(d) *Yes or No.* Professor Digit also says that because its causal model is naive, this knowledge base generally makes insufficient use of information about whether the horn is disconnected. Is he correct? Explain briefly.

(e) Continuing the case, suppose further that the event-qualifying knowledge was obtained that the generator was new. Assume that this means that the generator is unlikely to be defective or badly installed. How would that information be used by MOLE?

(f) *Yes or No.* Professor Digit says that the battery could be dead even if the generator is not broken and that the battery could be working even if the generator is broken. He complains that the network does not represent these possibilities. Is he correct? Explain briefly.

Ex. 3 [10] *Dependency-Directed Backtracking in MOLE.* The MOLE performance program for diagnosis is based on classification.

(a) What kinds of assumptions does MOLE make to simplify its search?

(b) What steps in MOLE's method could cause it to discover that its assumptions are wrong for a particular problem?

Ex. 4 [10] *Distributed Classification Specialists.* The MDX application illustrates an implementation of classification where there is a subprogram for every possible solution. Briefly, how could the modules be reorganized to admit a data-driven approach to classification?

Assuming that the system considers only those hypotheses that cover data that have been acquired, there are some control issues. Discuss one of them briefly.

Ex. 5 [05] *Knowledge Beats Search?* Professor Digit tries to explain the knowledge principle. He decides to use MYCIN as an example. As he says, "The key to better performance in MYCIN is more knowledge, not better search. Furthermore, no one would confuse MYCIN's inference methods (mostly backward chaining) with powerful and sophisticated approaches for search and inference. This proves that knowledge is more important than search."

(a) Why is it confusing to say that MYCIN's search method is backward chaining? Did MYCIN eliminate search?

(b) Briefly, what's wrong with Professor Digit's claim about the supremacy of knowledge over search?

7.4 Knowledge and Methods for Classification

In the preceding sections we considered several knowledge systems that use classification. We presented the basic conjunctive classification model and discussed the kinds of knowledge that are brought to bear in the example applications. This section refers back to our basic models for configuration and the case studies to show what is involved in knowledge-level and symbol-level analyses of classification domains. It then sketches and compares methods for classification tasks. Our goal is to show how variations in the requirements of classification can be accommodated by changes in method. The issues that arise in creating specialized methods for classification occur for other kinds of search. This section shows how general symbol-level and knowledge-level issues interact with the task requirements to shape the specialized methods for classification.

The methods for classification have two challenges: (1) They must provide a framework for organizing and maintaining the different kinds of knowledge that can be brought to bear in a task, and (2) they must accommodate different assumptions about classification, which are im-

posed by the larger tasks to which classification is in service. In the following, we introduce sketches of classification methods. Our goal is more to illustrate issues than to provide detailed algorithms. To simplify the presentation, we follow the approach of basic classification methods where solutions are made from singleton classes. That is, none of the methods handles composite solutions. We also assume that the search is exhaustive. Variations are left to the reader.

7.4.1 Knowledge-Level and Symbol-Level Analysis of Classification Domains

It is useful to distinguish between knowledge-level analysis and symbol-level analysis. Knowledge-level analysis is concerned with the kinds of knowledge required for a task. Symbol-level analysis is concerned with alternate implementations involving both representation and search.

Knowledge-Level Analysis

A knowledge-level analysis of a task domain describes the kinds of knowledge that are needed and how they are used. In this section, however, we are not interested in a single task domain, but rather in tasks modeled using classification. In our analysis, we draw on examples from the task domains described in our case studies. The key kinds of relations used in classification tasks are illustrated in the **horseshoe diagram**: relations that abstract data, relations that map abstract data to or from abstract classes in the solution space, and refinement relations in the solution space. We consider these three kinds of relations in turn.

In his characterization of classification, Clancey (1985) recognized three categories of **abstraction relations**. A **definitional abstraction** focuses on necessary and essential features of an abstraction of the data. ("If a bacterium can live in an environment where there is no free oxygen, then it is an anaerobic bacterium.") A **qualitative abstraction** reduces and simplifies quantitative data, such as by comparing it to an expected value. ("If the patient is an adult and white-blood-cell count is less than 2,500, then the white-blood-cell count is low.") A **generalization abstraction** reduces a description to a more general case in a subtype hierarchy. ("If the person is a father, then the person is a man.") Data abstractions can be about a wide range of kinds of data including temporal data, anatomical data, and physiological data.

Our categories of abstraction relations are not exhaustive or mutually exclusive. They do not cover all the abstraction relations that can be useful in hierarchical classification, and there are cases where the choice of which term to use in modeling is ambiguous. The categories are not a fixed part of classification theory or the epistemology of relations. Nonetheless, these categories of abstraction relations have a certain practical relevance because they exemplify broad ways of starting from data in problems that are grounded in the real world. These categories of abstraction relations are representative of relations established by knowledge-guided reasoning seen in many classification systems. What they have in common is that they map data from specific descriptions to more abstract descriptions and that they tend to be highly certain.

Class-matching relations are connections that relate available data to classes of solutions. In the most straightforward case, these are the heuristic matching relations that lead from observed data to solution classes. However, class-matching relations can be used in either direction. They can be used to validate hypotheses about solution classes, guiding the selection of data. They can also be used to match possible classes based on given data. The latter case applies when there are enough data for testing only a subset of the criteria associated with a class. If fur-

ther assumptions are not allowed, a class cannot be established unless we know that all of its criteria are satisfied. Some of the methods in the next section often use matches on partial data to suggest "possible" solution classes, short of establishing the applicability of the classes.

Some regularities appear in the categories of matching relations that have been used in classification programs. A **causal association** explains data in terms of mechanisms involving causes and their direct effects. In an **empirical correlation**, the data and the solutions are often associated with each other in typical cases, but no causal mechanism is understood. For example, in a medical situation it may be known that people of a certain age or of a particular region are predisposed to having a particular problem. The utility of this knowledge is only statistical and not necessarily predictive for individual cases as more data are acquired. Empirical relations are not based on an understanding of mechanism, and in diagnostic tasks they may not suggest anything more specific than a symptomatic treatment. In intermediate cases, a direction of causality may be known but the conditions of the process may not.

As with our categories of abstraction relations, the categories of matching relations do not provide a solid epistemology for a theory of classification. Also as before, they have a certain practical value. They capture typical, useful connections for problem solving, organized to provide efficient indexing of solutions from available data. As a group, matching associations tend to be less reliable than the inferences about abstraction.

Refinement relations connect broad characterizations of solutions to more specific ones. These relations can be the inverses of abstraction relations, located in the solution space rather than in the data space. For example, a refinement mapping from *enterobacteriaceae* (bacteria normally found in the digestive tract) to *E. coli* (a kind of bacterium) is a subtype relation. Alternatively, refinement may trace a component hierarchy, following part-of relations to smaller system modules. Still other refinement relations may trace out increasingly localized areas of a physical or logical space. Still others may simply reason toward smaller sets, where there is a subset relation but no descriptive subtype relation beyond membership. What all refinement mappings have in common is a path toward greater specificity in determining solutions. Hierarchical solution spaces are composed through these relations.

Classification models are most typically used together with other models in characterizing a task domain. In an important special case, multistage classification models are used together to characterize a single task. Characterizing a classification model in terms of separate stages can increase clarity by partitioning both the spaces of interest and the kinds of knowledge that are used. In our earlier discussion we described two views of a multistage model. The composed mappings view emphasizes the idea that the same relations for abstraction, heuristic match, and refinement can occur in multiple stages. This view is especially useful for visualizing classification problems in stages where the solutions of one stage map to the data of the next.

Alternatively, the intermediate spaces view emphasizes how classification can link intermediate spaces of different character. Each space may have unique properties, and this view makes it easier to show the differences among the spaces. This latter view is helpful when intermediate spaces include additional kinds of reasoning beyond classification, such as temporal or causal reasoning. Chapter 9 discusses the CASNET/GLAUCOMA system among its case studies of medical diagnosis systems. The intermediate spaces view of CASNET is particularly useful in visualizing how it works.

Symbol-Level Analysis

A symbol-level analysis is about implementation. There are many possibilities for representing data and hypotheses in classification systems, including parallel processing approaches that compute solutions in parallel. Since representation and reasoning topics are discussed in detail in other parts of this book, we limit our discussion here to a few general words about how search can be governed in classification methods.

To be effective, all classification methods must routinely rule out most of the solution space. When a task is based on a classification model, it imposes a search model to the spaces that describe the domain. There are three basic approaches.

- A **data-directed search** starts with data and searches for possible solutions. It matches data against solutions, stopping when all relevant inferences have been made. This approach is also called bottom up where solutions and data correspond to the root and leaves of a search tree, respectively. The MORE and MOLE systems are largely data directed.
- A **solution-directed search** works back from possible solutions to explanatory data. This is also called top-down or goal-driven search. The MYCIN and MDX systems are mostly solution directed. The PROSPECTOR system starts out in a data-directed mode and then switches to a solution-directed mode.
- An **opportunistic search** works in both directions as information becomes available to trigger new inferences or to guide the search. New data can trigger a search for immediate abstractions. Heuristic rules can pose hypotheses, which are then tested under a focused hypothesis-directed search.

Given these basic variations in search, systems can make different assumptions about the data. In the simplest version of classification, all the classes that match the data are reportable as solutions and the others are not. Going beyond this simplest case brings us at once to more complex variations in implementation.

Some tasks assume that a complete listing of all relevant data is given as input at the beginning, implicitly discounting the possibility that data could be "missing." Other tasks assume that some data may be missed or left out accidentally. In this latter case, methods must seek additional data during the classification process or make assumptions about typical values for data.

Some tasks assume that all given data are reliable. Other methods assume that some data are more reliable than others. In the latter case, methods must evaluate the preponderance of evidence in classification.

All methods explicitly rule out classes when they violate the class criteria. Some rule out classes implicitly, based on mutual exclusion relations for classes for which there is positive evidence or by parsimony rules that weigh the preponderance of evidence. For example, on definitional grounds high blood pressure is enough to rule out low blood pressure. In many domains, such default and exclusion reasoning can lead to cases where there is an apparent contradiction in the interpretation of evidence. For example, there can be data supporting two classes that are supposed to be mutually exclusive. In another example, data acquired later may contradict the choice of classes suggested by applying parsimony rules to the data first acquired. In either case,

these methods need to account for what to do in such cases, favoring one interpretation over another or simply reporting a contradiction. In the case where competing and possibly contradictory solutions need to be compared and reported, the system may apply techniques of truth maintenance or backtracking to keep track of competing alternatives.

7.4.2 MC-1: A Strawman Generate-and-Test Method

We begin in Figure 7.33 with an extremely simple "method" for classification, called MC-1, which is based on generate-and-test. The quotation marks around the term *method* are intended to emphasize that MC-1 is used more as a strawman to illustrate basic assumptions rather than as a practical method for typical applications. MC-1 simply steps through the set of possible solutions and tests whether any solution matches the data. It has a solution tester, which compares candidates against the data, rejecting ones that are consistent. MC-1 is formulated as **exhaustive generate-and-test**, with no ranking of multiple solutions. All the knowledge used by MC-1 is embodied in the predefined set of solutions and in the matching and evaluation criteria used by the solution tester. The solution tester itself may organize and represent the set of evaluation criteria for the classes in an explicit discrimination network or in any of a number of other ways.

MC-1 relies on several assumptions about classification. It assumes:

- The set of solution classes is small enough to consider each class individually.
- All the necessary data can be obtained at the beginning of the process.

The first assumption corresponds to the usual caveat about complexity for generate-and-test methods. The second assumption concerns the system's ability to reason from partial data. In MC-1, we do not postulate any mechanisms for providing default data or for acquiring data incrementally.

None of our example applications of classification uses MC-1 because none of the applications satisfies these simplifying assumptions. Other task requirements dominate the design of the specialized methods. MC-1 does not employ any knowledge about the major kinds of data abstraction or solution refinement relations cited in our analysis of the applications, nor does it exploit any hierarchical structure of the solution space. In the rest of this section, we relax the assumptions of MC-1 a bit at a time and consider how the methods can be modified accordingly.

To perform classification using MC-1:

```
1.  Initialize the list of solutions to empty.
2.  Obtain the data and abstract it.
3.  For each candidate class do
4.        If solution-tester(candidate, data) then add the candidate to
          the list of solutions.
5.  Report the solutions.
```

FIGURE 7.33. MC-1, an extremely simple method for classification based on exhaustive generate-and-test.

As the requirements become more realistic, the search control and the required knowledge for classification become more complex.

7.4.3 MC-2: Driving from Data to Plausible Candidates

Our next method, MC-2, is also quite simple. MC-2 reduces the amount of computational work when there is a large number of solutions. Instead of stepping through all possible solutions, MC-2 limits the number of solutions considered by evaluating only those that potentially explain the data. It also reduces the cost per candidate by keeping track of which candidates have been considered already and by evaluating each candidate exactly once. It depends on the same assumptions as MC-1 except that the first assumption is replaced with the following:

- There is an efficient procedure for retrieving all the candidate solutions that can potentially explain a data element and that are consistent with it.
- The given data are highly discriminatory, so that the set of candidate solutions is much smaller than the complete set of possible solutions.

The first assumption requires that there is a means of retrieving covering solutions from data. In MC-2, this is carried out by the candidate retriever. As discussed in Section 7.2, different policies are possible for the candidate retriever. For example, it could return all candidates that are consistent with the data or it could limit itself to consistent candidates that cover some of the data.

A data abstraction step is done before candidate retrieval. For example, a reading of blood pressure (a datum) may be abstracted to "high blood pressure." MC-2 uses a data abstractor and a solution tester as shown in Figure 7.34.

The data abstractor uses the abstraction operations discussed earlier in our knowledge-level analysis. It could use qualitative abstraction, temporal abstraction, spatial or part-whole

To perform classification using MC-2:

```
1.  Initialize the list of solutions to empty.
2.  Obtain the data.
3.  Abstract the data using the data-abstractor.

    /* Find solutions that cover the data. Different policies are
    possible for the candidate retriever. */
4.  candidates := candidate-retriever(data)

    /* Evaluate solutions on the full data. This is where knowledge for
    ranking candidates is used. */
5.  For each class on the list of candidates do
6.        If solution-tester(candidate, data) then add the candidate to
          the list of solutions.
7.  Return the list of solutions.
```

FIGURE 7.34. MC-2 is a simple variation of MC-1 that requires an efficient and reliable means (the candidate retriever) for retrieving solutions that could potentially cover data elements.

abstraction, or definitional abstraction. The data abstractor is a small classification system in its own right. There are many implementation choices here. For example, in some applications it may be convenient to formalize a language of abstractions and to write a rule grammar for recognizing them. There may be more than one set of possible abstractions for a set of data.

The second assumption for MC-2 is that the candidate retriever greatly reduces the number of candidate classes to be evaluated. If it returns too few solution candidates, then MC-2 may miss a correct solution. If it returns too many, then MC-2 will show no improvement in performance over MC-1. After the initial set of candidates is acquired, then the solution tester must compare competing solutions. In the absence of acquiring more data, it must rely on some other measures of the solutions beyond their consistency with data. It may draw on information about prior probabilities of the candidates. It could employ a single-solution assumption or other knowledge for ranking candidates.

Like MC-1, MC-2 is simpler than any of the methods for the sample applications. Nonetheless, its revised assumptions are closer to the ones used in the example applications. The first set of method specializations that we discuss below all use more knowledge to cope with a large number of possible solutions.

The methods MC-1 and MC-2 are variations of simple generate-and-test. Neither of them takes advantage of hierarchical relations on the set of solutions. As discussed in Chapter 2, the usual way to extend a search method for larger spaces is to organize the search space hierarchically.

7.4.4 *MC-3: Solution-Driven Hierarchical Classification*

Method MC-3 uses a hierarchical generate-and-test approach to classification. MC-3 assumes that the possible solutions are arranged in a class hierarchy. It traverses that hierarchy, using a top-down, breadth-first search. At each level, it compares candidates to data, enabling it to rule out branches of the hierarchy. It may request additional data to discriminate among candidates. It then proceeds to the next level along those branches that are consistent. This version assumes that solutions are leaf nodes and that other classes higher in the hierarchy are partial descriptions of solutions.

This process is illustrated in Figure 7.35 for the first two levels of a solution hierarchy. The shaded boxes are the solution classes that have been visited. The black boxes represent solution classes that have been ruled out, together with their descendants. In the data space, the boxes outlined with heavy lines represent the data that have been acquired.

MC-3 depends on several assumptions beyond those of MC-2:

- The solution classes are arranged in a superclass hierarchy or lattice that efficiently divides the classes of candidates.
- There is a subset of data useful for discrimination at each level of the solution space.
- The goal is to find the most-specific classes consistent with the data. (As in MC-2, additional evaluation criteria may be applied to rank competing candidates that are left after this.)

The first assumption is the key to MC-3's gain in efficiency. Subclasses are ruled out when their superclass is ruled out. If a class has multiple superclasses, it follows that the class can be elimi-

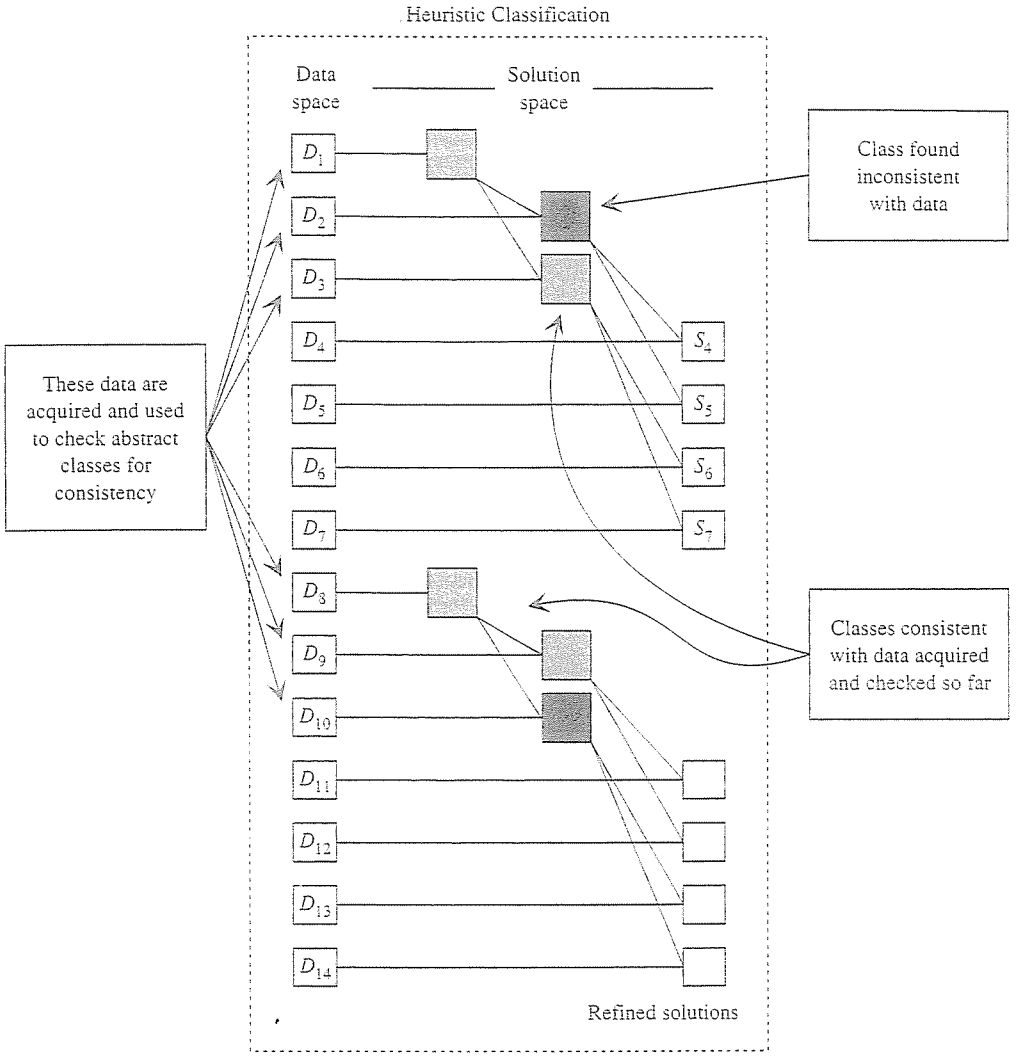

FIGURE 7.35. MC-3 traverses a hierarchy of classes. At each level it seeks data that will enable it to rule out branches of the taxonomy.

nated when any of its superclasses is ruled out. The assumption admits the early pruning condition, crucial for all applications of hierarchical generate-and-test. For the taxonomy to be effective, each level must partition the remaining candidates into distinct classes.

The third assumption explains why the method does not just stop when it has eliminated competing solutions at a level. The search is for the most-specific characterizations of solutions. Figure 7.36 presents the method. MC-3 is vague about the process used for selecting discriminatory data. Some theoretical measures of information entropy can be brought to bear on this question, as discussed in Chapter 9.

```
To perform classification using MC-3:
1.  Initialize the list of solutions to empty.
2.  For each level of the solution hierarchy do
3.  begin
4.        Obtain the raw data useful for discriminating among the
          candidates at this level.
5.        Abstract the data using the data-abstractor.
6.        For each class at this level do
7.        begin
8.              If the ~ solution-tester(candidate, data) then discard
                the candidate
9.              Else if the candidate is incomplete, then remove it from
                the list and add its successors
10.             Else add the candidate to the list of solutions.
11.       end
12. end
13. Rank the solutions and return them.
```

FIGURE 7.36. The second stage of MC-3 is to map a subset of the data to initial "seed" classes. Many of these classes will prove inconsistent with the data, leading to a selection of further data.

7.4.5 *MC-4: Data-Driven Hierarchical Classification*

MC-3 comes closer to the methods used in the example knowledge systems in this chapter, but it is still simplistic. One major difference is that MC-3 starts every case by asking about the same data. In a troubleshooting application, for example, this would be silly. It would fail to make good use of highly focused initial data provided by an observer reporting faulty behavior. Similarly, in a medical setting it is more typical to reason from the presenting symptoms.

Like MC-3, MC-4 uses a hierarchical search space. Like MC-2, it uses a data-driven approach that draws on such data. MC-4 begins by establishing "seed classes" suggested by data offered by an external source. Like MC-3, it rules out classes that are inconsistent with the data, but it uses the data it is given to guide the search and rules out classes that are consistent with the data if those classes do not bear on the data. In a medical task, if MC-4 were given that the patient has a swelling in the ankles, it would not consider diseases that fail to explain those symptoms, such as "broken arm" or "measles."

In the following we introduce the method and its assumptions in stages. Figure 7.37 illustrates the first part of MC-4: data acquisition and abstraction. We assume that an initial subset of data has been preselected and that it is abstracted before further use. The data items highlighted with boxes indicate those that have been preselected. Abstractions are computed from these data to yield input data to the next part of the classification process.

The next part of MC-4 matches the data against high-level classes in the taxonomy of candidate solutions. We assume that there are many possible classifications and that fine discrimination requires a wide range of different data. MC-4 distinguishes data according to the classes that