Heuristic Classification*

William J. Clancey
Stanford Knowledge Systems Laboratory, 701 Welch Road, Building C, Palo Alto, CA 94304, U.S.A.

Recommended by Allen Newell

ABSTRACT

A broad range of well-structured problems—embracing forms of diagnosis, catalog selection, and skeletal planning—are solved in 'expert systems' by the methods of heuristic classification. These programs have a characteristic inference structure that systematically relates data to a pre-enumerated set of solutions by abstraction, heuristic association, and refinement. In contrast with previous descriptions of classification reasoning, particularly in psychology, this analysis emphasizes the role of a heuristic in routine problem solving as a non-hierarchical, direct association between concepts. In contrast with other descriptions of expert systems, this analysis specifies the knowledge needed to solve a problem, independent of its representation in a particular computer language. The heuristic classification problem-solving model provides a useful framework for characterizing kinds of problems, for designing representation tools, and for understanding non-classification (constructive) problem-solving methods.

> *To understand something as a specific instance of a more general case—which is what understanding a more fundamental principle or structure means—is to have learned not only a specific thing but also a model for understanding other things like it that one may encounter.* [13]

1. Introduction

Over the past decade, a variety of heuristic programs, commonly called 'expert systems', have been written to solve problems in diverse areas of science, engineering, business, and medicine. Developing these programs involves satisfying an interacting set of requirements: Selecting the application area and specific problem to be solved, bounding the problem so that it is computationally and financially tractable, and implementing a prototype program—to name a few obvious concerns. With continued experience, a number of programming environments or 'tools' have been developed and successfully used to imple-

* Expanded version of "Classification Problem Solving", in: *Proceedings Fourth National Conference on Artificial Intelligence*, Austin, TX (August 1984) 49–55.

Artificial Intelligence **27** (1985) 289–350

ment prototype programs [51]. Importantly, the representational units of tools (such as 'rules' and 'attributes') provide an orientation for identifying manageable subproblems and organizing problem analysis. Selecting appropriate applications now often takes the form of relating candidate problems to known computational methods, our tools.

Yet, in spite of this experience, when presented with a given 'knowledge engineering tool', such as EMYCIN [94], we are still hard-pressed to say what kinds of problems it can be used to solve well. Various studies have demonstrated advantages of using one representation language instead of another—for ease in specifying knowledge relationships, control of reasoning, and perspicuity for maintenance and explanation [2, 3, 21, 27, 92]. Other studies have characterized in low-level terms why a given problem might be inappropriate for a given language, for example, because data are time-varying or subproblems interact [51]. While these studies reveal the weaknesses and limitations of the rule-based formalism, in particular, they do not clarify the form of analysis and problem decomposition that has been so successfully used in these programs. In short, attempts to describe a mapping between *kinds of problems* and programming languages have not been satisfactory because they don't describe what a given program *knows*: Applications-oriented descriptions like 'diagnosis' are too general (e.g., solving a diagnostic problem does not necessarily require a device model), and technological terms like 'rule-based' do not describe what kind of problem is being solved [48, 49]. We need a better description of what heuristic programs do and know—a computational characterization of their competence—independent of task and independent of programming language implementation. Logic has been suggested as a basis for a 'knowledge-level' analysis to specify what a heuristic program does and might know [69, 71]. However, we have lacked a set of terms and relations for doing this.

In an attempt to characterize the *knowledge-level competence* of a variety of expert systems, a number of programs were analyzed in detail.[1] There is a striking pattern: These programs proceed through easily identifiable phases of data abstraction, heuristic mapping onto a hierarchy of pre-enumerated solutions, and refinement within this hierarchy. In short, these programs do what is commonly called *classification*, but with the important twist of *relating concepts in different classification hierarchies by non-hierarchical, uncertain inferences.* We call this combination of reasoning *heuristic classification.*

Note carefully: The heuristic classification model characterizes *a form of knowledge and reasoning*—patterns of familiar problem situations and solutions, heuristically related. In capturing problem situations that tend to occur and solutions that tend to work, this knowledge is essentially *experiential*, with

[1] Including: Ten rule-based systems (MYCIN, PUFF, CLOT, HEADMED, SACON from the EMYCIN family [15], plus WINE, BANKER, The Drilling Advisor, and other proprietary systems developed at Teknowledge, Inc.), a frame-based system (GRUNDY), and a program coded directly in LISP (SOPHIE III).

an overall form that is problem-area-independent. Heuristic classification is a method of computation, not a kind of problem to be solved. Thus, we refer to 'the heuristic classification method', not 'classification problem'.

Focusing on epistemological content rather than representational notation, this paper proposes a set of terms and relations for describing the knowledge used to solve a problem by the heuristic classification method. Subsequent sections describe and illustrate the model in the analysis of MYCIN, SACON, GRUNDY, and SOPHIE III. Significantly, a knowledge-level description of these programs corresponds very well to psychological models of expert problem solving. This suggests that the heuristic classification problem-solving model captures general principles of how experiential knowledge is organized and used, and thus generalizes some cognitive science results. A thorough discussion relates the model to schema research; and use of a conceptual graph notation shows how the inference-structure diagram characteristic of heuristic classification can be derived from some simple assumptions about how data and solutions are typically related (Section 4). Another detailed discussion then considers "what gets selected", possible kinds of solution (e.g., diagnoses). A taxonomy of problem types is proposed that characterizes *solutions of problems* in terms of synthesis or analysis of some *system* in the world (Section 5). We finally turn to the issue of inference control in order to further characterize tool requirements for heuristic classification (Section 6), segueing into a brief description of constructive problem solving (Section 7).

This paper explores different perspectives for describing expert systems; it is not a conventional description of a particular program or programming language. The analysis does produce some specific and obviously useful results, such as a distinction between electronic and medical diagnosis programs (Section 6.2). But there are also a few essays with less immediate payoffs, such as the analysis of problem types in terms of systems (Section 5) and the discussion of the pragmatics of defining concepts (Section 4.5). Also, readers who specialize in problems of knowledge representation should keep in mind that the discussion of schemas (Section 4) is an attempt to clarify the knowledge represented in rule-based expert systems, rather than to introduce new representational ideas.

From another perspective, this paper presents a methodology for analyzing problems, preparatory to building an expert system. It introduces an intermediate level of knowledge specification, more abstract than specific concepts and relations, but still independent of implementation language. Indeed, one aim is to afford a level of awareness for describing expert system design that enables knowledge representation languages to be chosen and used more deliberately.

We begin with the motivation of wanting to formalize what we have learned about building expert systems. How can we classify problems? How can we select problems that are appropriate for our tools? How can we improve our tools? Our study reveals patterns in knowledge bases: Inference chains are not

arbitrary sequences of implications, they compose relations among concepts in a systematic way. Intuitively, we believe that understanding these high-level knowledge structures, implicitly encoded in today's expert systems, will enable us to teach people how to use representation languages more effectively, and also enable us to design better languages. Moreover, it is a well-established principle for designing these programs that the knowledge people are trying to express should be stated explicitly, so it will be accessible to auxiliary programs for explanation, teaching, and knowledge acquisition (e.g., [30]).

Briefly, our methodology for specifying the knowledge contained in an expert system is based on:

- a computational distinction between selection and construction of solutions;
- a relational breakdown of concepts, distinguishing between abstraction and heuristic association and between subtype and cause, thus revealing the classification nature of inference chains; and
- a categorization of problems in terms of synthesis and analysis of *systems in the world*, allowing us to characterize inference in terms of a sequence of classifications involving some system.

The main result of the study is the model of heuristic classification, which turns out to be a common problem-solving method in expert systems. Identifying this computational method is not to be confused with advocating its use. Instead, by giving it a name and characterizing it, we open the way to describing when it is applicable, contrasting it with alternative methods, and deliberately using it again when appropriate.

As one demonstration of the value of the model, classification in well-known medical and electronic diagnosis programs is described in some detail, contrasting different perspectives on what constitutes a diagnostic solution and different methods for controlling inference to derive *coherent* solutions. Indeed, an early motivation for the study was to understand how NEOMYCIN, a medical diagnostic program, could be generalized. The resulting tool, called HERACLES (roughly standing for 'Heuristic Classification Shell') is described briefly, with a critique of its capabilities in terms of the larger model that has emerged.

In the final sections of the paper, we reflect on the adequacy of current knowledge engineering tools, the nature of a knowledge-level analysis, and related research in psychology and artificial intelligence. There are several strong implications for the practice of building expert systems, designing new tools, and continued research in this field. Yet to be delivered, but promised by the model, are explanation and teaching programs tailored to the heuristic classification model, better knowledge acquisition programs, and demonstration that thinking in terms of heuristic classification makes it easier to choose problems and build new expert systems.

2. The Heuristic Classification Method Defined

We develop the idea of the heuristic classification method by starting with the

common sense notion of classification and relating it to the reasoning that occurs in heuristic programs.

2.1. Simple classification

As the name suggests, the simplest kind of classification is identifying some unknown object or phenomenon as a member of a known class of objects, events, or processes. Typically, these classes are stereotypes that are hierarchically organized, and the process of identification is one of matching observations of an unknown entity against features of known classes. A paradigmatic example is identification of a plant or animal, using a guidebook of features, such as coloration, structure, and size. MYCIN solves the problem of identifying an unknown organism from laboratory cultures by matching culture information against a hierarchy of bacteria (Fig. 2.1).[2]

The essential characteristic of classification is that the problem solver *selects* from a set of pre-enumerated solutions. This does not mean, of course, that the 'right answer' is necessarily one of these solutions, just that the problem solver will only attempt to match the data against the known solutions, rather than construct a new one. Evidence can be uncertain and matches partial, so the output might be a ranked list of hypotheses. Besides matching, there are several *rules of inference* for making assertions about solutions. For example, evidence for a class is indirect evidence that one of its subtypes is present.

2.2. Data abstraction

In the simplest problems, data are solution features, so the matching process is direct. For example, an unknown organism in MYCIN can be classified directly

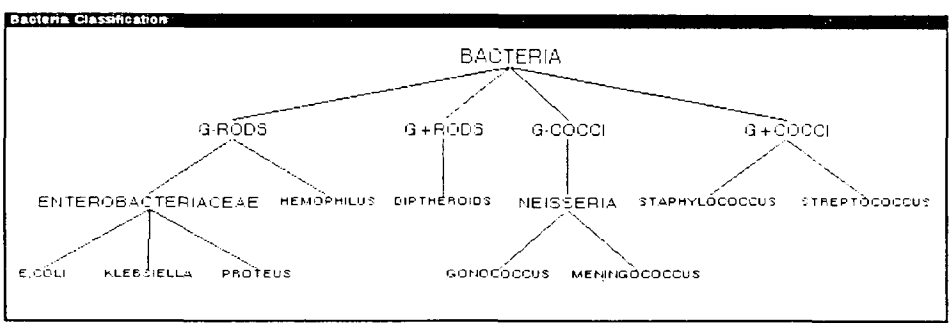


FIG. 2.1. MYCIN's classification of bacteria.

[2] For simplicity, we will refer to classification hierarchies throughout this paper, though in practice these structures are not trees, but almost always 'tangled' structures with some nodes having multiple parents.

given the supplied data of Gram stain and morphology. The features 'Gram-stain negative' and 'rod-shaped' match a class of organisms. The solution might be refined by getting information that allows subtypes to be discriminated.

For many problems, solution features are not supplied as data, but are inferred by *data abstraction.* There are three basic relations for abstracting data in heuristic programs:

(1) *definitional abstraction* based on essential, necessary features of a concept ("if the structure is a one-dimensional network, then its shape is a beam");

(2) *qualitative abstraction,* a form of definition involving quantitative data, usually with respect to some normal or expected value ("if the patient is an adult and white blood count is less than 2500, then the white blood count is low"); and

(3) *generalization* in a subtype hierarchy ("if the client is a judge, then he is an educated person").

These interpretations are usually made by the program with certainty; belief thresholds and qualifying conditions are chosen so the abstraction is categorical. It is common to refer to this knowledge as being 'factual' or 'definitional'.

2.3. Heuristic classification

In simple classification, data may directly match solution features or may match after being abstracted. In heuristic classification, solutions and solution features may also be matched *heuristically,* by direct, non-hierarchical association with some concept in *another* classification hierarchy. For example, MYCIN does more than identify an unknown organism in terms of visible features of an organism: MYCIN heuristically relates an abstract characterization of the patient to a classification of diseases. We show this *inference structure* schematically, followed by an example (Fig. 2.2).

Basic observations about the patient are abstracted to patient categories, which are heuristically linked to diseases and disease categories. While only a subtype link with E.coli infection is shown here, evidence may actually derive from a combination of inferences. Some data might directly match E.coli features (an individual organism shaped like a rod and producing a Gram-negative stain is seen growing in a culture taken from the patient). Descriptions of laboratory cultures (describing location, method of collection, and incubation) can also be related to the classification of diseases.

The important link we have added is a heuristic association between a characterization of the patient ('compromised host') and categories of diseases ('gram-negative infection'). Unlike definitional and hierarchical inferences, this inference makes a great leap. A *heuristic relation* is uncertain, based on assumptions of typicality, and is sometimes just a poorly understood correlation. A heuristic is often empirical, deriving from problem-solving

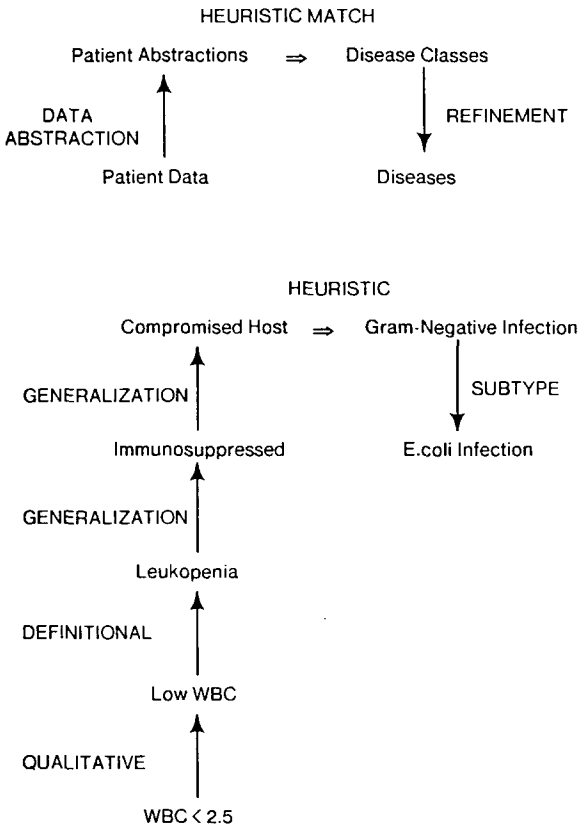


FIG. 2.2. Inference structure of MYCIN.

experience; heuristics correspond to the 'rules of thumb', often associated with expert systems [35].

Heuristics of this type reduce search by skipping over intermediate relations (this is why we do not call abstraction relations 'heuristics'). These associations are usually uncertain because the intermediate relations may not hold in the specific case. Intermediate relations may be omitted because they are unobservable or poorly understood. In a medical diagnosis program, heuristics typically skip over the causal relations between symptoms and diseases. In Section 4 we will analyze the nature of these implicit relations in some detail.

To summarize, in heuristic classification abstracted data statements are associated with specific problem solutions or features that characterize a solution. This can be shown schematically in simple terms (Fig. 2.3).

This diagram summarizes how a distinguished set of terms (data, data abstractions, solution abstractions, and solutions) are related systematically by

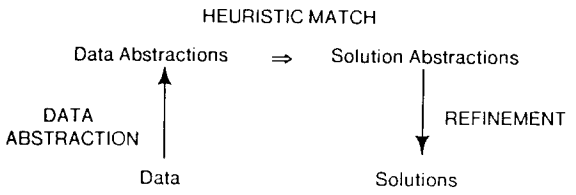


FIG. 2.3. Inference structure of heuristic classification.

different kinds of relations. This is the *structure of inference* in heuristic classification. The direction of inference and the relations 'abstraction' and 'refinement' are a simplification, indicating a common ordering (generalizing data and refining solutions), as well as a useful way of remembering the classification model. In practice, there are many operators for selecting and ordering inferences, discussed in Section 6.

3. Examples of Heuristic Classification

Here we schematically describe the architectures of SACON, GRUNDY, and SOPHIE III in terms of heuristic classification. These are brief descriptions, but reveal the value of this kind of analysis by helping us to understand what the programs do. After a statement of the problem, the general inference structure and an example inference path are given, followed by a brief discussion. In looking at these diagrams, note that sequences of classifications can be composed, perhaps involving simple classification at one stage (SACON) or omitting 'abstraction' or 'refinement' (GRUNDY and SACON).

In Section 4, we will reconsider these examples, in an attempt to understand the heuristic classification pattern. Our approach will be to pick apart the 'inner structure' of concepts and to characterize the kinds of relations that are typically useful for problem solving.

3.1. SACON

Problem: SACON [5] selects classes of behavior that should be further investigated by a structural-analysis simulation program (Fig. 3.1).

Discussion: SACON solves two problems by classification—heuristically analyzing a structure and then using simple classification to select a program. It begins by heuristically selecting a simple numeric model for analyzing a structure (such as an airplane wing). The numeric model, an equation, produces stress and deflection estimates, which the program then qualitatively abstracts as behaviors to study in more detail. These behaviors, with additional information about the material, definitionally characterize different configurations of the MARC simulation program (e.g., the inelastic-fatigue program).

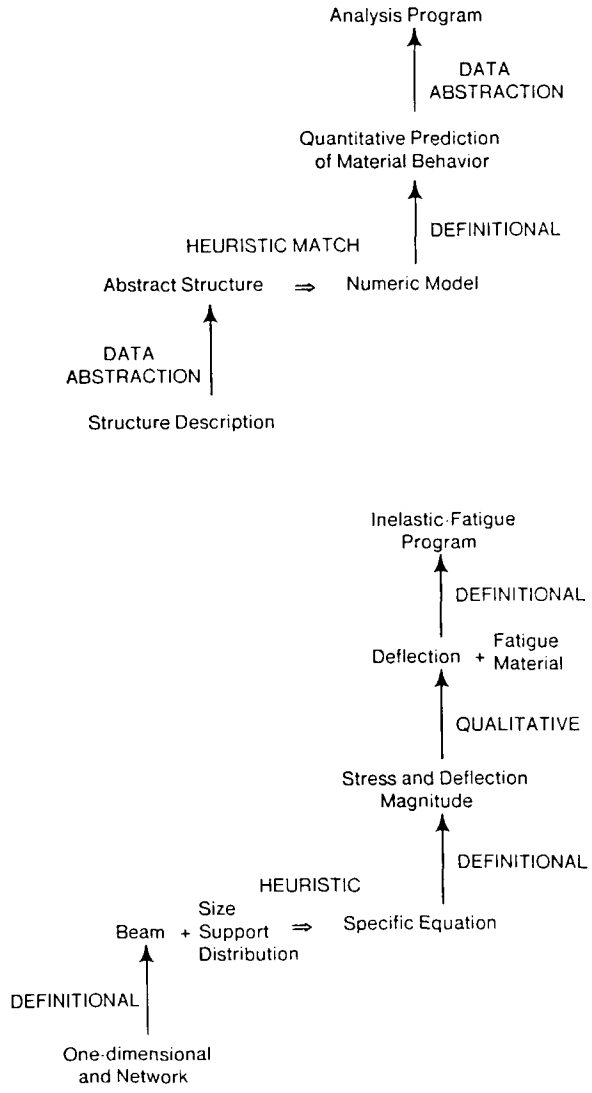


FIG. 3.1. Inference structure of SACON.

There is no refinement because the solutions to the first problem are just a simple set of possible models, and the second problem is only solved to the point of specifying program classes. (In another software configuration system we analyzed, specific program input parameters are inferred in a refinement step.)

3.2. GRUNDY

Problem: GRUNDY [78] is a model of a librarian, selecting books a person might like to read.

Discussion: GRUNDY solves two classification problems heuristically, classifying a reader's personality and then selecting books appropriate to this kind of person (Fig. 3.2). While some evidence for person stereotypes is by data abstraction (a JUDGE can be inferred to be an EDUCATED-PERSON), other evidence is heuristic (watching no TV is neither a necessary nor sufficient characteristic of an EDUCATED-PERSON).

Illustrating the power of a knowledge-level analysis, we discover that the people and book classifications are not distinct in the implementation. For example, 'fast plots' is a book characteristic, but in the implementation 'likes fast plots' is associated with a person stereotype. The relation between a person stereotype and 'fast plots' is heuristic and should be distinguished from abstractions of people and books. One objective of the program is to learn better people stereotypes (user models). The classification description of the user modeling problem shows that GRUNDY should also be learning better ways to characterize books, as well as improving its heuristics. If these are not treated separately, learning may be hindered. This example illustrates why a knowledge-level analysis should precede representation.

It is interesting to note that GRUNDY does not attempt to perfect the user model before recommending a book. Rather, refinement of the person stereo-

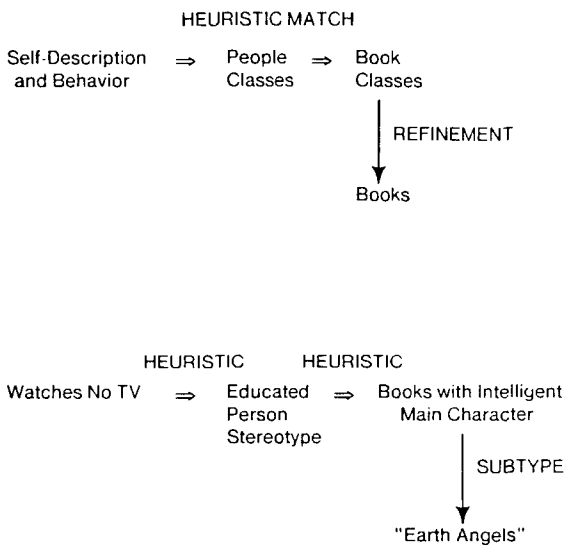


FIG. 3.2. Inference structure of GRUNDY.

type occurs when the reader rejects book suggestions. Analysis of other programs indicates that this multiple-pass process structure is common. For example, the Drilling Advisor makes two passes on the causes of drill sticking, considering general, inexpensive data first, just as medical programs commonly consider the 'history and physical' before laboratory data. The high-level, abstract structure of the heuristic classification model makes possible these kinds of descriptions and comparisons.

3.3. SOPHIE III

Problem: SOPHIE III [12] classifies an electronic circuit in terms of the component that is causing faulty behavior (Fig. 3.3).

Discussion: SOPHIE's set of pre-enumerated solutions is a lattice of valid and faulty circuit behaviors. In contrast with MYCIN, SOPHIE's solutions are device states and component flaws, not stereotypes of disorders. They are related causally, not by subtype. Data are not only external device behaviors, but include internal component measurements propagated by the causal analy-

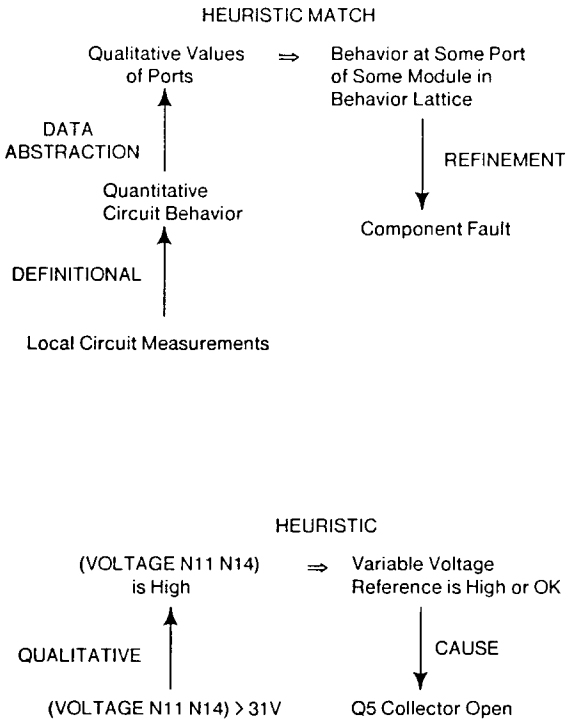


FIG. 3.3. Inference structure of SOPHIE.

sis of the LOCAL program. Nevertheless, the inference structure of abstractions, heuristic relations, and refinement fits and heuristic classification model, demonstrating its generality and usefulness.

4. Understanding Heuristic Classification

The purpose of this section is to develop a principled account of why the inference structure of heuristic classification takes the characteristic form we have discovered. Our approach is to describe what we have heretofore loosely called 'classes', 'concepts', or 'stereotypes' in a more formal way, using the conceptual graph notation of Sowa [88]. In this formalism, a concept is described by graphs of typed, usually binary relations among other concepts. This kind of analysis has its origins in semantic networks [77], the conceptual-dependency notation of Schank et al. [83], the prototype/perspective descriptions of KRL [7], the classification hierarchies of KL-ONE [85], as well as the predicate calculus.

Our discussion has several objectives:

(1) to relate the knowledge encoded in rule-based systems to structures more commonly associated with 'semantic net' and 'frame' formalisms;

(2) to explicate what kinds of knowledge heuristic rules leave out (and thus their advantages for search efficiency and limitations for correctness); and

(3) to relate the kinds of conceptual relations collectively identified in knowledge representation research (e.g., the relation between an individual and a class) with the pattern of inference that typically occurs during heuristic classification problem solving (yielding the characteristic inverted horeshoe inference structure of Fig. 2.3).

One important result of this analysis is a characterization of the 'heuristic relation' in terms of primitive relations among concepts (such as preference, accompaniment, and causal enablement), and its difference from more essential, 'definitional' characterizations of concepts. In short, we are trying to systematically characterize *the kind of knowledge that is useful for problem solving*, which relates to our larger aim of devising useful languages for encoding knowledge in expert systems.

4.1. Schemas vs. definitions

In the case of matching features of organisms (MYCIN) or programs (SACON), features are essential (necessary), identifying characteristics of the object, event, or process. This corresponds to the Aristotelian notion of concept definition in terms of necessary properties.[3] In contrast, features may be only 'incidental', corresponding to typical manifestations or behaviors. For example,

[3] Sowa [88] provides a good overview of these well-known philosophical distinctions. See also [28, 72].

E. coli is normally found in certain parts of the body, an incidental property. It is common to refer to the combination of incidental and defining associations as a 'schema' for the concept.[4] Inferences made using incidental associations of a schema are inherently uncertain. For example, we might infer that a particular person, because he is educated, likes to read books, but this might not be true. In contrast, an educated person must, by definition, have learned a great deal about something (though maybe not a formal academic topic).

The nature of schemas and their representation has been studied extensively in AI. As stated in the introduction (Section 1), our purpose here is to exploit this research to understand the knowledge contained in rules. We are not advocating one representation over another; rather we just want to find some way of writing down knowledge so that we can detect and express patterns. We use the conceptual graph notation of Sowa because it is simple and it makes basic distinctions that we find to be useful:

- A schema is made up of *coherent statements* mentioning a given concept, not a list of isolated, independent features. (A statement is a complete sentence.)
- A schema for a given concept contains *relations to other concepts*, not just 'attributes and values' or 'slots and values'.
- A concept is typically described from different points of view by *a set of schemata* (called a 'schematic cluster'), not a single 'frame'.
- The totality of what people know about a concept usually extends well beyond the schemas that are pragmatically encoded in programs for solving limited problems.

Finally, we adopt Sowa's definition of a *prototype* as a 'typical individual', a specialization of a concept schema to indicate typical values and characteristics, where ranges or sets are described for the class as a whole. Whether a program uses prototype or schema descriptions of its solutions is not important to our discussion, and many may combine them, including 'normal' values, as well as a spectrum of expectations.

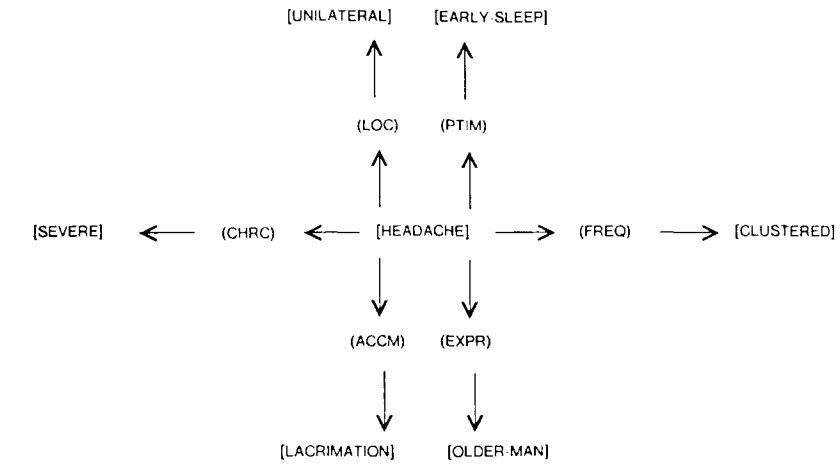
4.2. Alternative encodings of schemas

To develop the above points in some detail, we will consider a conceptual graph description and how it relates to typical rule-based encodings. Fig. 4.1 shows how knowledge about the concept 'cluster headache' is described using the conceptual graph notation.[5]

Concepts appear in brackets; relations are in parentheses. Concepts are also related by a type hierarchy, e.g., a HEADACHE is a kind of PROCESS, an

[4] Here we use the word 'schema' as a *kind of knowledge*, not a construct of a particular programming language or notation. See [49] for further discussion of this distinction.

[5] One English translation would be: "A cluster headache is a headache that occurs with a frequency in clusters, experienced by an older man, accompanied by lacrimation, with characteristic severe, of location unilateral, occurring at a point in time of early sleep."




```
[EARLY-SLEEP] is
        [TIME: [STATE: [SLEEP]] → (AFTER) → [TIME-PERIOD: @few-hrs]]

[CLUSTERED] is
        [DAILY] ← (FREQ) ← [EVENT] → (DURATION) → [TIME-PERIOD: @1week]

[OLDER-MAN] is
        [MAN] → (CHRC) → [OLD]
```

FIG. 4.1. Schema describing the concept CLUSTER-HEADACHE and some related concepts.

OLDER-MAN is a kind of MAN. Relations are constrained to link concepts of particular types, e.g., PTIM, a point in time, links a PROCESS to a TIME. For convenience, we can also use Sowa's linear notation for conceptual graphs. Thus, OLDER-MAN can be described as a specialization of MAN, "a man with characteristic old". CLUSTERED in "an event occurring daily for a week". EARLY-SLEEP is "a few hours after the state of sleep".

We make no claim that a representation of this kind is complete, computationally tractable, or even unambiguous. For our purposes here, it is simply a *notation* with the advantage over English prose of systematically revealing how what we know about a concept can be (at least partially) described in terms of its relations to concepts of other types.

For contrast, consider how this same knowledge might be encoded in a notation based upon objects, attributes, and values, as in MYCIN. Here, the object would be the PATIENT, and typical attributes would be HEADACHE-ONSET (with possible values EARLY-MORNING, EARLY-SLEEP, LATE-AFTERNOON) and DISORDER (with possible values CLUSTER-HEADACHE, INFECTION, etc.). A typical rule might be, "If the patient has headache onset during sleep, then the disorder of the patient is cluster headache." The features of a cluster headache might be combined in a single rule. Generally, since none of the features are logically necessary, they are

considered in separate rules, with certainty factors denoting how strongly the symptom (or predisposition, in the case of age) is correlated with the disease. A primitive 'frame' representation, as in INTERNIST [76], is similar, with a list of attributes for each disorder, but each attribute is an 'atomic' unit that bundles together what is broken into object, attribute, and value in MYCIN, e.g., "HEADACHE-ONSET-OCCURS-EARLY-SLEEP".

The idea of relating a concept (such as CLUSTER-HEADACHE) to a set of attributes or descriptors, is common in AI programs. However, a relational analysis reveals marked differences in what an attribute might be:

(a) *An attribute is an atomic proposition.* In INTERNIST, an attribute is a string that is only related to diseases or other strings, e.g., HEADACHE-ONSET-EARLY-SLEEP-EXPERIENCED-BY-PATIENT.

(b) *An attribute is a relation characterizing some class of objects.* In MYCIN, an attribute is associated with an instance of an object (a particular patient, culture, organism, or drug).

(b1) *An attribute is a unary relation.* A MYCIN attribute with the values 'yes' or 'no' corresponds to a unary relation, (⟨attribute⟩ ⟨object⟩), e.g., (HEADACHE-ONSET-EARLY-SLEEP PATIENT), "headache onset during early sleep is experienced by the patient."

(b2) *An attribute is a binary relation.* A MYCIN attribute with *values* corresponds to a binary relation, (⟨attribute⟩ ⟨object⟩ ⟨value⟩), e.g., (HEADACHE-ONSET PATIENT EARLY-SLEEP), "headache onset experienced by the patient is during early sleep."

(c) *An attribute is a relation among classes. Each class is a concept.* Taking the same example, there are two more primitive relations, ONSET and EXPERIENCER, yielding the propositions: (ONSET HEADACHE EARLY-SLEEP), "the onset of the headache is during early sleep", and (EXPERIENCER HEADACHE PATIENT), "the experiencer of the headache is the patient." More concisely,

$$[\text{EARLY-SLEEP}] \leftarrow (\text{ONSET}) \leftarrow [\text{HEADACHE}] \rightarrow (\text{EXPR}) \rightarrow [\text{PATIENT}] .$$

These relations and concepts can be further broken down, as shown in Fig. 4.1.

The conceptual graph notation encourages clear thinking by forcing us to unbundle domain terminology into defined or schematically described terms and a constrained vocabulary of relations (restricted in the types of concepts each can link). Rather than saying that "an object has attributes", we can be more specific about the relations among entities, describing abstract concepts like 'headache' and 'cluster' in the same notation we use to describe concrete objects like patients and organisms. In particular, notice that headache onset is a characterization of a headache, not of a person, contrary of the MYCIN statement that "headache onset is an attribute of person." Similarly, the

relation between a patient and a disorder is different from the relation between a patient and his age.[6]

Breaking apart 'parameters' into concepts and relations has the additional benefit of allowing them to be easily related, through their schema descriptions. For example, it is clear that HEADACHE-ONSET and HEADACHE-SEVERITY both characterize HEADACHE, allowing us to write a simple, general inference rule for deciding about relevancy: "If a process type being characterized (e.g., HEADACHE) is unavailable or not relevant, then its characterization (e.g., HEADACHE-ONSET) is not relevant." As another example, consider a discrimination inference strategy that compares disorder processes on the basis of their descriptions as *events*. Knowing what relations are comparable (e.g., location and frequency), the inference procedure can automatically gather relevant data, look up the schema descriptions, and make comparisons to establish the best match. To summarize, the rules in a program like MYCIN are implicitly making statements about schemas. This becomes clear when we separate conceptual links from rules of inference, as in NEOMYCIN.

4.3. Relating heuristics to conceptual graphs

Given all of the structural and functional statements we might make about a concept, describing processes and interactions in detail, some statements will be more useful than others for solving problems. Rather than thinking of schemas as inert, static descriptions, we are interested in how they link concepts to solve problems. The description of CLUSTERED-HEADACHE given in Fig. 4.1 includes the knowledge that one typically finds in a diagnostic program. To understand heuristics in these terms, consider first that some relations appear to be less 'incidental' than others. The time of occurrence of the headache, location, frequency, and characterizing features are all closely bound to what a cluster headache is. They are not necessary, but they together distinguish CLUSTER-HEADACHE from other types. That is, these relations *discriminate* this headache from other types of headache.

On the other hand, accompaniment by lacrimation (tearing of the eyes) and the tendency for such headaches to be experienced by older men are correlations with other concepts.[7] Here, in particular, we see the link between different kinds of entities: a DISORDER-PROCESS and a PERSON. This is the link we have identified as a heuristic—a direct, non-hierarchical association

[6] The importance of defining relations has been discovered repetitively in AI. Woods' analysis of semantic networks [100] is an early, well-known example. The issue of restricting and defining relations was particularly important in the development of OWL [61]. Researchers using rule-based languages, like MYCIN's, felt curiously immune from these issues, not realizing that their 'attributes' were making similar confusions.

[7] What discriminates is relative. If kinds of headache tended to be associated with different ages of people, then this might be a CLUSTER-ELDERLY-HEADACHE and we would consider the age of the experiencer to be a discriminating characteristic.

between concepts of different types. Observe that *why* an older man experiences cluster headaches is left out. Given a model of the world that says that all phenomena are caused, we can say that each of the links with HEADACHE could be explained causally. Whether the explanation has been left out or is not known cannot be determined by examining the conceptual graph, a critical point we will return to later.

When heuristics are stated as rules in a program like MYCIN, even known relational and definitional details are often omitted. This often means that intermediate concepts are omitted as well. We say "X suggests Y", or "X makes you think of Y". Unless the connection is an unexplained correlation, such a statement can be expanded to a full sentence that is part of the schema description of X and/or Y. Thus, the geologist's rule "goldfields flowers→ serpentine rock" might be restated as, "Serpentine rock has nutrients that enable goldfields to grow well." Fig. 4.2 shows the conceptual graph notation of this statement (with "enable" shown by the relation "instrument" linking an entity, nutrients, to an act, growing).

The concepts of nutrients and growing are omitted from the rule notation, just as the causal details that explain the growth process are skipped over in the conceptual graph notation. The rule indicates what you must observe (goldfields flowers growing) and what you can assert (serpentine rock is near the surface). It captures knowledge not as mere static descriptions, but an efficient, useful connections for problem solving. Moreover, the example makes clear the essential characteristic of a heuristic inference—a non-hierarchical and non-definitional connection between concepts of distinct classes.

Heuristics are selected statements that are useful for inference, particularly how one class choice *constraints* another. Consider the goldfields example. Is the conceptual graph shown in Fig. 4.2 a schema for serpentine, goldfields, nutrient, or all three? First, knowledge is something somebody knows; whether goldfields is associated with nutrients will vary from person to person. (And for at least a short time, readers of this paper will think of goldfields when the word 'nutrient' is mentioned.) Second, the real issue is how knowledge is *practically indexed.* The associations a problem solver forms and the directionality of these associations will depend on the kinds of situations he is called upon to interpret, and what is given and what is derived. Thus, it seems plausible that a geologist in the field would see goldfields (data) and think

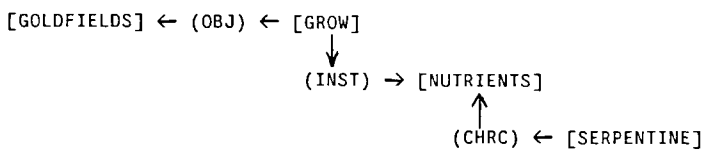


FIG. 4.2. A heuristic rule expanded as a conceptual graph.

about serpentine rock (solution). Conversely, his task might commonly be to find outcroppings of serpentine rock; he would work backwards to think of observables that he might look for (data) that would indicate the presence of serpentine. Indeed, he might have many associations with flowers and rocks, and even many general rules for how to infer rocks (e.g., based on other plants, drainage properties of the land, slope). Fig. 4.3 shows one possible inference path.

In summary, a heuristic association is a connection that relates data that is commonly available to the kinds of interpretations the problem solver is trying to derive. For a physician starting with characteristics of a person, the patient, connections to diseases will be useful. It must be possible to relate new situations to previous interpretations and this is what the abstraction process in classification is all about (recall the quotation from Bruner in Section 1). The specific person becomes an 'old man' and particular disorders come to mind.

Problems tend to start with objects in the real world, so it makes sense that practical problem-solving knowledge would allow problems to be restated in terms of stereotypical objects: kinds of people, kinds of patients, kinds of stressed structures, kinds of malfunctioning devices, etc. Based on our analysis of expert systems, links from these data concepts to solution concepts come in different flavors:

- agent or experiencer (e.g., people predisposed to diseases);
- cause, co-presence, or correlation (e.g., symptoms related to faults);
- preference or advantage (e.g., people related to books);
- physical model (e.g., abstract structures related to numeric models).

These relations do not characterize a solution in terms of 'immediate properties'—they are not definitional or type discriminating. Rather, they capture *incidental associations between a solution and available data, usually concrete concepts.* (Other kinds of links may be possible; these are the ones we have discovered so far.)

The essential characteristic of a heuristic is that it reduces search. A heuristic rule reduces a conceptual graph to a single relation between two concepts. Through this, heuristic rules reduce search in several ways:

(1) Of all possible schemas that might describe a concept, heuristic con-

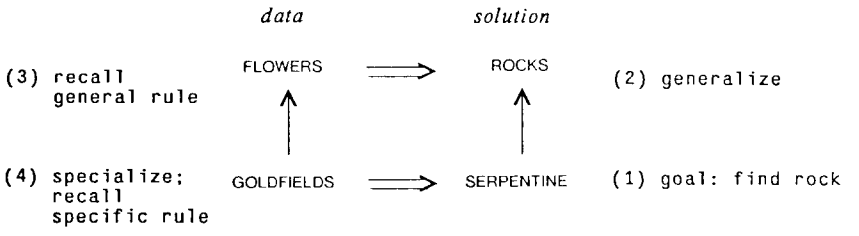


FIG. 4.3. Using a general rule to work backwards from a solution.

nections are those that constrain a categorization on the basis of available data (e.g., the strength of SERPENTINE rock may be irrelevant for inferring the presence of hidden deposits).

(2) A heuristic eliminates consideration of intermediate (and often invariant) relations between the concepts it mentions, associating salient cases directly (e.g., the goldfields rule omits the concept NUTRIENT).

While not having to think about intermediate connections is advantageous, this sets up a basic conflict for the problem solver—his inferential leaps may be wrong. Another way of saying that the problem solver skips over things is that there are *unarticulated assumptions* on which the interpretation rests. We will consider this further in the section on inference strategies (Section 6).

4.4. Relating inference structure to conceptual graphs

In the inference-structure diagrams (such as Fig. 3.2) nodes stand for propositions (e.g., "the reader is an educated person"). The diagrams relate propositions on the basis of how they can be inferred from one another: type, definition, and heuristic. So far in this section we have broken apart these atomic propositions to distinguish a heuristic link from essential and direct characterizing relations in a schema; and we have argued how direct, incidental connections between concepts, which leave out intermediate relations, are valuable for reducing search.

Here we return to the higher-level, inference-structure diagrams and include the details of the kinds of links that are possible. In Fig. 4.4 each kind of

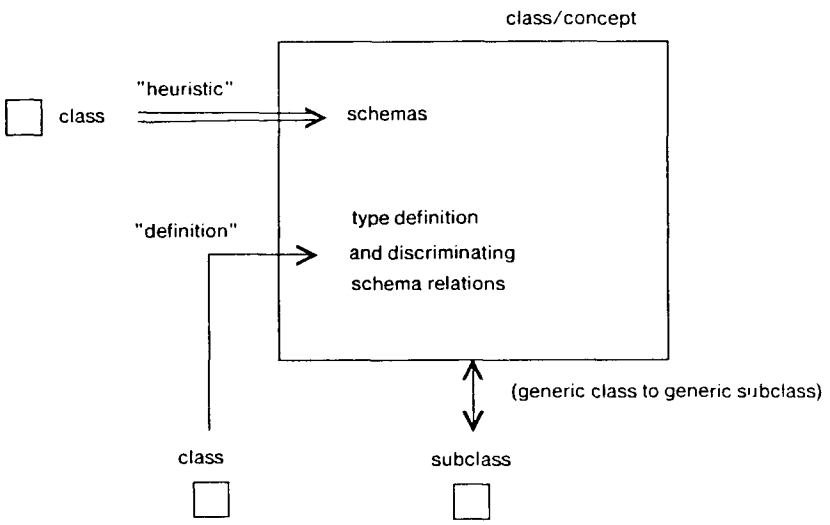


FIG. 4.4. Conceptual relations used in heuristic classification.

inference relation between concepts is shown as a line. Classes can be connected to one another by any of these three kinds of inference relations. We make a distinction between heuristics (direct, non-hierarchical, class-class relations, such as the link between goldfields and serpentine rock) and definitions (including necessary and discriminating relations, plus qualitative abstraction (see Section 2.2)). Definitional and subtype links are shown vertically, to conform to our intuitive idea of generalization of data to higher categories, what we have called *data abstraction.*

It is important to remember that the 'definitional' links are often non-essential, 'soft' descriptions. The 'definition' of leukopenia as white blood count less than normal is a good example. 'Normal' depends on everything else happening to the patient, so inferring this condition always involves making some assumptions.

Note also that this is a diagram of static, structural relations. In actual problem solving other links will be required to form a *case-specific model,* indicating propositions the problem solver believes to be true and support for them. In particular, *surrogates* [88] (also called *individuals* [10], such the MYCIN 'context' ORGANISM-1) will stand for unknown objects or processes in the world that are identified by associating them with a class in a type hierarchy.[8]

Now we are ready to put this together to understand the pattern behind the inference structure of heuristic classification. Given that a sequence of heuristic classifications, as in GRUNDY, is possible, indeed common, we start with the simplest case by assuming that data classes are not inferred heuristically. Instead, data are supplied directly or inferred by definition. When solution classes are inferred by definition, we have a case of *simple classification* (Section 2.1), for example, when an organism is actually seen growing in a laboratory culture (like a smoking gun). In order to describe an idealized form of heuristic classification, we leave out definitional inference of solutions. Finally, inference has the general form that problem descriptions must be

[8] It is not often realized that each MYCIN 'context' has a *distinguished attribute* called its 'name' that corresponds to the link between the surrogate (entity to be classified) and a classification hierarchy. The pattern was only evident to system designers to the extent that they realized that each 'context' type has some identifying attribute that allows it to be translated. For example, after identifying an organism, the program says "the E.coli" rather than ORGANISM-1 (or whatever its number was), referring to the object/context hierarchy if there are more than one, "the E.coli from the blood culture of 3/14/77." Thus, we have the identity of the organism, name of the infection, site of the culture, etc. Corresponding to each of these identifying attributes is a hierarchy of 'values' with static properties. Thus, there are tables of organisms, infections, culture sites, etc. It is in such a table that MYCIN stores the information that E.coli is a gram-negative rod. A single, general rule uses the table to identify the unknown organism. These tables are also called 'grids'; we were unaware at the time (1974–1977) that we were recording the same kind of information other AI programmers were storing in 'frame hierarchies'. The pattern was partially obscured by our use of special-case rules, for example, to allow for inforrect data, making the grids appear to be a convenient computational short-hand for collapsing similar rules, rather than a notation for describing classes.

abstracted (proceeding from subclass to class) and partial solutions must be refined (proceeding from class to subclass).

If we thus specialize the right side of the inference diagram in Fig. 4.4 to a data class and a solution class and glue them together, we get a refined version of the original inverted horseshoe (Fig. 2.3). Fig. 4.5 shows how data and solution classes are typically inferred from one another in the simplest case of heuristic classification. This diagram should be contrasted with all of the possible networks we could construct, linking concepts by the three most general relations (subtype, definitional, incidental). For example, all links might have been definitional, all concepts subsumed by a single class, or data only incidentally related to other concepts. Furthermore, considering knowledge apart from how it is used, we might imagine complex networks of concepts, intricately related, as suggested by Fig. 4.1. Instead, we find that diverse classification structures are often linked directly, omitting relational details. Clearly independent of programming language, this pattern is very likely an essential aspect of practical, experiential models of the world.

4.5. Pragmatics of defining concepts

In the course of writing and analyzing heuristic programs, we have been struck by the difficulty of defining terms. What is a 'compromised host'? How is it different from 'immunosuppression'? Is an alcoholic immunosuppressed? We do not simply write down descriptions of what we know. The very process of formalizing terms and relations changes what we know, and itself brings about concept formation.

In many respects, the apparent unprincipled nature of MYCIN is a good reflection of the raw state of how experts talk. Two problems we encountered

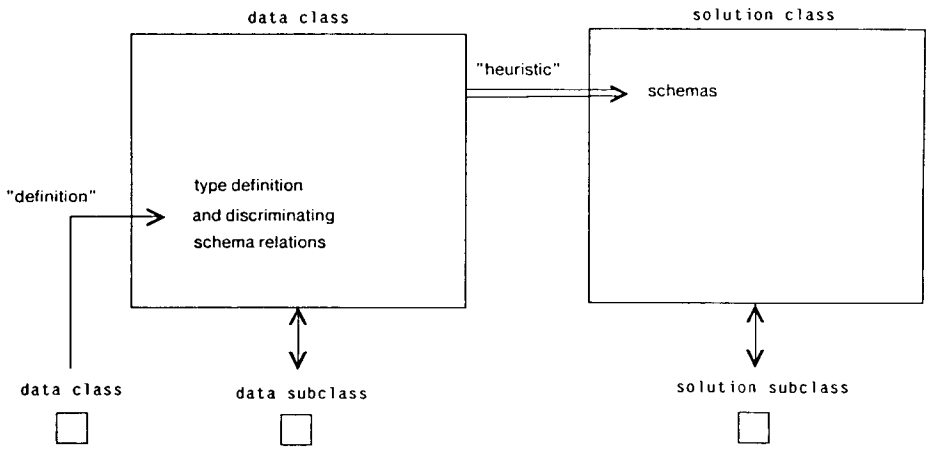


FIG. 4.5. Typical conceptual relations in simplest form of heuristic classification.

illustrate the difficulty of proceeding without a formal conceptual structure, and thus, reflect the unprincipled state of what experts know about their own reasoning:

(a) Twice we completely reworked the hierarchical relations among immunosuppression and compromised host conditions. There clearly is no agreed-upon network that we can simply write down. People do not know schema hierarchies in the same sense that they know phone numbers. A given version is believed to be better because it makes finer distinctions, so it leads to better problem solving.

(b) The concepts of 'significant organism' and 'contaminant' were sometimes confused in MYCIN. An organism is significant if there is evidence that it is associated with a disease. A contaminant is an organism growing on a culture that was introduced because of dirty instruments or was picked up from some body site where it normally grows (e.g., a blood culture may be contaminated by skin organisms). Thus, evidence against contamination supports the belief that the discovered organism is significant. However, a rule writer would tend to write 'significant' rather than 'not contaminant', even though this was the intended, intermediate interpretation. There may be a tendency to directly form a general, positive categorization, rather than to make an association to an intermediate, ruled-out category.

To a first approximation, it appears that what we 'really' know is what we can conclude given other information. That is, we start with just implication ($P \rightarrow Q$), then go back to abstract concepts into types and understand relations among them. For example, we start by knowing that "WBC < 2500 → LEUKOPENIA". To make this principled, we break it into the following pieces:

(1) 'Leukopenia' means that the count of leukocytes is impoverished:
[LEUKOPENIA] = [LEUKOCYTES] → (CHRC) →
[CURRENT-COUNT] → (CHRC) → [IMPOVERISHED].

(2) 'Impoverished' means that the current measure is much less than normal:
[IMPOVERISHED: x] =
[CURRENT-MEASURE: x] → ($\ll$) → [NORMAL-MEASURE: x].

(3) The (normal/current) count is a kind of measure:
[COUNT] < [MEASURE]

(4) A fact, the normal leukocyte count in an adult is 7000:
[LEUKOCYTES] → (CHRC) → [NORMAL-COUNT] → (MEAS) →
[MEASURE: 7000/mm3].

With the proper interpreter (and perhaps some additional definitions and relations), we could instantiate and compose these expressions to get the effect of the original rule. This is the pattern we follow in knowledge engineering, constantly decomposing terms into general types and relations to make explicit the rationale behind implications.

Perhaps one of the most preplexing difficulties we encounter is distinguishing between subtype and cause, and between state and process. Part of the problem is that cause and effect are not always distinguished by our experts. For example, a physician might speak of a brain-tumor as a kind of brain-mass-lesion. It is certainly a kind of brain-mass, but it causes a lesion (cut); it is not a kind of lesion. Thus, the concept bundles cause with effect and location: a *lesion* in the *brain* caused by a *mass* of some kind is a brain-mass-lesion (Fig. 4.6).

Similarly, we draw causal nets linking abnormal states, saying that brain-hematoma (mass of blood in the brain) is caused by brain-hemorrhage (bleeding). To understand what is happening, we profit by labeling brain-hematoma as a *substance* (a kind of brain-mass) and brain-hemorrhage as a *process* that affects or produces the substance. Yet when we began, we thought of brain-hemorrhage as if it were equivalent to the escaping blood.

It is striking that we can learn concepts and how to relate them to solve problems, without understanding the links in a principled way. If you know that WBC < 2500 is leukopenia, a form of immunosuppression, which is a form of compromised host, causing E.coli infection, you are on your way to being a clinician. As novices, we push tokens around in the same non-comprehending way as MYCIN.

Once we start asking questions, we have difficulty figuring out how concepts are related. If immunosuppression is the state of being unable to fight infection by mechanisms, then does impoverished white cells *cause* this state? Or is it *caused by* this state (something else affected the immunosystem, reducing the WBC as a side-effect)? (Worse yet, we may say it is an 'indicator', completely missing the fact that we are talking about causality.) Perhaps it is one way in which the immunosystem can be diminished, so it is *a kind of* immunosuppression. It is difficult to write down a principled network because we don't know the relations, and we don't know them because we don't know what the concepts mean—we don't understand the processes involved. Yet, we might know enough to relate data classes to therapy classes and save the patient's life!

A conceptual graph or logic analysis suggests that the relations among concepts are *relatively* few in number and fixed in meaning, compared to the number and complexity of concepts. The meaning of concepts depends on what we ascribe to the links that join them. Thus, in practice we jockey around concepts to get a well-formed network. Complicating this is our tendency to use terms that bundle cause with effect and to relate substances directly, leaving out intermediate processes. At first, novices might be like today's expert programs. A concept is just a token or label, associated with knowledge

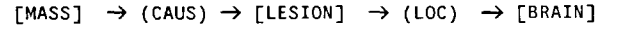


FIG. 4.6. Conceptual graph of the term 'brain-mass-lesion'.

of how to infer truth and how to use information (what to do if it is true and how to infer other truths from it). Unless the token is defined by something akin to a conceptual graph, it is difficult to say that the novice or program understands what it means. But in the world of action, what matters more than the functional, pragmatic knowledge of knowing what to do?

Where does this leave us? One conclusion is that 'principled networks' are impossible. Except for mathematics, science, economics, and similar domains, concepts do not have formal definitions. While heuristic programs sometimes reason with concrete, well-defined classifications such as the programs in SACON and the fault network in SOPHIE, they more often use experiential schemas, the knowledge we say distinguishes the expert from the novice. In the worst case, these experiential concepts are vague and incompletely understood, such as the diseases in MYCIN. In general, there are underlying (unarticulated or unexamined) assumptions in every schema description. Thus, the first conclusion is that for concepts in nonformal domains this background and context cannot in principle be made explicit [36]. That is, our conceptual knowledge is inseparable from our as yet ungeneralized memory of experiences.

An alternative point of view is that, regardless of ultimate limitations, it is obvious that expert systems will be valuable for replacing the expert on routine tasks, aiding him on difficult tasks, and generally transforming how we write down and teach knowledge. Much more can be done in terms of memory representation, learning from experience, and combining principled models with situation/action, pragmatic rules. Specifically, the problem of *knowledge transformation* could become a focus for expert systems research including compilation for efficiency, derivation of procedures for enhancing explanation [92], and re-representation for detecting and explaining patterns, thus aiding scientific theory formation. Studying and refining actual knowledge bases, as exemplified by this section, is our chief methodology for improving our representations and inference procedures. Indeed, from the perspective of knowledge transformation, it is ironic to surmise that we might one day decide that the 'superficial' representation of EMYCIN rules is a fine executable language, and something like it will become the target for our knowledge compilers.

5. Analysis of Problem Types in Terms of Systems

The heuristic classification model gives us two new angles for comparing problem-solving methods and kinds of problems. First, it suggests that we characterize programs by whether solutions are selected or constructed. This leads us to the second perspective, that different 'kinds of things' might be selected or constructed (diagnoses, user models, etc.). In this section we will adopt a single point of view, namely that a solution is most generally a set of beliefs describing what is true *about a system* or a set of actions (operations)

that will physically *transform a system* to a desired description. We will study variations of system description and transformation problems, leading to a hierarchy of *kinds of problems* that an expert might solve.

5.1. What gets selected?

This foray into systems analysis begins very simply with the observation that all classification problem solving involves *selection* of a solution. We can characterize kinds of problems by *what is being selected*:

- *diagnosis*: solutions are faulty components (SOPHIE) or processes affecting the device (MYCIN);
- *user model*: solutions are people stereotypes in terms of their goals and beliefs (first phase of GRUNDY);
- *catalog selection*: solutions are products, services, or activities, e.g., books, personal computers, careers, travel tours, wines, investments (second phase of GRUNDY);
- *model-based analysis*: solutions are numeric models (first phase of SACON);
- *skeletal planning*: solutions are plans, such as packaged sequences of programs and parameters for running them (second phase of SACON, also first phase of experimental planning in MOLGEN [38]).

Attempts to make knowledge engineering a systematic discipline often begin with a listing of kinds of problems. This kind of analysis is always prone to category errors. For example, a naive list of 'problems' might list 'design', 'constraint satisfaction', and 'model-based reasoning', combining a kind of problem, an inference method, and a kind of knowledge. For example, one might solve a VLSI chip design problem using constraint satisfaction to reason about models of circuit components. It is important to adopt a single perspective when making a list of this kind.

In particular, we must not confuse what gets selected—what constitutes a solution—with the method for computing the solution. A common misconception is that there is a kind of problem called a 'classification problem', opposing, for example, classification problems with design problems (for example, see [88]). Indeed, some tasks, such as identifying bacteria from culture information, are inherently solved by *simple classification.* However, heuristic classification as defined here is *a description of how a particular problem is solved by a particular problem solver.* If the problem solver has a priori knowledge of solutions and can relate them to the problem description by data abstraction, heuristic association, and refinement, then the problem can be solved by classification. For example, if it were practical to enumerate all of the computer configurations R1 might select, or if the solutions were restricted to a predetermined, explicit set of designs, the program could be reconfigured to solve its problem by classification. The method of solving a configuration problem is not inherent in the task itself.

With this distinction between problem and computational method in mind, we turn our attention to a systematic study of problem types. Can we form an explicit taxonomy that includes the kinds of applications we might typically encounter?

5.2. Background: Problem categories

One approach might be to focus on objects and what can be done to them. We can design them, diagnose them, use them in a plan to accomplish some function, etc. This seems like one way to consistently describe kinds of problems. Surely everything in the world involves objects.

However, in attempting to derive such a uniform framework, the concept of 'object' becomes a bit elusive. For example, the analysis of problem types in *Building Expert Systems* (hereafter BES, [51], see Table 5.1) indirectly refers to a program as an object. Isn't it really a process? Are procedures objects or processes? It's a matter of perspective. Projects and audit plans can be thought of as both objects and processes. Is a manufacturing assembly line an object or a process? The idea of a 'system' appears to work better than the more common focus on objects and processes.

By organizing descriptions of problems around the concept of a system, we can improve upon the distinctions made in BES. As an example of the difficulties, consider that a situation description is a description of a system. Sensor data are observables. But what is the difference between INTERPRETATION (inferring system behavior from observables) and DIAGNOSIS (inferring system malfunctions from observables)? Diagnosis, so defined, includes interpretation. The list appears to deliberately have this progressive design behind it, as is particularly clear from the last two entries, which are composites of earlier 'applications'. In fact, this idea of multiple 'applications' to something (student behavior, system behavior) suggests that a simplification might be found by adopting more uniform terminology. As a second example,

TABLE 5.1. Generic categories of knowledge engineering applications. From [51], Table 1.1, p. 14

INTERPRETATION	Inferring situation descriptions from sensor data
PREDICTION	Inferring likely consequences of given situation
DIAGNOSIS	Inferring system malfunctions from observables
DESIGN	Configuring objects under constraints
PLANNING	Designing actions
MONITORING	Comparing observations to plan vulnerabilities
DEBUGGING	Prescribing remedies for malfunctions
REPAIR	Executing a plan to administer a prescribed remedy
INSTRUCTION	Diagnosis, debugging, and repairing student behavior
CONTROL	Interpreting, predicting, repairing, and monitoring system behaviors.

consider that the text of BES says that automatic programming is an example of a problem involving planning. How is that different from configuration under constraints (i.e., design)? Is automatic programming a planning problem or a design problem? We also talk about experiment design and experiment planning. Are the two words interchangeable?

We can get clarity by turning things around, thinking about systems and what can be done to and with them.

5.3. A system-oriented approach

We start by informally defining a *system* to be a complex of interacting objects that have some process (I/O) behavior. The following are examples of systems:
- a stereo system,
- a VLSI chip,
- an organ system in the human body,
- a computer program,
- a molecule,
- a university,
- an experimental procedure.

Webster's defines a system to be "a set or arrangement of things so related or connected as to form a unity or organic whole." The parts taken together have some structure. It is useful to think of the unity of the system in terms of how it behaves. Behavior might be characterized simply in terms of inputs and outputs.

Figs. 5.1 and 5.2 summarize hierarchically what we can do to or with a system, revising the BES table. We group operations in terms of those that *construct* a system and those that *interpret* a system, corresponding to what is generally

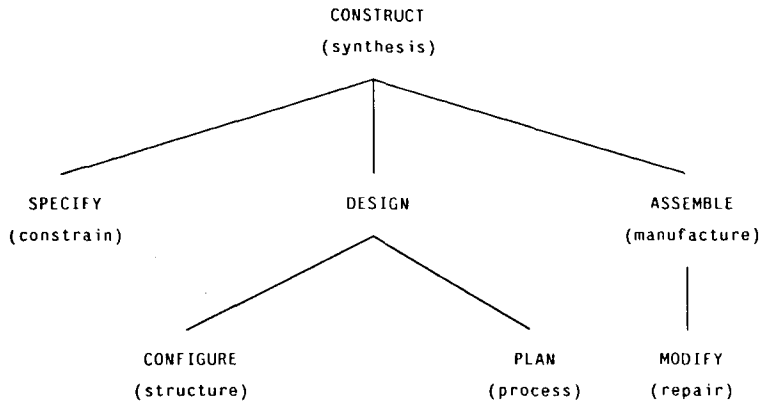


FIG. 5.1. Generic operations for synthesizing a system.

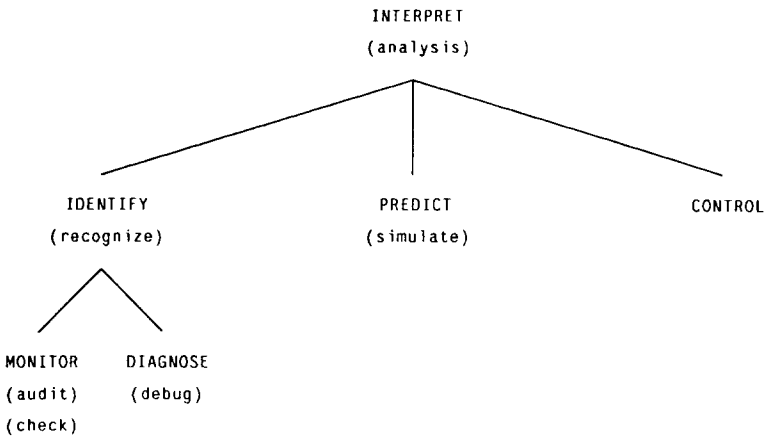


FIG. 5.2. Generic operations for analyzing a system.

called *synthesis* and *analysis*. Common synonyms appear in parentheses below the generic operations. In what follows, our new terms appear in upper case.

INTERPRET operations concern a working system in some environment. In particular, IDENTIFY is different from DESIGN in that it requires taking I/O behavior and mapping it onto a system. If the system has not been described before, then this is equivalent to (perhaps only partial) design from I/O pairs. PREDICT is the inverse, taking a known system and describing output behavior for given inputs. ('Simulate' is a specific method for making predictions, suggesting that there is a computational model of the system, complete at some level of detail.) CONTROL, not often associated with heuristic programs, takes a known system and determines inputs to generate prescribed outputs [96]. Thus, these three operations, IDENTIFY, PREDICT, and CONTROL, logically cover the possibilities of problems in which one factor of the set {input, output, system} is unknown.

Both MONITOR and DIAGNOSE presuppose a pre-existing system design against which the behavior of an actual, 'running' system is compared. Thus, one identifies the system with respect to its deviation from a standard. In the case of MONITOR, one *detects* discrepancies in behavior (or simply characterizes the current state of the system). In the case of DIAGNOSE, one *explains* monitored behavior in terms of discrepancies between the actual (inferred) design and the standard system.

To carry the analysis further, we compare our proposed terms to those used in *Building Expert Systems*:

(1) 'Interpretation' is adopted as a generic category that broadly means to describe a working system. The most rudimentary form is simply identifying some unknown system from its behavior. Note that an identification strictly

speaking involves a specification of constraints under which the system operates and a design (structure/process model). In practice, our understanding may not include a full design, let alone the constraints it must satisfy (consider the metarules of HERACLES (Section 6.1) versus our vague understanding of why they are reasonable). Examples of programs that identify systems are:

- DENDRAL: system = molecular (structure) configuration [16] (given spectrum behavior of the molecule).
- PROSPECTOR: system = geological (formation) configuration [47] (given samples and geophysics behavior).
- DEBUGGY: system = knowledge (program) configuration of student's subtraction facts and procedure [17] (given behavior on a set of subtraction problems).

(2) 'Prediction' is adopted directly. Note that prediction, specifically simulation, may be an important technique underlying *all* of the other operations (e.g., using simulation to generate and test possible diagnoses).

(3) 'Diagnosis' is adopted directly as a kind of IDENTIFICATION, with some part of the design characterized as faulty with respect to a preferred model.

(4) 'Design' is taken to be the general operation that embraces both a characterization of structure (CONFIGURATION) and process (PLANNING).

(5) 'Monitoring' is adopted directly as a kind of IDENTIFICATION, with system behavior checked against a preferred, expected model.

(6) 'Debugging' is dropped, deemed to be equivalent to DIAGNOSIS plus MODIFY.

(7) 'Repair' is more broadly termed MODIFY; it could be characterized as transforming a system to effect a redesign, usually prompted by a diagnostic description. MODIFY operations are those that change the *structure* of the system, for example, editing a program or using drugs (or surgery) to change a living organism. Thus, MODIFY is a form of 'reassembly' given a required design modification.

(8) The idea of 'executing a plan' is moved to the more general term ASSEMBLE, meaning the physical construction of a system. DESIGN is conceptual; it describes a system in terms of spatial and temporal interactions of components. ASSEMBLY is the problem of actually putting the system together in the real world. For example, contrast R1's problem with the problem of having a robot assemble the configuration that R1 designed. ASSEMBLY is equivalent to planning at a different level, that of a system that builds a designed subsystem.

(9) 'Instruction' is dropped because it is a composite operation that doesn't apply to every system. In a strict sense, it is equivalent to MODIFY.

In addition to the operations already mentioned, we add SPECIFY—referring to the separable operation of constraining a system description, generally in terms of interactions with other systems and actual realization in

the world (resources affecting components). Of course, in practice design difficulties may require modifying the specification, just as assembly may constrain design (commonly called 'design for manufacturing').

5.4. Configuration and planning

The distinction between configuration and planning requires some discussion. We will argue that they are two points of view for the single problem of designing a system. For example, consider how the problem of devising a vacation travel plan is equivalent to configuring a complex system consisting of travel, lodging, restaurant, and entertainment businesses and specifying how that system will service a particular person or group of people. 'Configure' views the task as that of organizing objects into some system that as a functioning whole will process/transform another (internal) system (its input). 'Plan', as used here, turns this around, viewing the problem in terms of how an entity is transformed by its interactions with another (surrounding) system. Fig. 5.3 illustrates these two points of view.

VLSI design is a paradigmatic example of the 'configuration' point of view. The problem is to piece together physical objects so that their behaviors interact to produce the desired system behavior.

The 'planning' point of view itself can be seen from two perspectives depending on whether a subsystem or a surrounding global system is being serviced:

(1) *We service some system by moving it around for processing by subsystems of a surrounding world.* Paradigmatic examples are experiment planning (e.g., MOLGEN [39, 89]) and shop scheduling (e.g., ISIS [37]). ASSEMBLY always involves planning of this form, and strictly speaking Stefik's MOLGEN solves an assembly problem, designing a system that physically constructs a DNA/cell configuration (a pre-designed subsystem). Equivalent examples are errand

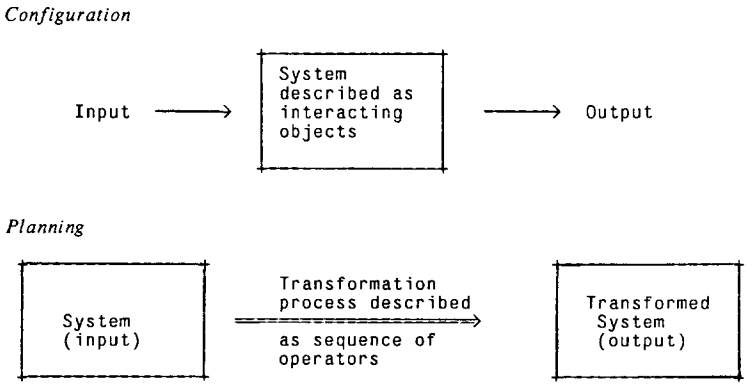


FIG. 5.3. The design problem seen from two perspectives.

planning [50], vacation, and education plans. Here there is a well-defined object that is transformed by a well-defined sequence of interactions. In general, we do not care how the surrounding system is modified by these interactions, except that there are resource constraints affecting planning when many systems are being serviced.

(2) *We specify how a well-defined object system will service a larger system in which it is contained.* Servicing is done by "moving the object system around". The paradigmatic example is the traveling salesman problem. Most realistic problems are hybrid because the 'service subsystem' is resource limited and must be 'restocked' and 'refueled' by the surrounding system. *Truckin'*, the game used for teaching LOOPS [90], makes this clear. The traditional traveling salesman problem takes this form when allowance is made for food or fuel stops, etc.

While we appear to have laid out three perspectives on design, they are all computationally equivalent. It's our point of view about *purpose* and structuredness of interactions that makes it easier to understand a system in one way rather than another. In particular, in the first form of planning, the serviced subsystem is getting more organized as a result of its interactions. The surrounding world is modified in generally entropy-increasing ways as its resources are depleted. In the second form of planning the serviced world is getting more organized, while the servicing subsystem depletes its resources. Without considering the entropy change, *there is just a single point of view of a surrounding system interacting with a contained subsystem.*

'Configuration' is concerned with the construction of well-structured systems. In particular, if subsystems correspond to physically-independent components, design is equivalent to organizing pieces so they spatially fit together, with flow from input to output ports producing the desired result. (Note that R1 is given some of the pieces, not the functional properties of the computer system it is configuring. The functional design is implicit in the roster of pieces it is asked to configure—the customer's order.) It is a property of any system that can be described in this way that it is hierarchically decomposable into modular, locally interacting subsystems—the definition of a well-structured system. As Simon [86] points out, it is sufficient for design tractability for systems to be 'nearly decomposable', with weak, but non-negligible interactions among modules.

Now, to merge this with the conception of 'planning', consider how an abstract process can be visualized graphically in terms of a diagram of connected operations. The recent widespread availability of computer graphics has revolutionized how we visualize systems (processes and computations). Examples of traditional and more recent attempts to visualize the structure of processes are:

(1) *Flowcharts.* A program is a system. It is defined in terms of a sequence of operations for transforming a subsystem, the data structures of the program.

Subprocedures and sequences of statements are subsystems that are structurally blocked and connected.

(2) *Automata theory.* Transition diagrams are one way of describing finite state machines. Petri nets and dataflow graphs are other, related, notations for describing computational processes (see [88] for discussion).

(3) *Actors.* A system can be viewed in terms of interacting, independent agents that pass messages to one another. Emphasis is placed on rigorous, local specification of behaviors [52]. Object-oriented programming [43] is in general an attempt to characterize systems in terms of a configuration, centering descriptions on objects that are pieced together, as opposed to centering on data transformations.

(4) *Thinglab.* Borning [9] emphasized the use of multiple, graphic views for depicting a dynamic simulation of mutually constrained components of a system. Borning mentions the advantages of visual experimentation for understanding complex system interactions.

(5) *Rocky's boots.* In this personal computer game[9], icons are configured to define a program, such as a sorting routine that operates on a conveyor belt. Movement icons permit automata to move around and interact with each other, thus describing 'planning' (how systems will interact) from a 'configuration' (combination of primitive structures) point of view.

(6) *FLIPP displays.* Decision rules can be displayed in analog form as connected boxes that are interpreted by top-down traversal (see Fig. 5.4). Subproblems can be visually 'chunked'; logical reasoning can be visualized in terms of adjacency, blocking, alternative routes, etc. Characteristic of analog representations, such displays are economical, facilitating encoding and perception of interactions [60].

(7) *Streams.* The structure of procedures can be made clearer by describing them in terms of the *signals* that flow from one stage in a process to another [1]. Instead of modeling procedures in terms of time-varying objects (variables, see 'planning' in Fig. 5.3), we can describe procedures in terms of time-invariant streams. For example, a program might be characterized as

ENUMERATE + FILTER + MAP + ACCUMULATE,

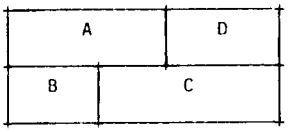


FIG. 5.4. Simple FLIPP display, encoding rules A→B, A→C, and D→C. (From [29].)

[9] The Learning Company, Menlo Park, CA.

a configuration of connected subprocesses. Stream descriptions, inspired by signal processing diagrams, allow a programmer to visualize processes in a succinct way that reveals the structural similarity of programs.

These examples suggest that we have not routinely viewed 'planning' problems in terms of system 'configuration' because we have not had adequate notations for visualizing interactions. In particular, we have lacked tools for graphically displaying hierarchical interactions and movement of subsystems through a containing system. Certainly a large part of the problem is that interactions can be *opportunistic*, so the control strategy that affects servicing (in either form of planning) is not specifiable as a fixed sequence of interactions. The inability to graphically illustrate flexible strategies was one limitation of the original Actors formalism [52]. On the other hand, control strategies themselves may be specifiable as a hierarchy of processes, even though they are complex and allow for opportunism. The representation of procedures in HERACLES (Section 6.1) as layered rule sets (corresponding to tasks) (with both data-directed reasoning encoded as a separate set of tasks and inherited 'interrupt" conditions) is an example of a well-structured encoding of an opportunistic strategy. More generally, strategy might be graphically visualized as layers of object-level operations and agenda-processing operations.

In general, a configuration point of view is impossible when physical or planning structures are unstable, with many global interactions [52]. It is difficult or impossible to plan in such a world; this suggests that most practical planning problems can be characterized in terms of configuration. It is interesting to note that replacing state descriptions (configurations) with process descriptions has played an important role in scientific understanding of the origins of systems [86]. As illustrated by the examples of this section, to understand these processes, we return to a configuration description, but now at the level of the structure of the process instead of the system it constructs or interprets.

5.5. Combinations of system problems

Given the above categorization of construction and interpretation problems, it is striking that the expert systems tend to solve a sequence of problems pertaining to a given system in the world. Two sequences that commonly occur are:

(a) *The construction cycle*: SPECIFY + DESIGN {+ ASSEMBLE}.

An example is R1 with its order processing front-end, XSEL. Broadly speaking, selecting a book for someone in GRUNDY is *single-step planning*; the person is 'serviced' by the book. Other examples are selecting a wine or a class to attend. The common sequence of terms in business, 'plan and schedule', are here named SPECIFY (objectives) and PLAN (activities).

(b) *The maintenance cycle*: {MONITOR + PREDICT + } DIAGNOSE + MODIFY.

This is the familiar pattern of medical programs, such as MYCIN. The sequence of MONITOR and PREDICT is commonly called *test* (repeatedly observing system behavior on input selected to verify output predictions). MODIFY is also called *therapy.*

This brings us back to the BES table (Table 5.1), which characterizes INSTRUCTION and CONTROL as a sequence of primitive system operations. We can characterize the expert systems we have studied as such sequences of operations:

MYCIN = MONITOR (patient state)
+ DIAGNOSE (disease category) + IDENTIFY (bacteria) + MODIFY (body system or organism)

GRUNDY = IDENTIFY (person type) + PLAN (reading plan)

SACON = IDENTIFY (structure type)
+ PREDICT (approximate numeric model)
+ IDENTIFY (classes of analysis for refined prediction)

SOPHIE = MONITOR (circuit state)
+ DIAGNOSE (faulty module/component)

When a problem solver uses heuristic classification for multiple steps, as in GRUNDY, we say that the problem-solving method is *sequential heuristic classification.* Solutions for a given classification (e.g., stereotypes of people) become data for the next classification step. Note that MYCIN does not strictly do sequential classification because it does not have a well-developed classification of patient types, though this is a reasonable model of how human physicians reason. However, it seems fair to say that MYCIN does perform a monitoring operation in that it requests specific information about a patient to characterize his state; this is clearer in NEOMYCIN and CASNET where there are many explicit, intermediate patient state descriptions. On the other hand, SOPHIE provides a better example of monitoring because it interprets global behavior, attempting to detect faulty components by comparison with a standard, correct model (discrepancies are violated assumptions).

It should be noted that how a problem is characterized in system terms may depend on our purpose, what 'problem' we are attempting to solve in doing the system analysis or in building an expert system. For example, the OCEAN program (a product of Teknowledge, Inc.) checks configurations of computer systems. From a broad perspective, it is performing a MONITOR operation of the system that includes the human designer. Thus, OCEAN's inputs are the constraints that a designer should satisfy, plus the output of his designing process. However, unlike DEBUGGY, we are not interested in understanding and correcting the designer's reasoning. Our purpose is to determine whether the computer system design meets certain specification constraints (e.g., power and

space limitations) and to make monor corrections to the design. Thus, it seems more straightforward to say that OCEAN is doing a CONFIGURATION task, and we have given it a possible solution to greatly constrain its search.

Finally, for completeness, we note that robotics research is concerned chiefly with ASSEMBLY. Robotics is also converting CONTROL of systems from a purely numeric to a symbolic processing task. PREDICT in systems analysis has also traditionally involved numeric models. However, progress in the area of qualitative reasoning (also called mental models) [6] has made this another application for heuristic programming. Speech understanding is a strange case of identifying a system interaction between two speakers, attempting to characterize its output given a partial description (at the level of sounds) and environmental input (contextual) information.

Heuristic classification is particularly well-suited for problems of interpretation involving a system that is known to the problem solver. In this case, the problem solver can select from a set of systems he knows about (IDENTIFY), known system states (MONITOR), known system faults (DIAGNOSE), or known system behaviors (PREDICT/CONTROL). The heuristic classification method relies on experiential knowledge of systems and their behaviors. In contrast, constructing a new system requires construction of new structures (new materials or new organizations of materials). Nevertheless, we intuitively believe that experienced problem solvers construct new systems by modifying known systems. This confluence of classification and constructive problem solving is another important area for research.

Another connection the reader may have noticed: We made progress in understanding what expert systems do by describing them in terms of inference-structure diagrams. This vividly demonstrates the point made about streams, that it is highly advantageous to describe systems in terms of their configuration, *structurally*, providing dimensions for comparison. Gentner points out [42], that structural descriptions lie at the heart of analogy formation. A structural map of systems reveals similar relations among components, even though the components and/or their attributes may differ. This idea has been so important in research in the humanities during this century that it has been characterized as a movement with a distinct methodology, termed *structuralism* [31]. The quotation by Bruner at the front of this paper describing the advantage of classification for a problem solver, applies equally well to the knowledge engineer.

6. Inference Strategies for Heuristic Classification

The arrows in inference-structure diagrams indicate the flow of inference, from data to conclusions. However, the actual order in which assertions are made is often not strictly left to right, from data to conclusions. The process, most generally called *search* or *inference control* has several aspects in heuristic classification:

- How does the problem solver get data? Is it supplied or must it be requested?
- If data is requested, how does the problem solver order his requests? (Called a *question-asking strategy.*)
- Does the problem solver *focus* on alternative solutions, requesting data on this basis?
- When new data is received, how is it used to make inferences?
- If there are choices to be made, alternative inference paths, how does the problem solver select which to attempt or which to believe?

In this section we first survey some well-known issues of focusing including data gathering, hypothesis testing, and data-directed inference. In this context, we introduce the HERACLES program, which is designed to solve problems by heuristic classification, and discuss its inference strategies. After this, we consider a kind of heuristic classification, termed *causal-process classification,* in order to understand the problem of choosing among inference paths. The discussion finally serves as a bridge to a consideration of non-classification or what we call *constructive* problem solving.

6.1. Focusing in heuristic classification

Focusing concerns what inferences the problem solver makes given new information or what inferences he attempts to make towards finding a solution.

The idea of a 'triggering' relation between data and solutions is pivotal in almost all descriptions of heuristic classification inference (see [3, 81, 93]). It is called a *constrictor* by Pople in recognition of how it sharply narrows the set of possible solutions [76]. We say that "a datum triggers a solution" if the problem solver immediately thinks about that solution upon finding out about the datum. However, the assertion may be conditional (leading to an immediate request for more data) and is always context-dependent (though the context is rarely specified in our restricted-domain programs) [23]. A typical trigger relation (from NEOMYCIN) is "Headache and red painful eye suggests glaucoma"—red, painful eye will trigger consideration of headache and thus glaucoma, but headache alone will not trigger this association. In PIP [75], there is a two-stage process in which possible triggers are first brought into working memory by association with solutions already under consideration. In general, *specificity*—the fact that a datum is frequently associated with just a few solutions—determines if a datum triggers a solution concept ("brings it to mind") in the course of solving a problem.

Triggers allow search to non-exhaustively combine reasoning forwards from data and backwards from solutions. Simple classification is constrained to be hierarchically top-down or directly bottom up from known data, but heuristic triggers make search opportunistic. Briefly, a given heuristic classification network of data and solution hierarchies might be interpreted in three ways:

(1) *Data-directed search*: The program works forwards from data to ab-

stractions, matching solutions until all possible (or non-redundant) inferences have been made.

(2) *Solution- or hypothesis-directed search*: The program works backwards from solutions, collecting evidence to support them, working backwards through the heuristic relations to the data abstractions and required data to solve the problem. If solutions are hierarchically organized, then categories are considered before direct features of more specific solutions.

(3) *Opportunistic search*: The program combines data and hypothesis-directed reasoning [50]. Data abstraction rules tend to be applied immediately as data become available. Heuristic rules trigger hypotheses, followed by a focused, hypothesis-directed search. New data may cause refocusing. By reasoning about solution classes, search need not be exhaustive.

Data- and hypothesis-directed search are not to be confused with the implementation terms 'forward' or 'backward chaining'. R1 provides a superb example of how different the implementation and knowledge-level descriptions can be. Its rules are *interpreted* by forward chaining, but it does a form of hypothesis-directed search, systematically setting up subproblems by a fixed procedure that focuses reasoning on spatial subcomponents of a solution [63].

The degree to which search is focused depends on the level of indexing in the implementation and how it is exploited. For example, MYCIN's 'goals' are solution classes (e.g., types of bacterial meningitis), but selection of rules for specific solution (e.g. E.coli meningitis) is unordered. Thus, MYCIN's search within each class is unfocused [22]. Generalized heuristics, of the form "data class implies solution class" (e.g., "compromised host implies gram-negative rods" or "flowers imply underlying rocks") make it possible to focus search on useful heuristics in both directions (e.g., if looking for serpentine rock, recall that flowers identify rocks; if describing area and see flowers, recall that flowers identify rocks).[10]

Opportunistic reasoning requires that at least some heuristics be indexed so that they can be applied in either direction, particularly so new data and hypothesized solutions can be related to previously considered data and solutions. The HERACLES program (Heuristic Classification Shell, the generalization of NEOMYCIN) cross-indexes data and solutions in several ways that were not available in EMYCIN. HERACLES' inference procedure consists of 75 metarules comprising 40 reasoning tasks [23]. Focusing strategies include:

– working upwards in type hierarchies before gathering evidence for subtypes;

[10] How human knowledge is indexed plays a major role in knowledge acquisition dialogues. The heuristic-classification model suggests that it may be efficient to proceed from data classes, asking the expert for associated solution classes. But it may be difficult to enumerate data classes. Instead, the expert might find it easier to work backwards from solutions (e.g., book categories) and then use a generate and test method to scan data prototypes (e.g., people stereotypes) for a match. Knowledge acquisition for heuristic classification is briefly considered in [24]. See also the discussion of ETS in Section 10.

- discriminating hypotheses on the basis of their descriptions as processes;
- making inferences that relate a new hypothesis to previously received data;
- seeking general classes of data before subclasses; and
- testing hypotheses by first seeking triggering data and necessary causes.

In HERACLES, the operators for making deductions are abstract, each represented by a set of metarules, corresponding to a procedure or alternative methods for accomplishing a task. Such a representation makes the explicit the inference control that is implicit in MYCIN's rules [22]. As an example, Fig. 6.1 illustrates how NEOMYCIN's abstract operators use backward deduction to confirm a hypothesized solution. (The program attempts to test the hypothesis TB by applying a domain rule mentioning ocular nerve dysfunction; to find this out, the program attempts to rule it out categorically, discovering that there is a CNS problem, but there are not focalsigns; consequently, the domain rule fails.)

As a second example, the following are some of the forward-deductive data-interpretation operators that HERACLES uses to relate a new datum to known solutions.

- Finding out more specific information so that a datum can be usefully related to hypotheses (e.g., given that the patient has a headache, finding out the duration and location of the headache).
- Making deductions that use the new datum to confirm 'active' solutions (those previously considered, their taxonomic ancestors, and immediate siblings), sometimes called 'focused forward-reasoning'.
- Triggering possible solutions (restricted to abnormal findings that must be explained or 'non-specific' findings not already explained by active solutions).

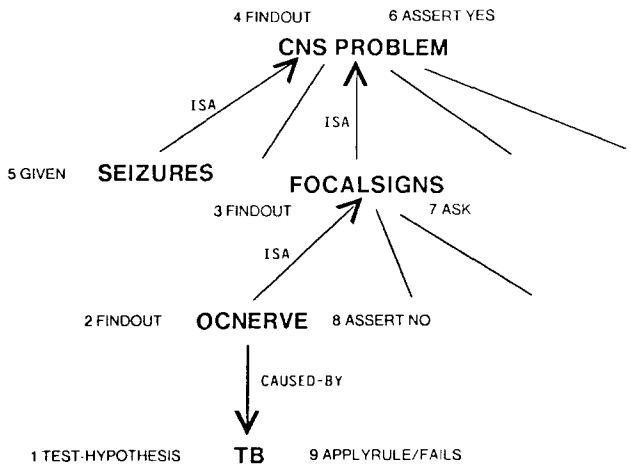


FIG. 6.1. Backward deduction in NEOMYCIN to confirm a solution.

In general, the rationale for an inference procedure might be very complex. A study of HERACLES' inference procedure reveals four broad categories of constraints:
- mathematical (e.g., efficiency advantages of hierarchical search),
- sociological (e.g., cost for acquiring data),
- cognitive (e.g., computational resources), and
- problem expectations (e.g., characteristics of typical problems).

These are discussed in some detail in [23]. Representing inference procedures so they can be explained and easily modified is currently an important research topic (e.g., see [21, 39, 68]). Making the *assumptions* behind these procedures explicit so they can be reasoned about the dynamically modified is a challenging issue that AI is just beginning to consider.

In its inference procedure representation, HERACLES brings together the advantages of rule and frame representations. The 'frame' paradigm advocates the point of view that domain concepts can be represented as hierarchies, separated from the inference procedure—an essential point on which the generality of the heuristic classification model depends. On the other hand, the 'rule' paradigm demonstrates that much of the useful problem-solving knowledge is in non-hierarchical associations, and that there are clear engineering benefits for procedures to be encoded explicitly, as well-indexed conditional actions. In HERACLES domain concepts are hierarchically related; domain rules represent heuristic, non-hierarchical associations; and metarules represent an inference procedure that interprets the domain knowledge, solving problems by heuristic classification. The architecture of HERACLES, with details about the encoding of metarules in MRS [41], the metarule compiler, and explanation program are described in [26].

6.2. Causal-process classification

A generic form of heuristic classification, commonly used for solving diagnostic problems, is *causal-process classification.* Data are generally observed malfunctions of some device, and solutions are abnormal processes causing the observed symptoms. We say that the inferred model of the device, the diagnosis, *explains* the symptoms. In general, there may be multiple causal explanations for a given set of symptoms, requiring an inference strategy that does not realize every possible association, but must reason about *alternative chains of inference.* In the worst case, even though diagnostic solutions are pre-enumerated (by definition), assertions might be taken back, so reasoning is non-monotonic. However, the most well-known programs that solve diagnostic problems by causal-process classification are monotonic, dealing with alternative lines of reasoning by assigning weights to the paths. Indeed, many programs do not even compare alternative explanations, but simply list all solutions, rank-ordered.

In this section, we will study SOPHIE in more detail (which reasons non-monotonically, using assumption-based belief maintenance), and compare it to CASNET [99] (which compares alternative chains of inference without explaining contradictions), and NEOMYCIN [27] (which reasons exhaustively, using certainty factors to rank alternative inference chains). In these programs, solutions are pre-enumerated, but paths to them must be constructed. Our study serves several purposes: (1) to use the heuristic classification model to relate electronic and medical diagnosis, revealing that medical programs are generally trying to solve a broader problem; (2) to describe alternative heuristic classification inference strategies; and (3) to distinguish between classification and constructive problem solving.

6.2.1. *Electronic and medical diagnosis compared*

In SOPHIE, valid and abnormal device states are exhaustively enumerated, can be directly confirmed, and are causally related to component failures. None of this is generally possible in medical diagnosis, nor is diagnosis in terms of component failures alone sufficient for selecting therapy. Medical programs that deal with multiple disease processes (unlike MYCIN) do reason about abnormal states (called *pathophysiologic states*, e.g., "increased pressure in the brain"), directly analogous to the abnormal states in SOPHIE. But curing an illness generally involves determining the cause of the component failure. These 'final causes' (called diseases, syndromes, etiologies) are processes that affect the normal functioning of the body (e.g., trauma, infection, toxic exposure, psychological disorder). Thus, medical diagnosis more closely resembles the task of computer system diagnosis in considering how the body relates to its environment [56]. In short, there are two problems: First to explain symptoms in terms of abnormal internal states, and second to explain this behavior in terms of external influences (as well as congenital and degenerative component flaws). This is the inference strucsture of programs like CASNET and NEOMYCIN (Fig. 6.2).

A network of pathophysiologic states causally relates data to diseases. States are linked to states, which are then linked to deseases in a classification hierarchy. Diseases may also be non-hierarchically linked by heuristics ("X is a complication of Y" [93]). The causal relations in these programs are heuristic because they assume certain physiologic structure and behavior, which is often poorly understood and not represented. In contrast with pathophysiologic states, diseases are abstractions of *processes*—causal stories with agents, locations, and sequences of events. Disease networks are organized by these process features (e.g., an organ system taxonomy organizes diseases by location). A more general term for disease is *disorder stereotype*. In *process control* problems, such as chemical manufacturing, the most general disorder stereotypes correspond to stages in a process (e.g., mixing, chemical reaction, filtering, packaging). Subtypes correspond to what can go wrong at each stage [23].

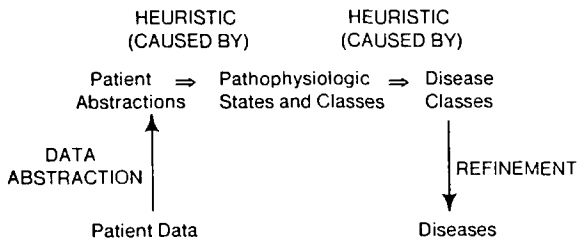


FIG. 6.2. Inference structure of causal-process classification.

Structure/function models are often touted as being more general by doing away with 'ad hoc symptom-fault rules' [40]. But programs that make a single fault assumption, such as DART, select a diagnosis from a pre-enumerated list, in this case negations of device descriptions, e.g., (NOT (XORG X1)), "X1 is not an exclusive-or gate". However, a structure/function model makes it possible to *construct tests* (see Section 7). Note that it is not generally possible to construct structure/function models for the human body, and is currently even impractical for the circuit SOPHIE diagnoses (IP-28 power supply).

To summarize, a knowledge-level analysis reveals that medical and electronic diagnosis programs are not all trying to solve the same kind of problem. Examining the nature of solutions, we see that in an electronic circuit diagnosis program like SOPHIE solutions are component flaws. Medical diagnosis programs like CASNET attempt a second step, *causal-process classification*, which is to explain abnormal states and flaws in terms of processes external to the device or developmental processes affecting its structure. It is this experiential knowledge—what can affect the device in the world—that is captured in disease stereotypes. This knowledge cannot simply be replaced by a model of device structure and function, which is concerned with a different level of analysis.

The heuristic classification model and our specific study of causal-process classification programs significantly clarifies what NEOMYCIN knows and does, and how it might be improved:

(1) diseases are processes affecting device structure and function;
(2) disease knowledge takes the form of schemas;
(3) device history schemas (classes of patients) are distinct from diseases;
(4) pathophysiologic states are malfunctioning module behaviors.

Furthermore, it is now clear that the original (bacteremia) MYCIN system does a combination of heuristic classification (using patient information to classify cultures as contaminated or significant) and simple classification (matching features of organisms, such as Gram stain and morphology). The meningitis knowledge base is more complex because it can infer the organism class heuristically, from patient abstractions, without having a culture result. NEOMY-

CIN goes a step further by dealing with *different processes* (infectious, trauma, psychogenic, etc.) and reasoning backwards from internal descriptions of current system states (e.g., brain-mass-lesion) to determine original causes (etiologies).

An important idea here is that medical diagnostic programs should separate descriptions of people from descriptions of diseases heuristically associated with them. Triggers should suggest patient types, just as they select diseases. Thus, medical diagnostic reasoning, when it takes the form of heuristic classification, is analogous to the problem-solving stages of GRUNDY, the expert librarian.

6.2.2. *Inference control for coherency*

As mentioned above, programs differ in whether they treat pathophysiologic states as independent solutions (NEOMYCIN) or find the causal path that best accounts for the data (CASNET). If problem features interact, so that one datum causes another (D1 → D2 in Fig. 6.3), then paths of inference cannot be correctly considered independently. The second feature explains the first, so classifications (alternative explanations) of the former can be omitted; there is a 'deeper cause' (C2 dominates C1). This presumes a single fault, an assumption common to programs that solve problems by heuristic classification. CASNET uses a more comprehensive approach of finding the path with the greatest number of confirmed states and no denied states. The path describes a causal process, with an etiology or ultimate cause at the head and the path of states linking the etiology to the findings serving as a causal explanation of the findings.

In the simplest rule-based systems, such as MYCIN, search is exhaustive and alternative reasoning paths are compared by numerical belief propagation (e.g., certainty factors). For example, Fig. 6.4 shows that a datum, D1, is explained by two processes, C1 and C2. MYCIN and NEOMYCIN would make all three inferences, using certainty factor combination to determine which of C1 and C2 is more likely.

A more complicated approach involves detecting that one reasoning path is subsumed by another, such as the conflict-resolution strategy of ordering production rules according to specificity. HERACLES/NEOMYCIN's treatment of

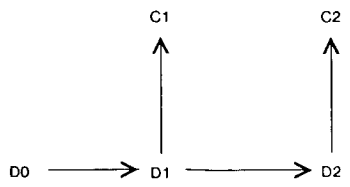


FIG. 6.3. Interacting data in classification.

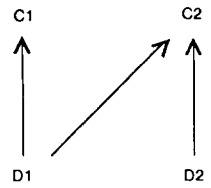


FIG. 6.4. Multiple explanations for a datum.

non-specific and 'red-flag' triggers is similar. In this case, assuming that D1 is a non-specific finding (associated with many disorders and may not need to be explained) and D2 is a red-flag finding (a serious symptom that must be explained) that triggered C2, NEOMYCIN will not make the inference relating D1 to C1 because D1 is already explained by C2. Therefore, C1 will not be added to the list of possible solutions.

After finding some single classification that accounts for the most data, a residue of unexplained findings may remain. The approach used in CASNET and INTERNIST is to remove the explained data and simply begin another cycle of classification to explain what remains. In our example, both D1 and D2 would be explained by C2, so nothing would remain.

To summarize, when there are multiple causal links for classifying data—multiple explanations—inference must be controlled to avoid redundancy, namely multiple explanations where one would have been sufficient. The aim is to produce a *coherent* model that is *complete* (accounting for the most data) and *simple* (involving one fault process, if possible). In contrast with the idea of focussing discussed earlier, coherency places constraints on the sum total of inferences that have been made, not just the order in which they are made.

Of course, for an explanation based on a pathophysiological network to be coherent, it is necessary that inferences be consistent. For example, if D1 and D2 are contradictory, the network shown in Fig. 6.4 will not produce consistent explanations. (C2 would depend on contradictory facts.) Presumably the knowledge engineer has ensured that all paths are consistent and that contradictory alternatives are explicit (e.g., by introducing (NOT D1) to the path including D2).

An ideal way to avoid these problems is to perform the diagnosis using a model of a correctly working device, in contrast with a network of pathophysiological states. This is the method used by SOPHIE. A consistent interpretation includes the observed data and assumptions about the operation of circuit components. A fault is detected by making inferences about circuit behavior in the case at hand until a contradiction is found. Specifically, SOPHIE uses assumption-based belief maintenance to detect faults. It propagates constraints (describing device behavior), records assumptions (about correct behavior of components and modules) upon which inferences are based, explains con-

tradictory inferences in terms of violated assumptions, and makes measurements to narrow down the set of possible violated assumptions to a single fault. Making assumptions explicit and reasoning about them ensures coherency, rather than relying on its implicit and ad hoc encoding in the design of a state network.

6.2.3. *Multiple solutions and levels of detail*

The first step beyond selecting single pre-enumerated faults is to dynamically construct *sets of alternative faults*, as proposed for CADUCEUS [76]. Each set of faults consistutes a *differential diagnosis* or set of alternative diagnoses. Each diagnosis consists of one or more faults. A diagnosis of multiple faults is constructed by operators that combine disorders in terms of subtype and cause. For example, referring to Fig. 6.4, one differential diagnosis would include C1 & C2; another would include C2, but not C1.

The next more complicated approach allows for interactions among disorders, as in ABEL [73]. Interaction can take the form of masking or subtracting (quantitative) effects, summation of effects, or superimposition with neither subtraction nor summation. These interactions are predicted and explained in ABEL by finding a state network, including normal states, that is *consistent on multiple levels of detail.* Combinatorial problems, as well as elegance, argue against pre-enumerating such networks, so solutions must be constructed. Each diagnostic hypothesis is a separately constructed *case-specific model*—the links describing the individual case do not all pre-exist in the knowledge base.

A simple way of comparing this kind of reasoning to what occurs in classification is to consider how concepts are instantiated and related in the two approaches. In a program like MYCIN, there are case-specific instances, but they are only the most general concepts in the knowledge base—the patient, laboratory cultures, organisms, and drugs. Links among these instances (or individuals), constituting the 'context tree', are dynamically created (albeit given as data) to form a case-specific model. In contrast, in ABEL case-specific, constructed descriptions are also at the level of individual disorders and their causes—disease components are instantiated and linked together in novel ways (thus allowing for interaction among diseases).

6.2.4. *Constructing implication paths vs. constructing solutions*

If all programs construct inference paths, aren't they all solving problems by construction of solutions? At issue is a point of view about what is a solution.

In NEOMYCIN, CASNET, and SOPHIE, the solutions are single faults, pre-enumerated. Reasoning about inference paths is a mechanism for selecting these solutions. While an inference path through the causal network of CASNET or NEOMYCIN is a disease process description, it is only a linear path, no different from a chain of implication in MYCIN. While links represent subtype and cause, they are interpreted in a uniform way for propagating weights. We conclude

that if there is only one operator for building inference paths, the program is not constructing a solution, but is only selecting the nodes at the end points of its reasoning chains. All of the programs we characterize as doing constructive problem solving either have a generator for solutions or must choose among multiple operators for constructing a solution. The solutions aren't explicitly enumerated, so there can be *no pre-existing links for mapping problem descriptions to solutions directly.* In ABEL and CADUCEUS, solutions are descriptions of disease processes, constructed by operators for incrementally elaborating and aggregating solution components, which is more than just propagating belief (what we commonly call 'implication'). The constructed solution is not simply an inference path from data to solution, but a *configuration* of primitive solution components; these programs *configure disease descriptions*, they do not select them directly.

It should now be abundantly clear that it is incorrect to say that diagnosis is a 'classification problem'. As Pople concluded in analyzing the inadequacies of INTERNIST, only *routine* medical diagnosis problems can be solved by classification [76]. When there are multiple diseases, there are too many possible combinations for the problem solver to have considered them all before. The problem solver must construct a consistent network of interacting diseases to explain the symptoms. The problem solver *formulates a solution*; he doesn't just make yes-no decisions from a set of fixed alternatives. For this reason, Pople calls non-routine medical diagnosis an ill-structured problem [87], though it may be more appropriate to reserve this term for the theory formation task of the physician-scientist who is defining new diseases. To make the point most boldly: for GRUNDY, the librarian, to satisfy a reader by constructing a solution, would have to write a new book.

7. Constructive Problem Solving, an Introduction

In a study of problem solving, Greeno and Simon characterize kinds of problems in terms of the constraints imposed on the problem solver.

> In a transformation problem such as the Tower of Hanoi, or finding a proof for a theorem, the goal is a specific [given] arrangement of the problem objects, such as a specific location for all of the disks in the Tower of Hanoi, or a specific expression to be proved in logic. Thus, the question is not what to construct, as it is in a problem of design, but how the goal can be constructed with the limited set of operators available.... [45]

While different tasks do impose different constraints on the problem solver, we have argued that experimental knowledge allows a 'design' problem to be solved as if it were a 'transformation' problem. For while design problems may not generally provide the problem solver with a specific solution to attain, he

may from experience know of a solution that will work. In heuristic classification the solution space is known to the problem solver as a *set* of explicit alternatives, and problem solving takes the form of 'proving' that one of them is best or acceptable.[11]

For example, in diagnostic programs that assume a single fault, such as NEOMYCIN, CASNET, SOPHIE, and DART, the inference process is equivalent to finding the most specific and most likely theorem (solution) that can be proved correct. Thus, the spectrum of problem-solving effort and methodology is aligned at least as much with experience as with the nature of the task. Amarel makes this point in distinguishing between 'derivation' and 'formation' problems [4], emphasizing that experience provides knowledge for mapping problem conditions to solutions. Thus, experience moves tasks for a given problem solver to the 'derivation' end of the spectrum—heuristic classification.

Often problems of DESIGN and DIAGNOSIS are not amenable to solution by heuristic classification because possible final 'states' cannot be practically enumerated, exhaustively learned (from experience or direct teaching), or for some reason a previously used solution is just not acceptable; solutions must be constructed rather than selected. However, even when solutions are constructed, classification might play a role, for example, in planning the problem-solving process or in selecting the pieces to be configured.

The essential differences between heuristic construction and classification are the need for some 'data structure' to post the assembled solution and operators for proposing and reasoning about solution fragments [34]. In classification, triggers focus the search, but may not be necessary; controlled forward-deduction from given data and hierarchical search in the set of fixed solutions may be sufficient. In construction, triggers may be essential, as well as knowledge about how parts of a solution fit together.

The following are some examples of heuristic construction operators:

(1) In HASP, the ocean surveillance signal interpretation system [70], one operator attaches a new line segment (from the input sensor) to a previous line that was last heard less than thirty minutes ago (with a certainty of 0.5), thus extending the model of the location of a particular vessel.

(2) ABEL has six 'structure building' operators, including *projection* (to hypothesize associated findings and diseases suggested by states in the case-specific model) and *causal elaboration* (to determine causal relations between states at a detailed level, based on causal relations between states at the next aggregate level) [74].

(3) AM has operators for proposing (syntactic) structural modifications to

[11] In adopting the heuristic classification model as a psychological theory, we must be more careful about this issue of explicitness. Human memory has properties different from knowledge base representations, so there is a difference between 'explicitly known now' and 'previously known'. In practice, remembering a previous solution may require reconstruction, and hence some elements of constructive problem solving.

concepts. For example, a concept is generalized by deleting a conjunction in its characteristic function definition in LISP [58]. One lesson of EURISKO is that complex concept formation requires a more extensive set of operators (defined in terms of conceptual relations, or as Lenat puts it, "slots for describing heuristics").

(4) MOLGEN [89] has both assembly-design operators and laboratory-domain operators. PROPOSE-OPERATOR is a design operator that proposes a laboratory operator that will reduce differences between the goal and current state, extending the plan forward in time. It is "responsible for linking new laboratory steps correctly to the neighboring laboratory steps and goals." There are four kinds of physical, structure-modifying laboratory operators: merge, amplify, react, and sort.

(5) In DART, diagnosis is done by classification (and the use of the proof method is made explicit in the program description), but testing the circuit to gather more information (MONITOR) is done by construction. The abstract operator (IF (AND a1 ... am Ob) THEN (OR (NOT P1) ... (NOT Pn))) serves as a template for generating test, where the ai are achievable settings or structure changes, Ob is one or more observations to be made, and the Pi are assumptions about correct device structure or function. Thus, a particular device setting and observations will confirm that one or more assumptions are violated, narrowing down the set of possible faults (similar to SOPHIE). Note that the heuristics used in DART are general search strategies used to control the deductive process, not domain-specific links. DART has heuristics and uses classification (for diagnosis), but it does not do heuristic classification, in the form we have described it. Specifically, it lacks experiential, schema knowledge for classifying device states and describing typical disorders.

(6) MYCIN's antibiotic therapy program [25] generates combinations of drugs from 'instructions' that abstractly describe how the number of drugs and their preference are related. These generator instructions can be viewed as operators (or a grammar) for constructing syntactically correct solutions.

To summarize the alternative means of computing solutions we have seen:

(1) Solutions might be *selected* from a pre-enumerated set, by classification.

(2) Solutions may be *generated* whole, as in DENDRAL and MYCIN's therapy program.

(3) Solutions may be *assembled* from primitives, incrementally, as in HASP.

As Simon indicates [86], these methods can be combined, sequentially or hierarchically (as in hierarchical planning), with perhaps alternative decompositions for a single system.

8. Relating Tools, Methods, and Tasks

In our discussion we have emphasized the question, "What is the method for computing a solution?" We have made a distinction between data and solution

in order to clarify the large-scale computational issues of constructing a solution versus selecting it from a known set. The logical next step is to relate what we have learned about conceptual structures, systems problems, and the classification/construction distinction to the tools available for building expert systems. Conventionally, the thing to do at this point would be to provide a big table relating tools and features. These can be found in many books (e.g., [46]). The new analysis we can provide here is to ask which tool features are useful for heuristic classification and which are useful for constructive problem solving.

While simple rule-based languages like EMYCIN omit knowledge-level distinctions we have found to be important, particularly schema hierarchies, they have nevertheless provided an extremely successful programming framework for solving problems by classification. Working backwards (backchaining) from a pre-enumerated set of solutions guarantees that only the relevant rules are tried and useful data considered. Moreover, the program designer is encouraged to use means-ends analysis, a clear framework for organizing rule writing. EMYCIN and other simple rule-based systems are appropriate for solving problems by heuristic classification because inference chains are short (commonly five or fewer steps between raw data and solution), and each new rule can be easily viewed as adding a link in the mapping between data (or some intermediate abstraction) and solutions.

With the advent of more complex knowledge representations, such as KL-ONE, it is unclear whether advantages for explicit representation will be outweighed by the difficulty of learning to use these languages correctly. The analysis needed for identification of classes and relations and proper adherence to representational conventions might require considerable experience, or even unusual analytical abilities (recall the analysis of concepts in Section 4.5). Recent research indicates that it might be difficult or practically impossible to design a language for conceptual structures that can be unambiguously and consistently used by knowledge engineers [11]. Just as the rule notation was 'abused' in MYCIN by ordering rule clauses to encode hierarchical and procedural knowledge, users of KL-ONE implicitly encode knowledge in structural properties of concept hierarchies, relying on the effect of the interpreter to make correct inferences. Brachman, et al. propose a model of *knowledge representation competence*, in which a program is told what is true and what it should do, and left to encode the knowledge according to its own conventions to bring about the correct reasoning performance.[12]

[12] See [59] for details. In apparent conflict with our use of inference diagrams to describe what a heuristic-classification problem solver knows, Levesque says, "There is nothing to say about the *structure* of these abstract bodies of knowledge called knowledge bases." One way of resolving this is to say that knowledge *content* has structure, but knowledge-level specification is not *about* structures in the agent (problem solver). This is supported by Newell's remark, "Relationships exist

While many of the conceptual structures and inference mechanisms required to encode a heuristic classification problem solver have now been identified, no knowledge engineering tool today combines these capabilities in a complete package. Perhaps the best system for classification we could imagine might be a combination of KL-ONE (so that conceptual relations are explicit and to provide automatic categorization of concepts [85]), HERACLES (so that the inference procedure is explicit, well-structured, and independent of domain knowledge representation), and SHRINK [54] (to provide automatic refinement of classifications through problem-solving experience). In this respect, it should be noted there is some confusion about the nature of heuristic classification in some recent commercial tools on the market. Close inspection reveals that they are capable of only simple classification, lacking structures for data abstraction, as well as a means to separate definitional features from heuristic associations between concepts [46].

Regarding constructive problem solving, the major distinction among tools appears to be the method for coping with alternative choices in configuring a solution. Tools for constructive problem solving necessarily include methods for controlling search that go beyond the focusing operations found in tools that solve problems by classification. For example, well-known search control methods used in construction include: Least-commitment [89] (avoiding decisions that constrain future choices), representing explicitly multiple 'hypothetical' worlds (branching on choices to construct alternative solutions), variable propagation or relaxation (systematic refinement of solutions), backtracking (retracting constructions), version space search [67] (bounding a solution using variables and constraints), and debugging [91] (modifying an unsatisfactory solution).

What have we learned that enables us to match problems to tools? Given a task, such as troubleshooting, we might have previously asked, "Is this tool good for diagnosis?" Now, we insert an intermediate question about *computational requirements*: Is it possible or acceptable to pre-enumerate solutions? Is it possible or acceptable to rank order solutions? Rather than matching tasks to tools directly, we interpose some questions about the method for computing solutions. The basic choice of classification versus construction is the missing link for relating implementation terminology ('rules', 'blackboard', 'units') to high-level conceptual structures and inference requirements.

In summary, we suggest the following sequence of questions for matching problems to tools:

Continued from p. 336

between goals, of course, but these are not realized in the structure of the system, but in knowledge" [69]. This is our intent in separating the abstract characterization of what a problem solver knows (heuristic classification model) from its encoding in the agent's symbol system (expert system representation).

(1) Describe the problem in terms of a sequence of operations relating to systems. If the problem concerns ASSEMBLY or construction of a perceptual system, seek a specialist in another area of AI. If the problem concerns numerical PREDICTION or CONTROL, it might be solved by traditional systems analysis techniques.

(2) Do constraints on customization or naturally occurring variety allow the solution space to be practically pre-enumerated? If so, use heuristic classification. If not, is there a hierarchical or grammatical description that can be used to generate possible solutions? Are there well-defined solution-construction operators that are constrained enough to allow an incremental (state-space) search?

(3) Are there many uncertain choices that need to be made? If a few, exhaustive generation with a simple certainty-weighing model may be sufficient; if many, some form of lookahead or assumption/justification-based inference mechanism will be necessary.

The notation of inference-structure diagrams used in this paper can also be used to form a knowledge specification that can then be mapped onto the constructs of a particular knowledge representation language. First, identify and list possible solutions, data, and intermediate (more general) categories. Examining inference chains, classify links between concepts as definitional, type categorization, and heuristic. Then, draw an inference-structure diagram to arrange relations within a type hierarchy vertically and show heuristics as horizontal lines. Finally, map this diagram into a given representation language. For example, subtype links are represented as ordered clauses in an EMYCIN rule: "If the patient has an infection, and the kind of infection is meningitis, and. . . ."

Our study suggests two additional perspectives for critiquing constructive tools. Viewing *solutions* as models of systems in the world, we require means for detecting and controlling the coherency (completeness and consistency) of inferences. In describing *computational methods* in terms of operators, we need means to construct, record, and relate inference graphs. We conclude that the method by which inference is controlled—how an inference graph representing a system model is computed—is a crucial distinction for comparing alternative knowledge engineering tools for constructive problem solving. Relating the above methods for constructing solutions (e.g., version space, least commitment, blackboard architecture) to problem tasks is beyond the scope of this paper. It is possible that the problem categories of Section 5 will be useful. Though they may prove to be an orthogonal consideration, as we discovered in distinguishing between classification and construction.

9. Knowledge-level Analysis

As a set of terms and relations for describing knowledge (e.g., data, solutions, kinds of abstraction, refinement operators, the meaning of 'heuristic'), the

heuristic classification model provides a *knowledge-level analysis* of programs [69]. As defined by Newell, a knowledge-level analysis "serves as a specification of what a reasoning system should be able to do." Like a specification of a conventional program, this description is distinct from the representational technology used to implement the reasoning system. Newell cites Schank's conceptual dependency structure as an example of a knowledge level analysis. It indicates "what knowledge is required to solve a problem... how to encode knowledge of the world in a representation." It should be noted that Newell intends for the knowledge-level specification to include the closure of what the reasoning system *might* know. Our approach to this problem is to characterize the problem solver's computational method and the structure of his knowledge. What a heuristic classification problem solver "is able to do" is specified in terms of the patterns of problem situations and solutions he knows and the space of (coherent) mappings from raw data to solutions.

After a decade of 'explicitly' representing knowledge in AI languages, it is ironic that the pattern of heuristic classification should have been so difficult to see. In retrospect, certain views were emphasized at the expense of others:

(a) *Procedureless languages.* In an attempt to distinguish heuristic programming from traditional programming, procedural constructs are left out of representation languages (such as EMYCIN, OPS, KRL [57]). Thus, inference relations cannot be stated separately from how they are to be used [48, 49].

(b) *Heuristic nature of problem solving.* Heuristic association has been emphasized at the expense of the relations used in data abstraction and refinement. In fact, some expert systems do only simple classification; they have no heuristics or 'rules of thumb', the key idea that is supposed to distinguish this class of computer programs.

(c) *Implementation terminology.* In emphasizing new implementation technology, terms such as 'modular' and 'goal-directed' were more important to highlight than the content of the programs. In fact, "goal directed" characterizes any rational system and says very little about how knowledge is used to solve a problem. 'Modularity' is a representational issue of indexing, how the knowledge objects can be independently accessed.

Nilsson has proposed that logic should be the *lingua franca* for knowledge-level analysis [71]. Our experience suggests that the value of using logic is in adopting a set of terms and relations for describing knowledge (e.g., kinds of abstraction). Logic is especially valuable as a tool for knowledge-level analysis because it emphasizes *relations*, not just implication.

10. Related Analyses in Psychology and Artificial Intelligence

Only a monograph-length review could do justice to the vast amount of research that relates to heuristic classification. Every discipline from ancient philosophy through modern psychology seems to have considered some part of the story.

Several AI researchers have described heuristic classification in part, influencing this analysis. For example, in CRYSALIS [33] data and solution abstraction are clearly separated. The EXPERT rule language [97] similarly distinguishes between 'findings' and a taxonomy of hypotheses. In PROSPECTOR [47], rules are expressed in terms of relations in a semantic network. In CENTAUR [3], a variant of EMYCIN, solutions are explicitly *prototypes* of diseases. Chandrasekaran and his associates have been strong proponents of the classification model: "The normal problem-solving activity of the physician... (is) a process of classifying the case as an element of a disease taxonomy" [19, 44]. Recently, Chandrasekaran, Weiss, and Kulikowski have generalized the classification schemes used by their programs (MDX [18] and EXPERT [98]) to characterize problems solved by other expert systems.

In general, rule-based research in AI emphasizes the importance of heuristic association; frame systems emphasize the centrality of concepts, schema, and hierarchical inference. A series of knowledge representation languages beginning with KRL have identified structured abstraction and matching as a central part of problem solving [7]. These ideas are well-developed in KL-ONE, whose structures are explicitly designed for classification [85].

Building on the idea that 'frames' are not just a computational construct, but a theory about a kind of knowledge [49], cognitive science studies have described problem solving in terms of classification. For example, routine physics problem solving is described by Chi [20] as a process of data abstraction and heuristic mapping onto solution schemas ("experts cite the abstracted features as the relevant cues (of physics principles)"). The inference structure of SACON, heuristically relating structural abstractions to numeric models, is the same. In NEWTON, De Kleer referred to packages of equations, associated with problem features, as RALCMs (Restricted Access Local Consequent Methods) ("with this representation, only a few decisions are required to determine which equations are relevant") [32].

Related to the physics problem solving analysis is a very large body of research on the nature of schemas and their role in understanding [82, 83]. More generally, the study of classification, particularly of objects, also called *categorization*, has been a basic topic in psychology for several decades (e.g., see the chapter on "conceptual thinking" in [53] and [80]). However, in psychology and emphasis has been on the nature of categories and how they are formed (an issue of learning). The programs we have considered make an identification or selection from a pre-existing classification (an issue of memory retrieval). In recent work, Kolodner combines the retrieval and learning process in an expert system that learns from experience [54]. Her program uses the MOPS representation, a classification model of memory that interleaves generalizations with specific facts [55].

Probably the most significant work on classification was done by Bruner and his colleagues in the 1950s [14]. Bruner was concerned with the nature of

concepts (or categories), how they were attained (learned), and how they were useful for problem solving. A few quotes illustrate the clarity and relevance of his work:

> To categorize is to render discriminably different things equivalent, to group objects and events and people around us into classes, and to respond to them in terms of their class membership rather than their uniqueness. [14, p. 1]
>
> ... the task of isolating and using a concept is deeply imbedded in the fabric of cognitive life; that indeed it represents one of the most basic forms of inferential activity in all cognitive life. [14, p. 79]
>
> ... [what] we have called "concept attainment" in contrast to "concept formation" is the search for the testing of attributes that can be used to distinguish exemplars from nonexemplars of various categories, the search for good and valid anticipatory cues. [14, p. 233]

Bruner described some of the heuristic aspects of classification:

> ... regard a concept as a network of sign-significate inferences by which one goes beyond a set of observed criteria properties exhibited by an object or event in question, and then to additional inferences about other *unobserved* properties of the object or event. [14, p. 244]
>
> What does categorizing accomplish for the organism? ... it makes possible the sorting of functionally significant groupings in the world. [14, p. 245]
>
> We map and give meaning to our world by relating classes of events rather than by relating individual events. The moment an object is placed in a category, we have opened up a whole vista for "going beyond" the category by virtue of the superordinate and causal relationships linking this category to others. [14, p. 13]

Bruner was well ahead of AI in realizing the centrality of categorization in problem solving. Particularly striking is his emphasis on strategies for selecting cues and examples, by which the problem solver directs his learning of new categories ('information gathering strategies'). Bruner's study of hypothesis formation and strategies for avoiding errors in learning is particularly well-developed, "For concern about error, we contend, is a necessary condition for evoking problem-solving behavior" (page 210) (compare to 'failure-driven memory' [84] and 'impasses' [95]).

On the other hand, Bruner's description of a concept is impoverished from today's point of view. The use of toy problems (colored cards or blocks, of

course) suggested that categorization was based on 'direct significants'—a logical combination of observable, discriminating (perhaps probabilistic) features. This Aristotelian view persisted in psychology and psychometrics well into the 1970s, until the work of Rosch, who argues that concepts are prototypical, not sets of definitional features [28, 45, 64, 79]. Rosch's work was influenced by AI research, but it also had its own effect, particularly in the design of KRL [7].

The heuristic classification model presented in this paper builds on the idea that categorization is not based on purely essential features, but rather is primarily based on *heuristic*, non-hierarchical, *but direct* associations between concepts. Bruner, influenced by game theory, characterizes problem solving (once a categorization was achieved) in terms of a payoff matrix; *cues are categorized to make single decisions of actions to take.* Influenced by the psychology of the day, he views problem solving in the framework of a stimulus and a response, where the stimulus is the 'significant' (cues) and the response is the action that the categorization dictates. He gives examples of medical diagnosis in this form.

We have had the advantage of having a number of working models of reasoning to study—expert systems—whose complexity go well-beyond what Bruner was able to formally describe. We have observed that problem solving typically involves a sequence of categorizations. Each categorization can be characterized generically as an operation upon a system—specify, design, monitor, etc. Most importantly, we have seen that each classification is not a final consequence or objective in "a payoff matrix governing a situation" (page 239). Rather it is often a plateau chaining to another categorization. Bruner's payoff matrix encodes heuristic associations.

Building upon Rosch's analysis and developments in knowledge representation, recent research in cognitive science has significantly clarified the nature of concepts [28, 79]. In particular, attention has turned to why concepts take the form they do. While many concepts are based on *natural kinds* (e.g., MYCIN's organisms and GRUNDY's books), others are *experiential* (e.g., reader and patient stereotypes of people), or *analytic* (e.g., SOPHIE's module behavior lattice and SACON's programs). Miller [66] suggests that formation of a category is partly constrained by its heuristic implication. Thus, therapeutic implication in medicine might serve to define diagnostic and person categories, working backwards from pragmatic actions to observables. This functional, even behavioral, view of knowledge is somewhat disturbing to those schooled in the definition of concepts in terms of essential features, but it is consistent with our analysis of expert systems. Future studies of what people know, and the nature of meaning, will no doubt depart even more from essential features to consider heuristic 'incidental' associations in more detail.

Finally, learning of classifications has been a topic in AI for some time. Indeed, interest goes back to early work in pattern recognition. As Chandrase-

karan points out [18], it is interesting to conceive of Samuels' hierarchical evaluation functions for checker playing as an implicit conceptual hierarchy. Recent work in classification, perhaps best typified by Michalski's research [65], continues to focus on learning essential or definitional features of a concept hierarchy, rather than heuristic associations between concepts. However, this form of learning, emphasizing the role of object attributes in classification, is an advance over earlier approaches that used numeric measures of similarity and lacked a conceptual interpretation. Also, working from the traditional research of psychometrics, Boose's ETS knowledge acquisition program [8] makes good use of a psychological theory of concept associations, called *personal construct theory.* However, ETS elicits only simple classifications from the expert, does not exploit distinctions between hierarchical, definition, and heuristic relations, and has no provision for data abstraction.

Perhaps the greatest value of the heuristic classification model is that it provides an overarching point of view for relating pattern recognition, machine learning, psychometrics, and knowledge representation research.

11. Summary of Key Observations

The heuristic classification model may seem obvious or trivial after it is presented, but the actual confusion about knowledge engineering tools, problem-solving methods, and kinds of problems has been quite real in AI for the past decade. Some might say, "What else could it be? It had to be classification"—as if a magic trick has been revealed. But the point of this paper is not to show a new kind of trick, or a new way of doing magic tricks, but to demystify traditional practice.

Sowa's reference to Levi-Strauss' anthropological 'systems analysis' is apt:

> The sets of features . . . seem almost obvious once they are presented, but finding the right features and categories may take months or years of analysis. The proper set of categories provides a structural framework that helps to organize the detailed facts and general principles of a system. Without them, nothing seems to fit, and the resulting system is far too complex and unwieldy.

Expert systems are in fact *systems.* To understand them better, we have given high-level descriptions of how solutions are computed. We have also related the tasks of these programs to the kinds of things one can do to or with a concrete system in the world. Below is a summary of the main arguments:

(1) A broad view of how a solution is *computed* suggests that there are two basic problem-solving methods used by expert systems: heuristic classification and construction.

(2) Kinds of inference in different stages of routine problem solving vary

systematically, so data are often generalized or redefined, while solutions are more often matched schematically and refined. A domain-specific heuristic is a direct, non-hierarchical association between different classes. It is not a categorization of a 'stimulus' or 'cue' that directly matches a concept's definition. Rather, there may be a chain of abstraction inferences before reaching categories that usefully characterize problem features. This pattern is shown two ways in Figs. 2.3 and 4.5. The inference structure of heuristic classification is common in expert systems, independent of the implementation in rules, frames, ordinary code, or some combination.

(3) Selecting solutions involves a form of proof that is often characterized as derivation or planning; constructing solutions involves piecing together solutions in a manner characterized as configuration. To *select* a solution, the problem solver needs experiential ('expert') knowledge in the form of *patterns of problems and solutions* and heuristics relating them. To *construct* a solution, the problem solver applies models of structure and behavior, in the form of constraints and inference operators, by which objects can be designed, assembled, diagnosed, employed in some plan, etc.

(4) A broad view of kinds of problems, described in terms of synthesis and analysis of *systems*, suggests two points of view for describing a system's design: a configuration in terms of structural relations of functional components, versus a plan for the processes that characterize the system's behavior. From the point of view of a *system*, reasoning may involve a limited set of generic operations, e.g., MONITOR, DIAGNOSE, MODIFY. In heuristic classification this takes the form of a sequence of mapping between classifications corresponding to each generic operation.

(5) In a manner analogous to *stream* descriptions of computer programs, the inference-structure diagrams used in this paper reveal the patterns of reasoning in expert systems.

12. Implications

A wide variety of problems can be solved by heuristic mapping of data abstractions onto a fixed, hierarchical network of solutions. This problem-solving model is supported by psychological studies of human memory and categorization. There are significant implications for expert systems research. The model provides:

(1) *A high-level structure for decomposing problems*, making it easier to recognize and represent similar problems. For example, problems can be characterized in terms of sequences of system classifications. Catalog selection (single-step planning) programs might be improved by incorporating a more distinct phase of user modelling, in which needs or requirements are explicitly classified. Diagnosis programs might profitably make a stronger separation between device-history stereotypes and disorder knowledge. 'Blackboard' sys-

tems might re-represent 'knowledge sources' to distinguish between classification and construction inference operators.

(2) *A specification for a generic knowledge engineering tool* designed specifically for heuristic classification. The advantages for knowledge acquisition carry over into explanation and teaching.

(3) *A basis for choosing application problems.* For example, problems can be selected using the systems taxonomy (Figs. 5.1 and 5.2), allowing knowledge engineers to systematically gain experience in different kinds of problems. Problems might be chosen specifically because they can be solved by heuristic classification.

(4) *A foundation for characterizing epistemologic adequacy of representation languages* [62], so that the leverage they provide can be better understood. For example, for classification it is advantageous for a language to provide constructs for representing problem solutions as a network of schemas.

(5) *A focus for cognitive studies* of human categorization of knowledge and search strategies for retrieval and matching, suggesting principles that might be used in expert programs. Human learning research might similarly focus on the inference structure of heuristic classification.

Finally, it is important to remember that expert systems are programs. Basic computational ideas such as input, output, and sequence, are essential for describing what they do. The methodology of our study has been to ask, "What does the program conclude about? How does it get there from its input?" We characterize the flow of inference, identifying data abstractions, heuristics, implicit models and assumptions, and solution categories along the way. If heuristic programming is to be different from traditional programming, a knowledge-level analysis should always be pursued at least one level deeper than our representations, even if practical constraints prevent making explicit in the implemented program everything that we know. In this way, knowledge *engineering* can be based on sound principles that unite it with studies of cognition and representation.

ACKNOWLEDGMENT

These ideas were originally stimulated by discussions with Denny Brown in our attempt to develop a framework for teaching knowledge engineering. I am grateful to Teknowledge for providing an environment that allowed me to study and analyze a variety of expert systems.

I would also like to thank Tom Dietterich, Steve Hardy, and Peter Szolovits for their suggestions and early encouragement. Discussions with Jim Bennett, John Boose, Keith Butler, Lee Erman, Rick Hayes-Roth, Ramesh Patil, Paul Rosenbloom, Kurt VanLehn, and Beverly Woolf have also been helpful.

This research has been supported in part by ONR and ARI Contract N00014-79C-0302 and the Josiah Macy, Jr. Foundation. Computational resources have been provided by the SUMEX-AIM facility (NIH grant RR00785).

REFERENCES

1. Abelson, H., Sussman, G.J. and Sussman, J., *Structure and Interpretation of Computer Programs* (MIT, Cambridge, MA, 1985).
2. Aiello, N., A comparative study of control strategies for expert systems: AGE implementation of three variations of PUFF, in: *Proceedings Third National Conference on Artificial Intelligence*, Washington, DC (August 1983) 1–4.
3. Aikins, J.S., Prototypical knowledge for expert systems, *Artificial Intelligence* **20** (1983) 163–210.
4. Amarel, S., Basic themes and problems in current AI research, in: *Proceedings Fourth Annual AIM Workshop* (June 1978) 28–46.
5. Bennett, J., Creary, L., Englemore, R. and Melosh, R., SACON: A knowledge-based consultant for structural analysis, STAN-CS-78-699 and HPP Memo 78–23, Stanford University, CA, September 1978.
6. Bobrow, D.G., Qualitative reasoning about physical systems: An introduction, *Artificial Intelligence* **24** (1984) 1–5.
7. Bobrow, D.G. and Winograd, T., KRL: Another perspective, *Cognitive Sci.* **3** (1979) 29–42.
8. Boose, J., Personal construct theory and the transfer of human expertise, in: *Proceedings Fourth National Conference on Artificial Intelligence*, Austin, TX (August 1984) 27–33.
9. Borning, A., Thinglab: A constraint-oriented simulation laboratory, STAN-CS 79-746, Stanford University, CA, July 1979.
10. Brachman, R.J., What's in a concept: Structural foundations for semantic networks, *International J. Man-Machine Studies* **9** (1977) 127–152.
11. Brachman, R.J., Fikes, R.E. and Levesque, H.J., KRYPTON: A functional approach to knowledge representation, *IEEE Comput.* **16**(10) (1983) 67–73.
12. Brown, J.S., Burton, R.R. and De Kleer, J., Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III, in: D. Sleeman and J.S. Brown (Eds.), *Intelligent Tutoring Systems* (Academic Press, London, 1982) 227–282.
13. Bruner, J.S., *The Process of Education* (Harvard University Press, Cambridge, MA, 1960).
14. Bruner, J.S., Goodnow, J.J. and Austin, G.A., *A Study of Thinking* (Wiley, New York, 1956).
15. Buchanan, B.G. and Shortliffe, E.H., *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project* (Addison-Wesley, Reading, MA, 1984).
16. Buchanan, B.G., Sutherland, G. and Feigenbaum, E.A., Heuristic dendral: A program for generating explanatory hypotheses in organic chemistry, in: B. Meltzer and D. Michie (Eds.), *Machine Intelligence* **4** (Edinburgh University Press, Edinburgh, 1969) 209–254.
17. Burton, R.R., Diagnosing bugs in a simple procedural skill, in: D. Sleeman and J.S. Brown (Eds.) *Intelligent Tutoring Systems* (Academic Press, New York, 1982) 157–183.
18. Chandrasekaran, B., Expert systems: Matching techniques to tasks, in: W. Reitman (Ed.), *AI Applications for Business* (Ablex, Norwood, NJ, 1984) 116–132.
19. Chandrasekaran, B. and Mittal, S., Conceptual representation of medical knowledge, in: M. Yovits (Ed.), *Advances in Computers* (Academic Press, New York, 1983) 217–293.
20. Chi, M.T.H., Feltovich, P.J. and Glaser, R., Categorization and representation of physics problems by experts and novices, *Cognitive Sci.* **5** (1981) 121–152.
21. Clancey, W.J., The advantages of abstract control knowledge in expert system design, in: *Proceedings Third National Conference on Artificial Intelligence*, Washington, DC (August, 1983) 74–78.
22. Clancey, W.J., The epistemology of a rule-based expert system: A framework for explanation, *Artificial Intelligence* **20** (1983) 215–251.
23. Clancey, W.J., Acquiring, representing, and evaluating a competence model of diagnosis, HPP Memo 84–2, Stanford University, February 1984; also in; M.T.H. Chi, R. Glaser, and M. Farr (Eds.), *Contributions to the Nature of Expertise*, in press.

24. Clancey, W.J., Knowledge acquisition for classification expert systems, in: *Proceedings ACM Annual Conference* (October 1984) 11–14.
25. Clancey, W.J., Details of the revised therapy algorithm, in: B.G. Buchanan and E.H. Shortliffe (Eds.), *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project* (Addison-Wesley, Reading, MA, 1984) 133–146.
26. Clancey, W.J., Representing control knowledge as abstract tasks and metarules, in: M.J. Coombs and L. Bloc (Eds.) *Computer Expert Systems* (Springer, Berlin, 1985).
27. Clancey, W.J. and Letsinger, R., NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching, in: W.J. Clancey and E.H. Shortliffe (Eds.), *Readings in Medical Artificial Intelligence: The First Decade* (Addison-Wesley, Reading, MA, 1984) 361–381.
28. Cohen, B. and Murphy, G.L., Models of concepts, *Cognitive Sci.* **8** (1984) 27–58.
29. Cox, D.J., FLIPP: A method for acquiring and displaying domain knowledge for expert systems, Center for Creativity, Inc., Cincinnati, OH, 1984.
30. Davis R., Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases, HPP Memo 76-7 and AI Memo 283, Stanford University, CA, July 1976.
31. De George, R.T. and De George, F.M. (Eds.), *The Structuralists: From Marx to Levi-Strauss* (Doubleday, Anchor Books, Garden City, 1972).
32. De Kleer, J., Qualitative and quantitative reasoning in classical mechanics, in: P.H. Winston and R.H. Brown (Eds.), *Artificial Intelligence: An MIT Perspective* (MIT, Cambridge, MA, 1979) 9–30.
33. Englemore, R. and Terry, A., Structure and function of the CRYSALIS system, in: *Proceedings Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan (August 1979) 250–256.
34. Erman, L.D., London, P.E., and Fickas, S.F., The design and example use of Hearsay-III, in: *Proceedings Seventh International Joint Conference on Artificial Intelligence*, Vancouver, BC (August, 1981) 409–415.
35. Feigenbaum, E.A., The art of artificial intelligence: I. Themes and case studies of knowledge engineering, in: *Proceedings Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA (August 1977) 1014–1029.
36. Flores, C.F. and Winograd, T., *Understanding Computers and Cognition: A New Foundation for Design* (Ablex, Norwood, NJ, 1985).
37. Fox, M.S. and Smith, S.F., ISIS—A knowledge-based system for factory scheduling, *Expert Systems* **1**(1) (1984) 25–49.
38. Friedland, P.E., Knowledge-based experiment design in molecular genetics, Tech. Rept. STAN-CS-79-771, Stanford University, CA, October 1979.
39. Genesereth, M.R., An overview of meta-level architecture, in: *Proceedings Third National Conference on Artificial Intelligence*, Washington, DC (August 1983) 119–124.
40. Genesereth, M.R., The use of design descriptions in automated diagnosis, *Artificial Intelligence* **24** (1984) 411–436.
41. Genesereth, M.R., Greiner, R. and Smith, D.E., MRS Manual, Heuristic Programming Project Memo HPP-80-24, Stanford University, CA, December 1981.
42. Gentner, D. and Stevens, A. (Eds.). *Mental models* (Erlbaum, Hillsdale, NJ, 1983).
43. Goldberg, A. and Robson, D., *Smalltalk-80: The Language and its Implementation* (Addison-Wesley, Menlo Park, CA, 1983).
44. Gomez, F. and Chandrasekaran, B., Knowledge organization and distribution for medical diagnosis, in: W.J. Clancey and E.H. Shortliffe (Eds.), *Readings in Medical Artificial Intelligence: The First Decade* (Addison-Wesley, Reading, CA, 1984) 320–338.
45. Greeno, J.G. and Simon, H.A., Problem solving and reasoning, UPITT/LRDC/ONR/APS 14, University of Pittsburgh, February 1984; also in: A. Stevens (Ed.), *Handbook of Experimental Psychology* (Wiley, New York, Rev. ed., 1985).

46. Harmon, P. and King, D., *Expert Systems: Artificial Intelligence in Business* (Wiley, New York, 1985).
47. Hart, P.E., Observations on the development of expert knowledge-based systems, in: *Proceedings Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA (August 1977) 1001–1003.
48. Hayes, P.J., In defense of logic, in: *Proceedings Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA (August 1977) 559–565.
49. Hayes, P.J., The logic of frames, in: D. Metzing (Ed.), *Frame Conceptions and Text Understanding* (de Gruyter, Berlin, 1979) 45–61.
50. Hayes-Roth, B. and Hayes-Roth, F., A cognitive model of planning, *Cognitive Sci.* **3** (1979) 275–310.
51. Hayes-Roth, F., Waterman, D. and Lenat, D. (Eds.), *Building Expert Systems* (Addison-Wesley, Reading, MA, 1983).
52. Hewitt, C., Control structure as patterns of passing messages, in: P.H. Winston and R.H. Brown (Eds.), *Artificial Intelligence: An MIT Perspective* **2** (MIT, Cambridge, MA, 1979) 433–465.
53. Johnson-Laird, P.N. and Wason, P.C., *Thinking: Readings in Cognitive Science* (Cambridge University Press, Cambridge, 1977).
54. Kolodner, J.L., The role of experience in development of expertise, in: *Proceedings Second National Conference on Artificial Intelligence*, Pittsburgh, PA (August 1982) 273–277.
55. Kolodner, J.L., Maintaining organization in a dynamic long-term memory, *Cognitive Sci.* **7** (1983) 243–280.
56. Lane, W.G., Input/output processing, in: H.S. Stone (Ed.), *Introduction to Computer Architecture* (Science Research Associates, Chicago, IL, 2nd ed., 1980) Chapter 6.
57. Lehnert, W. and Wilks, Y., A critical perspective on KRL, *Cognitive Sci.* **3** (1979) 1–28.
58. Lenat, D.B. and Brown, J.S., Why AM and EURISKO appear to work, *Artificial Intelligence* **23** (1984) 269–294.
59. Levesque, H.J., Foundations to a functional approach to knowledge representation, *Artificial Intelligence* **23** (1984) 155–212.
60. Mackinlay, J. and Genesereth, M.R., Expressiveness of languages, in: *Proceedings Fourth National Conference on Artificial Intelligence*, Austin, TX (August 1984) 226–232.
61. Martin, W.A., Descriptions and the specialization of concepts, in: P.H. Winston and R.H. Brown (Eds.), *Artificial Intelligence: An MIT Perspective* **2** (MIT, Cambridge, MA, 1979) 375–419.
62. McCarthy, J. and Hayes, P., Some philosophical problems from the standpoint of artificial intelligence, in: B. Meltzer and D. Michie (Eds.), *Machine Intelligence* **4** (Edinburgh University Press, Edinburgh, 1969) 463–502.
63. McDermott, J., R1: A rule-based configurer of computer systems, *Artificial Intelligence* **19** (1982) 39–88.
64. Mervis, C.B. and Rosch, E., Categorization of natural objects, *Annual Review of Psychology* **32** (1981) 89–115.
65. Michalski, R.S. and Stepp, R.E., Learning from observation: conceptual clustering, in: R.S. Michalski, R.E. Stepp, and T.M. Mitchell (Eds.), *Machine Learning* (Tioga, Palo Alto, CA 1983) 331–363.
66. Miller, G.A., Practical and lexical knowledge, in: E. Rosch and B.B. Lloyd (Eds.), *Cognition and Categorization* (Erlbaum, Hillsdale, NJ, 1978) 305–319.
67. Mitchell, T.M., Toward combining empirical and analytical methods for inferring heuristics, LCSR-TR 27, Laboratory for Computer Science Research, March 1982.
68. Neches, R., Swartout, W.R., and Moore, J., Explainable (and maintainable) expert systems, in: *Proceedings Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA (August 1985) 382–389.
69. Newell, A., The knowledge level, *Artificial Intelligence* **18** (1982) 87–127.

70. Nii, H.P., Feigenbaum, E.A., Anton, J.J., and Rockmore, A.J., Signal-to-symbol transformation: HASP/SIAP case study, *The AI Magazine* **3**(2) (1982) 23–35.
71. Nilsson, N.J., The interplay between theoretical and experimental methods in artificial intelligence, *Cognition and Brain Theory* **4** (1981) 69–74.
72. Palmer, S.E., Fundamental aspects of cognitive representation, in: E. Rosch and B.B. Lloyd (Eds.), *Cognition and Categorization* (Erlbaum, Hillsdale, NJ, 1978) 259–303.
73. Patil, R.S., Szolovits, P. and Schwartz, W.B., Causal understanding of patient illness in medical diagnosis, in: *Proceedings Seventh International Joint Conference on Artificial Intelligence*, Vancouver, BC (August 1981) 893–899.
74. Patil, R.S., Causal representation of patient illness for electrolyte and acid-base diagnosis, MIT/LCR/TR 267, MIT, Cambridge, MA, October 1981.
75. Pauker, S.G., Gorry, G.A., Kassirer, J.P. and Schwartz, W.B., Toward the simulation of clinical cognition: taking a present illness by computer, *Amer. J. Math.* **60** (1976) 981–995.
76. Pople, H., Heuristic methods for imposing structure on ill-structured problems: the structuring of medical diagnostics, in: P. Szolovits (Ed.), *Artificial Intelligence in Medicine* (Westview Press, Boulder, CO, 1982) 119–190.
77. Quillian, M.R., Semantic memory, in: M. Minsky (Ed.), *Semantic Information Processing* (MIT, Cambridge, MA, 1968).
78. Rich, E., User modeling via stereotypes, *Cognitive Sci.* **3** (1979) 355–366.
79. Rosch, E., Principles of categorization, in: E. Rosch and B.B. Lloyd (Eds.), *Cognition and Categorization* (Erlbaum, Hillsdale, NJ, 1978) 27–48.
80. Rosch, E. and Lloyd, B.B. (Eds.), *Cognition and Categorization* (Erlbaum, Hillsdale, NJ, 1978).
81. Rubin, A.D., Hypothesis formation and evaluation in medical diagnosis, Tech. Rept. AI-TR-316, Artificial Intelligence Laboratory, MIT, Cambridge, MA, January 1975.
82. Rumelhart, D.E. and Norman, D.A., Representation in memory, Tech. Rept. CHIP-116, Center for Human Information Processing, University of California, June 1983.
83. Schank, R.C. and Abelson, R.P., *Scripts, Plans, Goals, and Understanding* (Erlbaum, Hillsdale, NJ 1975).
84. Schank, R.C., Failure-driven memory, *Cognition and Brain Theory* **4** (1981) 41–60.
85. Schmolze, J.G. and Lipkis, T.A., Classification in the KL-ONE knowledge representation system, in: *Proceedings Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany (August 1983) 330–332.
86. Simon, H.A., *The Sciences of the Artificial* (MIT, Cambridge, MA, 1969).
87. Simon, H.A., The structure of ill structured problems, *Artificial Intelligence* **4** (1973) 181–201.
88. Sowa, J.F., *Conceptual Structures* (Addison-Wesley, Reading, MA, 1984).
89. Stefik, M., Planning with constraints, STAN-CS-80-784 and HPP Memo 80-2, Stanford University, CA, January 1980.
90. Stefik, M., Bobrow, D.G., Mittal, S. and Conway, L., Knowledge programming in loops: Report on an experimental course, *The AI Magazine* **4**(3) (1983) 3–13.
91. Sussman, G.J., *A Computer Model of Skill Acquisition* (American Elsevier, New York, 1975).
92. Swartout, W.R., Explaining and justifying in expert consulting programs, in: *Proceedings Seventh International Joint Conference on Artificial Intelligence*, Vancouver, BC (August 1981) 815–823.
93. Szolovits, P. and Pauker, S.G., Categorical and probabilistic reasoning in medical diagnosis, *Artificial Intelligence* **11** (1978) 115–144.
94. Van Melle, W., A domain-independent production rule system for consultation programs, in: *Proceedings Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan (August 1979) 923–925.
95. VanLehn, K., Human procedural skill acquisition: Theory, model, and psychological validation, in: *Proceedings Third National Conference on Artificial Intelligence*, Washington, DC (August 1983) 420–423.

96. Vemuri, V., *Modeling of Complex Systems: An Introduction* (Academic Press, New York, 1978).
97. Weiss, S.M. and Kulikowski, C.A., EXPERT: A system for developing consultation models, in: *Proceedings Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan (August 1979) 942–947.
98. Weiss, S.M. and Kulikowski, C.A., *A Practical Guide to Designing Expert Systems* (Rowman and Allanheld, Totowa, NJ, 1984).
99. Weiss, S.M., Kulikowski, C.A., Amarel, S. and Safir, A., A model-based method for computer-aided medical decision making, *Artificial Intelligence* **11** (1978) 145–172.
100. Woods, W.A., What's in a link: Foundations for semantic networks, in: D.G. Bobrow and A. Collins (Eds.), *Representation and Understanding* (Academic Press, New York, 1975) 35–82.

Received March 1985; revised version received June 1985