCAI-97*, pp. 1292–1296.

Chomsky, N. (1957). *Syntactic Structures*. Mouton.

Cormen, T. H., Leiserson, C. E., and Rivest, R. (1990). *Introduction to Algorithms*. MIT Press.

Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of A*. *JACM*, *32*(3), 505–536.

Elkan, C. (1997). Boosting and naive Bayesian learning. Tech. rep., Department of Computer Science and Engineering, University of California, San Diego.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning about Knowledge*. MIT Press.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, *10*, 447–474.

Heinz, E. A. (2000). *Scalable search in computer chess*. Vieweg.

Held, M. and Karp, R. M. (1970). The traveling salesman problem and minimum spanning trees. *Operations Research*, *18*, 1138–1162.

Kay, M., Gawron, J. M., and Norvig, P. (1994). *Verbmobil: A Translation System for Face-To-Face Dialog*. CSLI Press.

Kearns, M. and Vazirani, U. (1994). *An Introduction to Computational Learning Theory*. MIT Press.

Knuth, D. E. (1975). An analysis of alpha–beta pruning. *AIJ*, *6*(4), 293–326.

Lambert, K. (1967). Free logic and the concept of existence. *Notre Dame Journal of Formal Logic*, *8*(1–2).

McAfee, R. P. and McMillan, J. (1987). Auctions and bidding. *Journal of Economic Literature*, *25*(2), 699–738.

McCorduck, P. (1991). *Aaron's code: Meta-art, artificial intelligence, and the work of Harold Cohen*. W. H. Freeman.

Milgrom, P. (1989). Auctions and bidding: A primer. *Journal of Economic Perspectives*, *3*(3), 3–22.

Milgrom, P. R. and Weber, R. J. (1982). A theory of auctions and competitive bidding. *Econometrica*, *50*(5), 1089–1122.

Mohr, R. and Henderson, T. C. (1986). Arc and path consistency revisited. *AIJ*, *28*(2), 225–233.

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping—Reinforcement learning with less data and less time. *Machine Learning*, *13*, 103–130.

Mostowski, A. (1951). A classification of logical systems. *Studia Philosophica*, *4*, 237–274.

Norvig, P. (1992). *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Quine, W. V. (1960). *Word and Object*. MIT Press.

Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer-Verlag.

Schulman, A. (1995). Shadows of the mind: A search for the missing science of consciousness (book review). *Commentary*, *99*, 66–68.

Shanahan, M. (1999). The event calculus explained. In Wooldridge, M. J. and Veloso, M. (Eds.), *Artificial Intelligence Today*, pp. 409–430. Springer-Verlag.

Smith, D. E., Genesereth, M. R., and Ginsberg, M. L. (1986). Controlling recursive inference. *AIJ*, *30*(3), 343–389.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*(3–4), 257–277.

Vickrey, W. (1961). Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, *16*, 8–37.

Wahlster, W. (2000). *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer Verlag.