Artificial Intelligence 53 (1992) 1–115
Elsevier

Model construction operators

William J. Clancey
Institute for Research on Learning, 2550 Hanover Street, Palo Alto, CA 94304, USA

Received February 1990
Revised October 1990

Abstract

Clancey, W.J., Model construction operators, Artificial Intelligence 53 (1992) 1–115.

Expert systems can be viewed as programs that construct a model of some system in the world so that it can be assembled, repaired, controlled, etc. In contrast with most conventional computer programs, these models represent processes and structures by relational networks. Control knowledge for constructing such a model can be described as operators that construct a graph linking processes and structures causally, temporally, spatially, by subtype, etc. From this perspective, we find that the terminology of blackboard expert systems is not specific to a particular set of programs, but is rather a valuable perspective for understanding what every expert system is doing.

This paper reviews different ways of describing expert system reasoning, emphasizing the use of simple logic, set, and graph notations for making dimensional analyses of modeling languages and inference methods. The practical question is, how can we systematically develop knowledge acquisition tools that capture general knowledge about types of domains and modeling methods? Examples of modeling operators from ABEL, CADUCEUS, NEOMYCIN, HASP, and ACCORD demonstrate how diverse expert system approaches can be explained and integrated by the model construction perspective. Reworked examples from TEIRESIAS, XPLAIN, and KNACK illustrate how to write metarules without using domain-specific terms, thus making explicit their model construction nature. Generalizing from these observations, we combine the system-model and operator viewpoints to describe the representation of processes in AI programs in terms of three nested levels of domain, inference, and communication modeling. This synthesis reveals how the use of relational networks in computer programs has evolved from programmer descriptions of computational processes (such as flowcharts and dataflow diagrams) to network representations that are constructed and manipulated by the programs themselves.

1. Introduction

How can we systematically develop knowledge acquisition tools that capture general knowledge about types of domains and problem solving methods? Generalizing from existing programs, we seek dimensions for describing types of expert systems [15, 24, 63]. One useful approach is to view expert systems as programs that construct a model of some system in the world so that it can be assembled, repaired, controlled, etc. In contrast with most conventional computer programs, these models represent processes and structures by relational

0004-3702/92/$05.00 © 1992 — Elsevier Science Publishers B.V. All rights reserved

networks. Control knowledge for constructing relational models can be described as operators that construct a graph linking processes and structures causally, temporally, spatially, by subtype, etc. Adopting this perspective, this paper synthesizes different ways of describing expert system reasoning, emphasizing the use of simple logic, set, and graph notations for making dimensional analyses of modeling languages and inference methods.

This study reveals that the familiar distinction between "shallow" and "deep" reasoning is too simplistic [42, 50]. We observe that all expert systems are "model-based", and proceed to distinguish between classification models and simulation models of processes. For the purpose of designing knowledge acquisition tools, *expert systems are fruitfully described in terms of how relational networks are used for modeling processes*. We ask: What system is being modeled, for what purpose? Do network nodes represent internal states, structures, functions, or processes? Does the program have predefined descriptions of entire system models? Does it reason about process interactions on multiple levels of detail? Can it assemble structural and process components into new system models? Strikingly, this model construction perspective reveals that the idea of a "blackboard" is not specific to a particular set of programs, but is rather a valuable perspective for understanding what every expert system is doing.

Generalizing from these observations, we combine the system-model and operator viewpoints to describe the representation of processes in AI programs in terms of three nested levels of domain, inference, and communication modeling. This synthesis reveals how the use of relational networks in computer programs has evolved from programmer descriptions of computational processes (such as flowcharts and dataflow diagrams) to network representations that are constructed and manipulated by the programs themselves. I conclude that *qualitative process modeling* is a good way of characterizing AI programming for scientists and engineers; providing a useful pedagogical answer to the question, "What constitutes an AI program?".

1.1. Origins in knowledge representation research

An important discovery in the design of expert systems is that representing control knowledge separately from *a model of the domain* facilitates maintenance of the knowledge base and provides a basis for a shell that can be reused for similar problems [5, 20, 71]. Consequentially, a broad collection of "generic" expert system shells have been developed. They are called *task-specific architectures* because the control knowledge upon which they are based is specialized for different tasks such as diagnosis or design and often different domains such as electronics or medicine [16, 24, 47, 62]. The idea that a procedure for controlling reasoning, also called *strategic knowledge*, should be represented and made explicit as a body of knowledge in its right has thus emerged as a basic topic in the building of expert systems.

The original motivation behind this research is that a domain model should, if possible, be accessible and interpretable for multiple purposes: in different situations within a given expert system (e.g., "metarules and rule schemas" [34]), in explanation (e.g., "domain principles" [87]), in tutoring (e.g., "strategies and structural relations" [21]). More generally, "what is true" (e.g., facts about the world, facts about the representation) is represented separately from "what to do" (e.g., inference, communication, and learning procedures). This research has progressed by moving from the idea of simply controlling rules (as in Davis' conception of metarules) to reasoning about models of the domain (as emphasized by Swartout's notion of strategic explanation). In NEOMYCIN research, this involved separating out control knowledge implicit in MYCIN's domain rules [21].

The technique for building expert systems has thus advanced from tools that provide an "empty knowledge base" (e.g., EMYCIN [91]) to tools that presuppose a particular task. These tools provide a way of organizing knowledge (a language for representing a domain model) and an inference procedure for applying this knowledge. As we collect these tools and attempt to integrate them, we need to understand the relation between specific expert systems and general ways in which knowledge can be organized and applied. From the perspective of knowledge acquisition tools, can we relate "problem types" and program designs in a big-switch knowledge acquisition program that could help a knowledge engineer choose and apply the appropriate task-specific architecture? In essence, how can we formalize the part of the knowledge base that gets reused so its capabilities and limits can be related to new problems?

1.2. Generalizing NEOMYCIN

In HERACLES-DX, a diagnostic shell developed from NEOMYCIN, the reusable knowledge consists of an inference procedure represented by metarules, organized into subprocedures called *subtasks*, and a language of relations by which domain knowledge is expressed (see appendices).[1] In effect this paper is a study of NEOMYCIN's metarules, analogous to the epistemological study of MYCIN's rules that led us to develop NEOMYCIN [21, 24]. Our approach is to look for patterns in metarules so we can describe what they are doing in more general terms.

Our first study of NEOMYCIN's metarules led to the heuristic classification description of inference structure (the relations among assertions). The analysis here focuses on the inference process itself, addressing the following questions:

- *What are guidelines for writing metarules for a new task (e.g., for design)?*

[1] To avoid confusion, I will consistently refer to *tasks* as the purpose for constructing a model (e.g., diagnosis, design). *Subtasks* are HERACLES-DX subprocedures that construct a model (e.g., TEST-HYPOTHESIS).

Can we relate NEOMYCIN's metarules for diagnosis to the configuration control knowledge in ACCORD [47]?

- *Can we relate the proliferating set of application shells?* What is the space of domains and modeling methods covered by existing programs? Is a dimensional analysis possible? How can we organize the *modeling operators* in programs such as NEOMYCIN [32], MDX [38], ABEL [74], CADUCEUS [75], and KNACK [53]?
- *Can we relate the diverse terminology used for representing control knowledge: metarules, blackboards, objects/methods?* For example, why doesn't NEOMYCIN use a blackboard? Is ABEL's patient-specific model a blackboard? How are blackboard architectures fundamentally different from other expert system shells? What do they enable and for what kinds of problems are they appropriate?
- *Can general knowledge in different programs be systematically combined in one shell?* How is a diagnostic procedure for electromechanical problems different from a diagnostic procedure for medicine? More generally, why is the control knowledge for diagnosis different from that used for design? Are there common subproblems?
- *How is an inference procedure for constructing a process model different from NEOMYCIN's heuristic classification method for building a case?* Heuristic classification is characterized in terms of "taking a model off the shelf", but even this involves contrasting alternatives and building a case by weighing contrary evidence. Given that any particular line of reasoning follows a characteristic abstraction/heuristic-association/specialization path, how are these alternative paths constructed, supported, and compared?

At the heart of this analysis is an evolving perspective of what constitutes a solution in reasoning or, put another way, how we can fruitfully describe and compare what inference procedures are doing? The paper begins with a chronological exposition: the description of search in MYCIN in terms of AND/OR trees [12]; search of subtype and causal networks in CASNET [55] and CADUCEUS; and construction of a situation-specific model in ABEL, ACCORD, HASP [73], and NEOMYCIN. The view that inference consists of constructing and comparing situation-specific models of processes is then applied to describe inference procedures in these programs in terms of *operators* for manipulating *graphs* that represent *processes*. These ideas—operators, graphs, and processes—interrelate and form the basis for a synthesis that firmly anchors knowledge engineering contributions to both computer science and traditional scientific modeling.

1.3. The system-model task perspective

As in the epistemology [21] and heuristic classification [24] papers, the result

here is not a new theory of intelligence or new architecture. Rather I abstract existing programs so the methods can be more easily taught and better tools can be designed. One result is an integrated description of object-oriented programming, rule-based programming, logic programming, and blackboard architectures. Although it may appear counterintuitive at first and even obscuring, a reformulation of past work indicates that we would be right to say that *every expert system has a blackboard* and, from a related perspective, *every expert system is "model-based"*. This is true in the same sense that we can say that MYCIN does top-down refinement of a disease classification, even though such a characterization would have been foreign to the original designers (indeed, they might have said that this was an appropriate characterization of the "opposition" approach used by researchers developing medical expert systems based on frame representations).

The redescription of NEOMYCIN's metarules in terms of graph manipulation operators follows the familiar pattern in science of relating empirical phenomena (the metarules and control knowledge from diverse expert systems) to a formal language (here, simple constructs from logic, set and graph theory) [4]. Unfortunately, a formalization phobia has made the very mention of mathematical techniques anathema to many researchers. The concrete examples provided here demonstrate that "doing logic on networks" [84] does not entail something lifeless or useless. Indeed, by such an analysis—which abstracts a wide variety of techniques to show a common approach—we find that expert systems researchers have perhaps unwittingly made major contributions to computer science in the development and use of languages for modeling processes.

Equally important, as we redescribe what expert systems are doing, we make a major shift from a one-dimensional programmer's view of rules and frames to descriptions on multiple levels, which are appropriate ways of describing any expert system:

- a *system* in the world being modeled for some purpose (a *task*),
- general and situation-specific *models of processes* in this system (corresponding to the knowledge base and a problem solution),
- *relational networks* that represent these processes (hierarchies and transitional graphs),
- *computational methods* for constructing these models (inference procedures), and
- an implementation of the relational networks and inference procedure in some *programming language* (e.g., frame and rule-based languages).

This general framework is called the *system-model perspective* and forms the basis, along with the set-graph-operator description of inference procedures, for answering the questions posed above. The title of this paper reflects the idea that *control knowledge can be fruitfully described in terms of operators for*

constructing models, and this is something that all expert systems do. As will become evident, we can use this perspective to define expert systems as a class of computer programs that use relational networks for representing processes so the system being modeled can be assembled, repaired, controlled, etc. Compilers and operating systems are interesting special cases—an important realization if we are to efficiently integrate the relational network approach of AI with computer science as a whole.

1.4. AI programming as qualitative process modeling

Successfully synthesizing the work on task-specific architectures and all its attendant jargon requires changing common ways of talking about AI programming in general. In particular, three fundamental changes are called for:

- view *qualitative reasoning* as including both classification and simulation models of processes (that is, all knowledge bases contain models and all expert systems do qualitative reasoning),
- view AI programming as a methodology for *representing processes by relational networks* (specifically, rule, frame, and blackboard languages are methods for encoding and manipulating relational networks),
- view a *domain expert as an informant* about some system in the world (therefore, knowledge acquisition is primarily concerned with modeling some system in the expert's world, in contrast with modeling his mental processes).

These shifts in perspective add up to a view of AI programming as a contribution to scientific and engineering modeling in general. In expert systems, processes are represented by spatial, temporal, causal, and subtype relations among objects and events, in contrast with traditional numeric models, which are based on relations among measures. The shift is from linear, quantitative measures of *substance* (mass, energy, length) to nonlinear descriptions of *form* (orders, grammars, space/time relations). The distinction between numeric programming and relational modeling thus reflects the scientific trend of this century and, more specifically, the origin of heuristic programming in cybernetics and operations research [4, 82, 92]. In effect, we are claiming that qualitative modeling (the system-model approach) provides dimensions for systems analysis and design that are lacking in traditional numeric modeling. Furthermore, by appeal to mathematical concepts and notation, qualitative modeling can be made as rigorous as numeric modeling (a point emphasized by Bateson [4]) and need not connote a "soft" science.

Perhaps the most difficult part of our analysis is to realize that there are always different perspectives for describing a representation and never one uniquely correct interpretation. Because representations can be used to implicitly encode relations by position and ordering, such as elements in a list or

clauses in a rule, we must often go beyond surface terminology to find the relational networks embedded in a set of expert system rules. Therefore, it requires some work to extract the process-graph-operator methods in most of today's programs.

We study existing programs because we believe there is a high payoff in synthesizing the common methods they are based on. An important claim of this paper is that *describing knowledge acquisition tools in terms of model construction operators facilitates collecting and sharing knowledge bases and representational languages*. These generalized methods can then be conveyed to communities of scientists and engineers for their use, something already happening today in a limited way in fields such as civil engineering and molecular genetics.

Identifying the historical roots of qualitative modeling and its contribution to the whole of science and engineering provides a substantial basis for teaching these methods and making them available to the application communities. This is no idle task, given that the very aim of expert systems research is to provide useful tools *for every community* from physicians to geneticists and civil engineers. And it is indeed in this connection that the system-modeling description is so intuitive and useful.

1.5. Organization of the paper

This paper begins by recapitulating the development of NEOMYCIN's subtask and metarule language (presented in Appendix A.1), emphasizing the nature and importance of *abstract* control knowledge for multiple use of a knowledge base. Domain-specific metarules, such as those in TEIRESIAS, are shown to be inadequate for representing model construction operators because reasoning strategies are implicit within the metarules and within the domain rules they control (Section 2). The idea of a situation-specific model is presented as the unifying concept for understanding the inference process (Section 3). Examples from ABEL, CADUCEUS, HASP, and ACCORD illustrate the clarifying nature of the model construction perspective for describing task-specific architectures; a blackboard is described as a graph that represents structures and processes in some system being modeled; MYCIN's context tree is shown to be a blackboard (Section 4). The idea of abstract control knowledge is then described as a special form of object-oriented programming, which reveals the flexibility offered by control blackboards in BB1 over NEOMYCIN's subtasks and metarules (Section 5).

Combining the system-model and blackboard perspectives, we then view the representation of processes in AI programs in terms of three nested levels of domain, inference, and communication modeling (Section 5). A formal framework using simple logic, set, and graph notations provides a dimensional analysis for describing NEOMYCIN's subtasks and relating model construction

operators in different programs (Section 6). Heuristic classification is reconsidered in this light, revealing the relation between classification and simulation models and the macrostructures found in knowledge bases (in the form of recurrent variants of hierarchies and transition networks) (Section 7). These ideas are then applied to examples from XPLAIN and KNACK, illustrating how to write metarules without using domain-specific terms, and thus how to describe knowledge acquisition tools in terms of families of tasks and domain processes (Section 8). Adopting a historical perspective, we consider how the use of relational networks in computer programs has evolved from programmer descriptions of computational processes (such as flowcharts and dataflow diagrams) to representations or *models of processes* that are constructed and manipulated by the programs themselves (Section 9). Finally, the argument is summarized and more general conclusions are drawn (Section 10).

2. Abstract control knowledge

To create knowledge acquisition tools, we generalize from the design of particular expert systems. We seek leverage by reusing the language for representing a model of the domain, as well as by reusing the inference procedure that interprets the domain model. This is difficult if the domain model and inference procedure are conflated, as they are in MYCIN. We have developed two guidelines for bringing about the desired separation: The clauses of domain rules must be arbitrarily ordered, and the inference procedure should not mention domain terms. We say that an inference procedure is *abstract* if it uses variables instead of domain terms [20, 31].

In this paper, we use the terms "inference procedure" and "inference strategies" and "control knowledge" synonymously. In this section, we review the methods and advantages of representing control knowledge as abstract metarules in NEOMYCIN. In the next section, we begin the study of these metarules, which unifies different ways of talking about control knowledge.

2.1. From TEIRESIAS to NEOMYCIN

The idea of abstract control knowledge evolved during the late 1970s from a combination of research:

- Davis' use of metarules in TEIRESIAS for controlling MYCIN's rules,
- Miller's [64] and Rubin's [78] studies of strategies in medical diagnosis,
- Greeno's [40] and Schoenfeld's [81] studies of strategies in mathematical problem solving,
- Brown, Collins, and Harris' [11] synthesis of strategic reasoning in algebra, reading, and electronic diagnosis.

The work in medical diagnosis and teaching made clear that there is a semantic level to control of reasoning, above the syntactic manipulation of rules or formulae. This semantic level is expressed as strategic concepts such as focus of attention, discrimination and grouping of problem solving methods, hypothesis formation, etc. The development of NEOMYCIN was an attempt to represent such strategic concepts, formalizing the work of Miller, Rubin, and related studies [19, 22]. However, the step from TEIRESIAS' metarules to NEOMYCIN is not direct. The steps in the development of the metarule language are illustrated here by an example.

First, strategic knowledge in NEOMYCIN is conceived not primarily as metarules or layers of these as in TEIRESIAS, but as procedures and subprocedures. This follows from the observation that knowledge for controlling tutorial dialogues in GUIDON [27] is naturally expressed as *ordered, conditional steps* in a procedure, not as a collection of rules. Furthermore, when describing the strategies for a complex problem such as medical diagnosis, we use procedural abstractions such as "differentiate among alternative hypotheses", which include subprocedures such as "test a hypothesis" and "determine whether a finding is present". From this perspective, metarules are the ordered, conditional steps within each subprocedure.

Second, the diagnostic metarules in TEIRESIAS are domain specific—they mention domain terms such as "pelvic-abscess" and "gram-positive rods" (Fig. 1). This prevents us from using a metarule directly in another application in a different domain—the strategy remains implicit. To generalize a domain-specific metarule, we remove the domain terms by representing the relations between them and write a metarule that uses these relations to control reasoning (Fig. 2). We now have a metarule that refers to common and unlikely causes of disorders, with statements relating infections and organisms made explicit in the domain model.

The strategy is now expressed not in terms of applying domain rules per se, but in terms of "considering a disorder", a step in a diagnostic procedure.

METARULE-1

IF: 1) The infection is PELVIC-ABSCESS, and

2) There are rules which mention in their premise ENTEROBACTERIACEAE, and

3) There are rules which mention in their premise GRAM-POSITIVE RODS,

THEN: There is suggestive evidence (.4) that the former should be done before the latter.

Fig. 1. Domain-specific metarule from TEIRESIAS.

```
What is true:

        (COMMON-CAUSE-OF  Enterobacteriaceae Pelvic-infections)
        (UNLIKELY-CAUSE-OF  G+ rods Pelvic-infections)

        Enterobacteriaceae organisms are
        <common causes of> pelvic infections.
        G+ rods are <unlikely causes of> pelvic infections.

What to do:

        SUBTASK: TEST-HYPOTHESIS
        FOCUS: $DISORDER

        IF:  (AND
                (COMMON-CAUSE-OF $DISORDER $HYP1 )
                (UNLIKELY-CAUSE-OF  $DISORDER $HYP2))

        THEN: (DO-BEFORE
                        (TASK TEST-HYPOTHESIS $HYP1 )
                        (TASK TEST-HYPOTHESIS $HYP2 ))

        <Common causes of> a disorder should be considered before
        <unlikely causes of> a disorder.
```

Fig. 2. Metarule restated abstractly by replacing domain terms by references to the relations between them.

Furthermore, in replacing the domain terms by variables, we abstract them into types, such as "disorder" and "hypothesis". The particular choice of words is not as important here as the fact that the procedures will have typed arguments, as indicated by the choice of variables $HYP1 and $HYP2 (i.e., TEST-HYPOTHESIS takes a hypothesis as an argument). Why this is so and its importance are not necessarily obvious at first; it is a pattern that is evident after abstracting a number of domain-specific control rules.

We have introduced a *classification* of domain terms in order to remove them from the metarule; this idea will return as a basic theme throughout this paper. The metarule uses *relations* by which the domain model is expressed and organized. A particular set of abstract metarules operates upon a knowledge base organized in a particular way. Operating on a knowledge base in different ways (e.g., for explanation, tutoring, or compilation) requires different procedures with different relations, and hence a reclassification of expressions in the knowledge base. This is a basic property of procedures and has important consequences for knowledge acquisition within a given task-specific architecture, as well as knowledge acquisition of new metarules.

The third step in developing the idea of abstract control knowledge is the realization that if strategies are implicit in the clause order of domain rules, then metarules must be able to reorder clauses for flexible control to be achieved. In particular, MYCIN's rules are designed to do top-down search of a disease taxonomy (Fig. 3). To consider more specific diseases first, one would

DOMAIN RULE-513

IF: 1) The infection is MENINGITIS,
2) The type of the infection is BACTERIAL,
3) The patient has undergone SURGERY,
4) The patient has undergone NEUROSURGERY,
5) The NEUROSURGERY-TIME was less than 2 months ago,
6) The patient received a VENTRICULAR-URETERAL-SHUNT

THEN: There is evidence that the organism which might be causing the infection is E.COLI (.8) KLEBSIELLA-PNEUMONIAE (.75) .

Fig. 3. Domain rule from MYCIN: when clause order matters, domain relations and strategic knowledge are implicit.

need to skip over or reorder clauses in almost every rule. Since TEIRESIAS' metarules only control the ordering of rules and not clauses, this mechanism is at the wrong grain size for expressing strategic reasoning in general. Specifically, without rewriting MYCIN's rules, we could not use TEIRESIAS' metarules to produce diagnostic search that is not top-down. Indeed, whenever rule clauses cannot be arbitrarily reordered without changing the correctness of a rule, there must be a relation between domain terms implicit in the rule. This is an important heuristic for identifying strategic knowledge in domain rules.

Figure 4 again illustrates the method of abstraction by which a relation is introduced, the SUBTYPE of a disorder. The action of the metarule corresponds to clauses (1) and (2) of the domain rule. By recursion, Klebsiella and E.coli, subtypes of bacterial meningitis, will be tested after bacterial meningitis. (The subtask PURSUE-HYPOTHESIS tests a hypothesis and then refines it, placing its children on the differential.) The resulting domain rule is shown in Fig. 5 (for brevity not all of the organisms in the original rule are listed in our examples).

Once again, the remaining clauses cannot be reordered—we want to ask about surgery before asking about neurosurgery, a strategy for minimizing the number of questions asked during the consultation pruning them at a more abstract level. This strategy is expressed in NEOMYCIN as a metarule for the subtask FINDOUT (Fig. 6). This metarule is in effect a generalization of the many "screening rules" in MYCIN (e.g., "If the patient has not had surgery, conclude that the patient has not had neurosurgery, cardiacsurgery, etc."), which were one means for removing redundant screening clauses. The advantage of this representation is that the domain relation between surgery and its subtypes is now explicit and only one rule is required.

Figure 6 also includes a definition for "recent neurosurgery", abstracting the specific time reference in the original rule. This definition is referenced by a similar FINDOUT metarule. Such an abstraction is advantageous for explana-

```
What is true:
        (SUBTYPE  Meningitis Bacterial-mengititis )
        (SUBTYPE  Bacterial-mengititis E.coli)
        (SUBTYPE  Bacterial-mengititis Klebsiella-pneumoniae)

        Bacterial-mengititis is a kind of meningitis infection.
        Diplococcus-pneumoniae is a kind of bacterial-meningitis.
        E.coli is a kind of bacterial-meningitis.

        (DIFFERENTIAL $HYP)
        There is evidence for the disorder $HYP.

What to do:
        SUBTASK: EXPLORE-AND-REFINE
        FOCUS: (the differential = the set of believed hypotheses)

        IF: (AND
                (DIFFERENTIAL $HYP)
                (SUBTYPE $HYP $CHILD))

        THEN: (DO-BEFORE
                        (TASK PURSUE-HYPOTHESIS $HYP )
                        (TASK PURSUE-HYPOTHESIS $CHILD))

        If there is evidence for a disorder,
        consider evidence for the parent before refining the disorder.
```

Fig. 4. Rule clause ordering restated abstractly by replacing domain terms by references to the relations between them.

tion to a user. Furthermore, now stated as an explicit constraint, the rule could be relaxed by the inference procedure to improve a specific diagnostic model.

In the final statement of the domain rule we have reworded the shunt concept to make explicit that it is the recent structural change that is causally significant (Fig. 7).[2] A surprising discovery is that after clause dependencies are replaced by domain propositions, abstractions, and definitions, 80% of the domain rules (of 176) have just a single clause (remaining multiple clauses usually represent conjunctive causal conditions). In fact, there is no uncontrolled backward chaining at all; all domain goals are deliberately pursued by metarules. The metarules themselves either make an assertion, request data from the user, or apply a domain rule (where again, the assertion, datum, and domain rule are all expressed by variables in the metarule). Figure 8 gives a

[2] Certainty factors in MYCIN's rules (e.g., Fig. 3) are artificially high to compensate for the effect of multiplying the conclusion's certainty by the minimum certainty from the premise clauses. When goals are nested, the multiplicative effect often results in values below the 0.2 threshold at the top of the reasoning chain (termed the "cascading certainty factor problem" [27]). MYCIN rules were written to take this propagation effect into account; for example, rules for concluding the type of the infection use 0.8 as the threshold of minimum positive evidence. NEOMYCIN handles inherited belief calculations separately, so the domain rules can use a uniform interpretation of the certainty factor scale, another advantage of our representation.

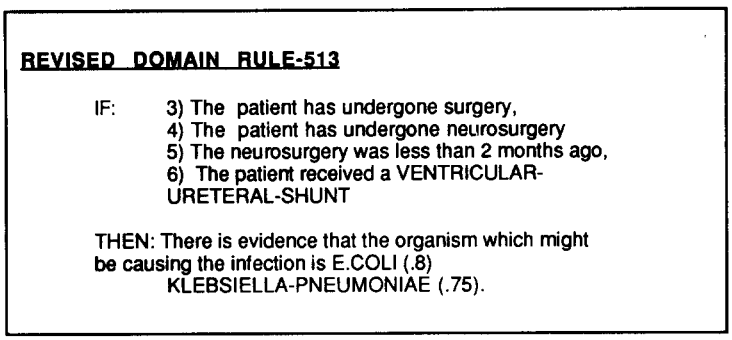

Fig. 5. Revised domain rule from MYCIN after implicit hypothesis testing strategy has been removed.

metarule for the subtask TEST-HYPOTHESIS, showing how it indirectly applies domain rules (this example should be contrasted with the metarule in Fig. 2, which orders hypothesis testing as opposed to data gathering).

In practice, we do not use the DO-BEFORE construct in NEOMYCIN. Instead, the metarule shown in Fig. 8 is written as two ordered metarules (corresponding to clauses (1) and (2) and clauses (3) and (4), both invoking

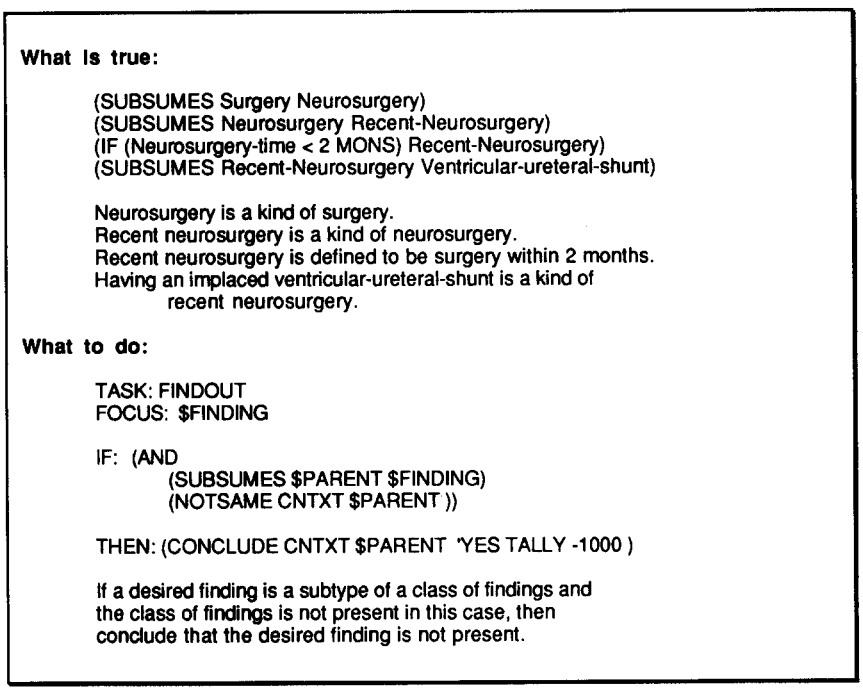

Fig. 6. Rule clause ordering restated abstractly by replacing domain terms by references to the relation between them.

NEOMYCIN'S DOMAIN RULE-513

IF: The patient has recently had an implaced ventricular-ureteral-shunt

THEN: There is evidence that the organism which might be causing the infection is E.COLI (.3) KLEBSIELLA-PNEUMONIAE (.3).

Fig. 7. Domain rule from NEOMYCIN with implicit strategies removed.

APPLYRULES). Thus, there are two kinds of subtasks, those in which the metarules constitute steps in a procedure (such as PURSUE-HYPOTHESIS, which tests before refining) and those which constitute alternative methods for accomplishing a single subtask (such as the metarules for FINDOUT and TEST-HYPOTHESIS).

To summarize, we began with a domain rule having six clauses and replaced it by subtype and causal relations and metarules for ordering data acquisition and hypothesis testing (Figs. 4, 6, and 8). Further details about NEOMYCIN's subtask and metarule language are provided in the appendices and [31]. The

What is true:

(ENABLING-CAUSE Bacterial-meningitis Exposure)
(CIRCUMSTANTIAL-CAUSE Bacterial-Meningitis Neurosurgery)

Exposure to an infectious agent is a necessary condition for Bacterial meningitis infection.
Neurosurgery is a circumstantial, unnecessary condition causing Bacterial meningitis infection.

What to do:

TASK: TEST-HYPOTHESIS
FOCUS: $HYPOTHESIS

IF: (AND
(ENABLING-CAUSE $HYPOTHESIS $FINDING1)
(EVIDENCE $HYPOTHESIS $FINDING1 $RULES1)
(CIRCUMSTANTIAL-CAUSE $HYPOTHESIS $FINDING2)
(EVIDENCE $HYPOTHESIS $FINDING2 $RULES2))

THEN: (DO-BEFORE
(TASK APPLYRULES $RULES1)
(TASK APPLYRULES $RULES2))

When testing a hypothesis, apply rules that mention findings that are enabling causal conditions before applying rules that mention circumstantial causal conditions.

Fig. 8. Metarule that invokes the domain rule interpreter.

procedural language includes inherited end conditions, control of the metarules by "do-while" and "case-statement" constructions, and primitives for accessing a history of subtask invocations.

2.2. What the relational representation reveals

Davis describes different ways for controlling reasoning, such as doing something after a goal is accomplished. However, the idea of control knowledge in TEIRESIAS was essentially that of *meta*rules, that is, rules controlling other rules. As we introduce terminology such as "finding" and "hypothesis", we find that strategies are expressed in the following terms: how to test a hypothesis, what to do when a new finding becomes known, how to infer a needed finding from an already known finding or believed hypothesis, etc. Thus, control knowledge is more specifically about the acquisition of data, the relation of data to the solution, the state of the current solution, the contrasting of solutions, etc.

Viewing the inference process just in terms of *applying rules* or *making assertions* misses the abstract patterns that constitute a common, reusable problem solving procedure for a task such as diagnosis in different domains (e.g., generalizing questions, pursuing common causes, explaining a finding). Metarules are just the notational language. *Strategic knowledge* is the recurrent collection of relations, subtasks, and ordering preferences. The possibility of this language and reusable body of knowledge was missed in TEIRESIAS because metarules were stated in a domain-specific way.

Stating the premises of metarules (and hence the domain knowledge) in a form of the predicate calculus reveals that strategic knowledge has a content that can be systematically described and shared between programs. Originally, metarule premises in NEOMYCIN were just LISP functions; however, this prevented writing a program that could explain why the metarules failed (besides requiring a redundant textual description of the metarule itself for providing strategic explanations) [44]. The restatement in a relational notation reveals how the domain terms are classified into a finding hierarchy (via the SUBSUMES relation) and a disease taxonomy (via the SUBTYPE relation). Other relations express aspects of causality between findings and diseases, as well as definitional and commonsense relations between findings (see Appendix A.5 for a complete listing). The idea of heuristic classification emerged from this analysis of the types of relations and how they were composed to link findings and hypotheses to each other. We extend this analysis in Section 6 to view the links as oriented edges in a graph (blackboard).

Perhaps more surprisingly, we find that writing a new metarule almost always requires defining a new domain relation, which further subclassifies the distinctions made before. For example, CAUSES becomes ENABLING-CAUSE and CIRCUMSTANTIAL-CAUSE. Each new relation effectively

creates a subset, intersection, or composition of the domain terms or rules defined by previously existing relations. The subtasks can therefore be viewed as operating on sets of findings, hypotheses, and domain rules, collecting, sorting, and filtering them in order to control how subtasks are accomplished. Relations in metarule premises (e.g., SUBSUMES, ENABLING-CAUSE) serve as conditions by which domain terms and rules are retrieved from the knowledge base, collected, and passed on to subtasks. We therefore find it convenient to state the metarules in terms of set operations, for example, "select *the set of findings* that are enabling causes of the hypothesis; select *the set of rules* that link these findings to the hypothesis; then apply these rules". A dimensional analysis of the subtasks in terms of typed sets makes it possible to detect missing metarules, missing subtasks, or subtasks that should be decomposed for clarity (Section 6).

2.3. Reusability of abstract control knowledge

Besides allowing us to relate different architectures—a central concern of this paper—abstract metarules also have many practical benefits, including knowledge acquisition (GUIDON-DEBUG, Section 6.3), strategic explanation [44], articulation of a relational domain model (Appendix A.5), modeling of student strategies [59, 93, 94], and line-of-reasoning strategic hints in a tutoring dialogue [77]. These applications are briefly described at appropriate places in this paper and summarized in [26].

It is interesting to consider how the notion of reusable knowledge changed during NEOMYCIN research. The first idea, stemming from MYCIN, was that a knowledge representation should be interpretable for explanation as well as problem solving. Davis nicely exploited this in the knowledge acquisition debugging dialogue of TEIRESIAS. Next, GUIDON illustrated how—with the clauses annotated to indicate their purposes—the same set of rules could be used in a case-method instructional dialogue. With NEOMYCIN came the possibility of literally reusing part of the knowledge base (and not just the backward-chaining inference engine) for different problem domains (illustrated by the CASTER sandcasting diagnosis system [89]). Thus HERACLES-DX became the first task-specific expert system architecture [21, 31].

However, a more subtle form of reusability lies within a given problem solving session itself, when a given metarule or domain fact gets used in different situations. It is no longer necessary for a knowledge engineer to redundantly encode strategies in the ordering of domain clauses, essentially restating the strategy in terms of every specific situation in which it must be used. The design is more elegant and less prone to error (Figs. 4, 6, and 8).

Furthermore, new domain facts can be added using the relational language and they will be applied appropriately by different metarules. If the program is told that the patient has neurosurgery, it can use the subsumption relation to

conclude that the patient has undergone surgery. Or if the program knows that the patient has not undergone any kind of surgery it knows about, it can use the closed-world assumption and conclude that the patient has not undergone surgery. *The knowledge base is easier to construct because the expert need not specify every situation in which a given fact or relation should be used.* New facts and relations can be added in a simple way; their meaning is procedurally represented by the abstract metarules, which concisely state how the relations will be used. The same generality makes the knowledge base more robust. The system is capable of using facts and relations for different purposes, perhaps in combinations that would be difficult to anticipate or enumerate. Figures 9, 10, and 11 illustrate the compositions that are typical, showing how forward-reasoning can lead to a focus change and hence recursive application of the TEST-HYPOTHESIS subtask.

Specifically, question (8) in Fig. 9 is directed at meningitis; after a follow-up question about seizures duration, the new information leads the program to pursue a different hypothesis, intracranial mass lesion. Thirteen metarules are on the line of reasoning between questions (9) and (10), shown in Fig. 10. There are two hypothesis-directed inferences (annotated as {1} and {4}) and two data-directed inferences ({2} and {3}), which are shown graphically in Fig. 11. The order in which the links of the general model of Fig. 11 are interpreted is not coded into the network; metarules can index and apply these relations in either a hypothesis or data-directed way, dependent on the problem solving situation.

NEOMYCIN's metarules were originally developed to enable a tutoring program to converse about diagnostic strategies. Perhaps the most revealing incarnation of this capability is in GUIDON-MANAGE, an instructional program in which the student issues commands to NEOMYCIN using the subtask language [77]. The instructional intent is to give the student a language for talking about the diagnostic process, providing a means for detecting missing domain knowledge, as well as a means for learning from observation of teachers and fellow students. For example, the student might say, "I know that I should test the hypothesis of intracranial pressure; is there an ENABLING-CAUSE I should know?". Figure 12 illustrates how the subtask stack from Fig. 10 is interpreted to generate hints for the student.

```
7) What is Susanne's temperature?
** 105.8 farenheit
8) Has Susanne experienced seizures recently?
** YES
9) What is the duration of Suzanne's seizures?
** 1 HOUR
10) Does Susanne have an abnormal fundoscopic exam?
** NO
```

Fig. 9. Sequence of questions from a NEOMYCIN consultation.

{1. Top of the line of reasoning; we are pursuing meningitis as a generalization of some hypothesis triggered by the initial data}

CONSULT
MAKE-DIAGNOSIS
COLLECT-INFO
ESTABLISH-HYPOTHESIS-SPACE
GROUP-AND-DIFFERENTIATE
TEST-HYPOTHESIS **[Meningitis]**
APPLYRULES [Rule060 Rule323]
APPLYRULE! [Rule060]

{2. After finding out about seizures to apply rule 60, we consider other data-directed inferences; the follow up question (#9) about seizures duration is generated; then rule 262, marked "antecedent," is applied}

FORWARD-REASON
PROCESS-FINDING **[Seizures]**
APPLYRULES.ANTE [Rule262]
APPLYRULE! [Rule262]

{3. Rule 262 concludes that seizures might also be caused by increased intracranial pressure; is that linked to anothing else we have been considering? It might be explained itself by an intracranial mass lesion, but more evidence is required before the rule can be applied. Test-hypothesis is now invoked recursively; a focus change has occurred}

FORWARD-REASON
PROCESS-HYPOTHESIS **[Increased-Intracranial-Pressure]**
APPLYRULES.ANTE [Rule239]
APPLYRULE! [Rule239]
FINDOUT **[Increased-Intracranial-Pressure]**
TEST-HYPOTHESIS **[Increased-Intracranial-Pressure]**
APPLYRULES [Rule209 Rule233 Rule373]
APPLYRULE! [Rule209]

{4. Rule209 requires information about papilledema; the inquiry is generalized to fundoscopic-abnormal, question #10}

FINDOUT **[Papilledema]**
FINDOUT **[Fundoscopic-Abnormal]**

Fig. 10. NEOMYCIN subtask stack, indicating that thirteen metarules lie between questions (9) and (10) (Fig. 9), with a focus change from meningitis to increased intracranial pressure. (Numbers in brackets correspond to domain rules in Fig. 11.)

In this sequence of hints, the program takes the student down a line of reasoning, beginning with the most abstract or highest subtask on the stack. In generating this subtask stack, GUIDON-MANAGE simulates what NEOMYCIN would do in the current situation. Because the subtasks are stated abstractly and refer to the evolving model of the patient, a separate data structure, they can be run at any time in any order. We were surprised that the translations of the subtasks, which can be too abstract in consultation explanations, are appropriately general when provided as hints. As can be seen by comparing Figs. 11 and 12, GUIDON-MANAGE uses heuristics for compressing the sequence, including direct statements of domain rules (rather than

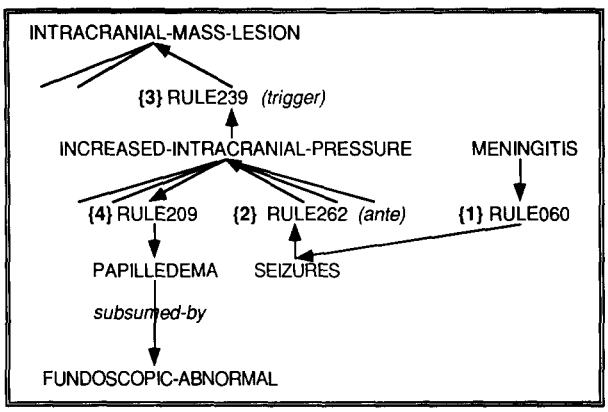

Fig. 11. Interpretation of domain relations in specific case, showing mixture of hypothesis-directed ({1} and {4}) and data-directed reasoning ({2} and {3}). All relations are causal, except for subtype relations between papilledema and fundoscopic-abnormal. (Numbers in braces correspond to subtask stack fragments in Fig. 10.)

saying "apply a rule") and special treatment of metarules with CONCLUDE actions. The classification of subtasks required to do this interpretation further illustrates how new distinctions are typically added to a representation in order for it to be applied by a new inference procedure. In this case, the subtasks are classified for GUIDON-MANAGE's simulation and hint procedures.

3. Three views of inference

To this point, we have described the method of representing control knowledge abstractly and illustrated its benefits. We are ready to study NEOMYCIN's subtasks and metarules. Our intuition is that the inference procedure is not an arbitrary program, but must be based on some logic, which could be used for driving the process of writing new metarules and evaluating their completeness and correctness. For example, can we provide a student with a global orientation that transcends the design of individual subtasks and metarules? Viewed as a task-specific knowledge acquisition tool, how is the HERACLES-DX model of diagnosis different from that in other programs?

More basically, how can we describe what an inference procedure is doing? Paralleling developments by other researchers, our view of inference has evolved through three perspectives:

- chaining goals and rules, such as a line of reasoning in MYCIN's explanations, an inferential or *means–ends analysis perspective*;
- traversing causal–associational networks, such as the path from a finding to a syndrome in CASNET, *a domain relation perspective*;

```
      Name          Age   Sex          Race
1)**  Susanne       44    FEMALE       CAUCASIAN
2) The chief complaints:
** HEADACHE
** PHOTOPHOBIA
** FEBRILE

NEXT TASK: CLARIFY-A-FINDING HEADACHE
3) How long has Susanne had this kind of headache?
** 6 HOURS
. . .
NEXT TASK: TEST-HYPOTHESIS MENINGITIS
8) Has Susanne experienced seizures recently?
** YES

NEXT TASK: CLARIFY-A-FINDING MENINGITIS
9) What is the duration of Suzanne's seizures?
** 1 HOUR

NEXT TASK: HINT
Determine the implications of a new finding.

NEXT TASK: HINT
Try focusing on seizures.

NEXT TASK: HINT
Seizures can be caused by increased intracranial pressure.

NEXT TASK: HINT
Determine the implications of a new hypothesis.

NEXT TASK: HINT
Try focusing on increased intracranial pressure.

NEXT TASK: HINT
An intracranial mass lesion can cause increased intracranial pressure.

NEXT TASK: HINT
Decide whether Susanne has increased intracranial pressure.

NEXT TASK: HINT
Increased intracranial pressure can cause papilledema.

NEXT TASK: HINT
Attempt to conclude about a more general finding.

NEXT TASK: HINT
10) Does Susanne have an abnormal fundoscopic exam?
** NO
```

Fig. 12. Excerpt from GUIDON-MANAGE illustrating generation of hints from NEOMYCIN's subtask stack (Fig. 10). (Student inputs are in bold font.)

- Constructing a situation-specific model, such as the patient-specific description of pathophysiological processes in ABEL and NEOMYCIN, *a system-modeling perspective.*

These perspectives are illustrated and discussed in turn.

3.1. Chaining goals and rules

From the early days of MYCIN, we wanted a way to describe a knowledge base abstractly, above the level of individual rules, if for no other reason than to teach novice knowledge engineers how to make large-scale changes to the rule set. Because procedural knowledge is encoded redundantly in rule premises, we found that there are characteristic patterns in how goals are related, independent of the patient case. For example, whenever a rule for determining the type of bacterial meningitis is applied, the program always determines first whether the patient has an infection and then whether it is meningitis. This is independent of which bacterial meningitis rule is applied first, because they all have the same initial clauses. We termed this recurrent pattern the *inference structure* of a knowledge base. Bennett demonstrated the usefulness of the idea for structuring initial knowledge acquisition sessions in ROGET [5]. GUIDON presents the inference structure of a knowledge base when outlining the main goals for approaching a new case [28].

Figure 13 shows part of the inference structure of MYCIN corresponding to the example discussed in Section 2. For the reasons already discussed, an AND/OR tree of domain goals and rules—a common way of describing rule-based inference—is not an adequate description of MYCIN's reasoning. Subtype relations are implicit in the terminology and attribute–value relations (compare the terms INFECTION, SUBTYPE, and COVERFOR to the representation used in Fig. 4). Clause order does not distinguish between logical conjunction and procedural relations. In short, the inference procedure is implicit, handicapping the debugging, explanation, and instructional programs that must reason about it. Another way must be found for describing the inference process.

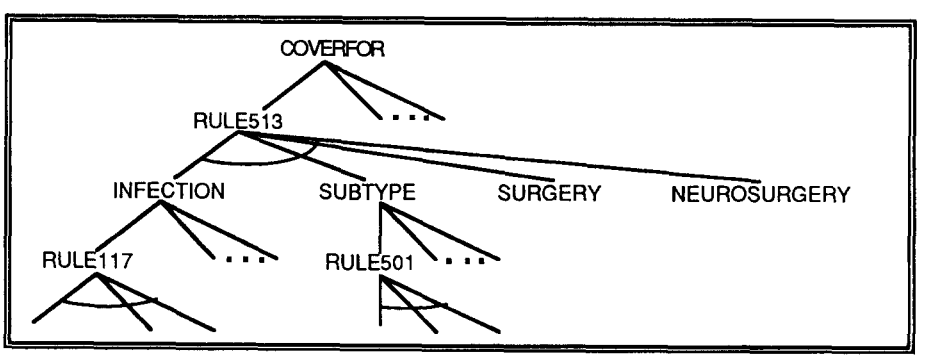

Fig. 13. Inference structure: the AND/OR tree of rules and goals generated by backward-chaining through rules.

3.2. *Traversing causal–associational networks*

Concurrent with MYCIN's development, other researchers were representing disease knowledge separately from the diagnostic procedure. Szolovits and Pauker [88] distinguish between categorical and probabilistic strategies for integrating evidence. Categorical evidence is used to rule out general processes (e.g., a traumatic problem is ruled out because the patient has not fallen recently). Probabilistic evidence is used to refine and sort out specific processes (e.g., enterobacteriaceae organisms are common causes of bacterial meningitis, so they are considered first). Thus, a general inference strategy is described in abstract terms, referring to subtype and causal relations among domain processes.

An even better example is provided by CASNET [55], in which a satisfactory diagnostic explanation of an abnormal finding is characterized in terms of a path of confirmed nodes on a path from the finding, through intermediate pathophysiological states, to syndrome types (which can then be heuristically related to therapies). Thus, a general inference strategy is described in terms of traversing paths in the knowledge base.

We adopted this perspective as the first way of describing NEOMYCIN's diagnostic strategy (Fig. 14). The figure shows a portion of a disease taxonomy. Initial information about a patient triggers an arbitrary hypothesis, such as Chronic-meningitis. The subtask ESTABLISH-HYPOTHESIS-SPACE leads the program to first look up to more general categories (GROUP-AND-DIFFERENTIATE), then to look down to establish specific causes. Thus, a general inference strategy is described in terms of the global, hierarchical structure of the knowledge base.

These examples are valuable for understanding the diagnostic process. The task-specific nature of diagnosis cannot be described in terms of applying rules or backward-chaining alone. Diagnosis involves considering categories of evidence, establishing causal–associational paths between findings and disease processes, and contrasting alternative paths and subprocesses (subtrees). In Fig. 14 we are viewing the knowledge base as a representation of processes that can occur in the system being diagnosed and viewing inference in terms of how this representation is searched.

Of particular importance is the idea that *diseases are processes*. The levels of NEOMYCIN's disease taxonomy are annotated in Fig. 14 to indicate how the description of the processes gets more specific as we move down the hierarchy. Each level adds a particular characteristic to the process description: the location, its duration, the agent that caused it (of course, not all subtrees have the same levels; trauma does not involve agents). A disease taxonomy can be viewed as a classification of abnormal processes, just specific enough to distinguish between therapy alternatives (e.g., viral meningitis is not broken down because all types are treated the same way). The top-level processes can

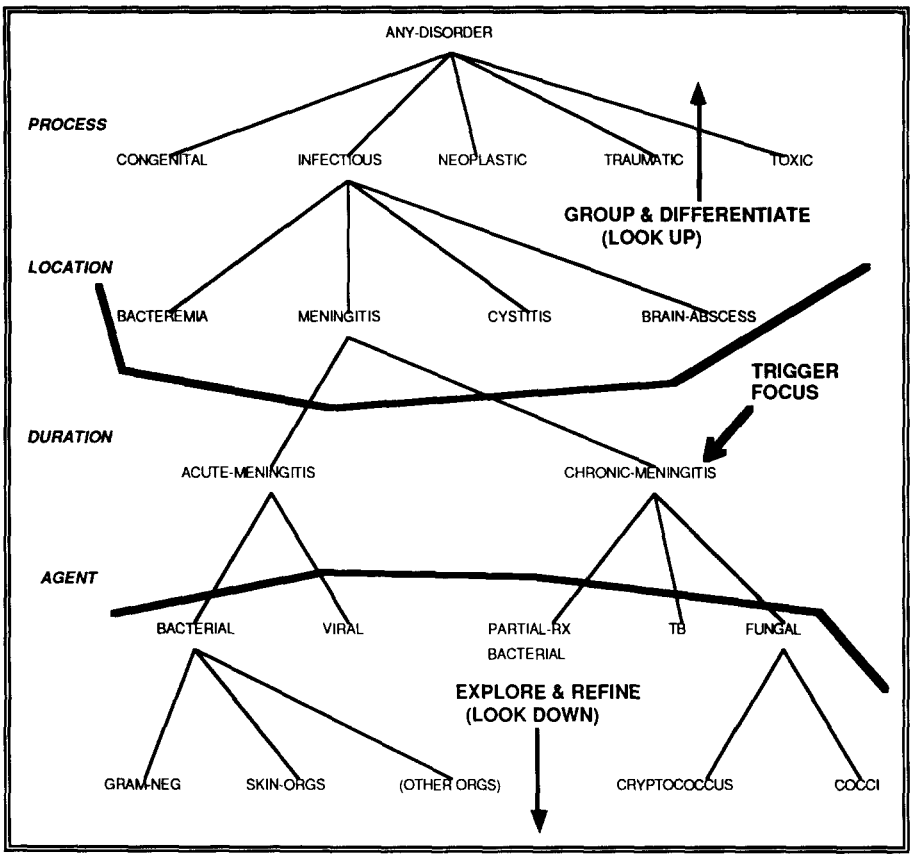

Fig. 14. Looking up and looking down in diagnostic search: viewing inference as searching the domain model.

be generalized: assembly flaw (congenital), environmental influence (infection, toxicity, trauma, psychological overload), and structural degeneration (vascular disorder, immunoresponse, muscular disorder). In contrast, in CASTER, an expert system built within the same architecture, the top-level abnormal processes correspond to what can go wrong during *stages* in the sand casting process (wooden pattern design, sand molding, metal melting, pouring, cooling/venting, etc.).

Both NEOMYCIN and CASTER contain models of physical systems. Externally observable manifestations of the system's behavior are explained in terms of internal system behavior (e.g., increased intracranial pressure), using a state-transition network (not shown here, but discussed in Section 7), and then tracked back to faulty structures and malfunctions of subsystems. These internal aberrations are then explained in terms of the etiologies, or final causes, of the disorder taxonomy, emphasizing processes in which the system interacted with its environment, bringing it to its current state. In medicine,

these etiologies include *congenital* problems (which may be caused by the mother's lifestyle or her environment), *psychogenic* problems such as emotional stress, *trauma* that structurally damages the body, food toxicities, etc. In the human body, internal systems regenerate new subsystem structures, so developmental and degenerative processes are central.

Stating inference in terms of searching a representation (the knowledge base) makes the structure of the knowledge base a subject of study. What are the different ways of organizing an abnormal process classification? How is the nature of the system in the world that is being diagnosed reflected in the process representations we use? Are there abstract categories for describing processes (such as environmental interaction processes) which we can use when approaching a new domain, to organize the questions we ask a domain expert or the cases we choose? We consider these questions further in Section 7, after considering a number of other examples.

3.3. Constructing a situation-specific model

Describing inference in terms of searching a domain model is useful, but it fails to characterize the purpose of the inference procedure. What is a good diagnosis? More generally, how can we relate the inference process to the goals an inference process is attempting to accomplish? From the perspective of these questions, the idea of looking up and down hierarchies is too superficial. Why should we look up before looking down? What underlying constraints suggest this procedure? Answers to these questions come from new ways of visualizing NEOMYCIN's problem solving process.

In GUIDON-WATCH, we were trying to find useful ways to visually present the program's reasoning, both as a debugging aid and for instructional explanations [76]. Although it may seem obvious now, it required many months to realize that the program's inferences could be shown as graphs linking the program's final diagnoses to the findings that support them (Fig. 15). Indeed, this representation shows that the program's solution is not the name of a disease, such as acute bacterial meningitis—our way of talking about program output since the early days of MYCIN—but rather a causal argument having the structure of a proof, called the *situation-specific model* (SSM).

Our conception of the situation-specific model combines two ideas from previous work. In Patil's ABEL program [74], a causal explanation is represented as a five-layered graph containing the particular findings, disorders, and their causal or subtype relations that are believed to be present in the particular patient being diagnosed. In ABEL this graph is called the patient-specific model (PSM) [60]. Although we knew about the PSM idea, we did not draw NEOMYCIN's solutions in this way because we did not think of the program this way; we thought we had a different kind of program, one without a PSM. As will become clear, *this single-minded perspective leads expert system*

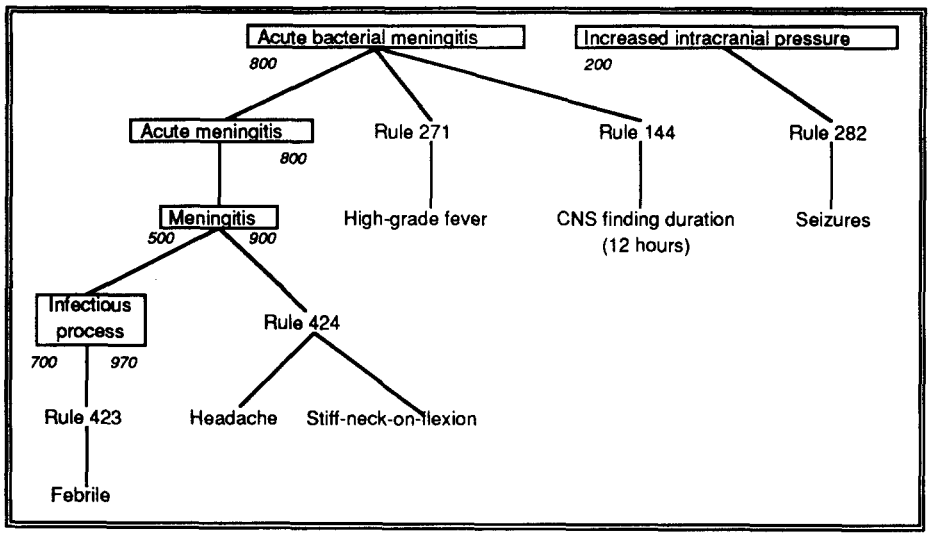

Fig. 15. Situation-specific model in NEOMYCIN.

researchers in general to believe that a blackboard depiction of a program's solution is something unique to a particular class of programs.

With the PSM idea in mind as an elegant way of describing the diagnostic process, we were struck by Anderson's use of a graph to represent a geometry proof [3]. Anderson replaces the standard linear two-column table of assertions and justifications by a graph linking the theorem to be proved to the intermediate axioms, theorems, and finally the terminal nodes of given facts. This representation, very similar to Fig. 15, helps a student keep track of the current state of a proof, revealing gaps, assertions that do not lead anywhere, and unused information. Furthermore, the graph reifies the possibility of working top-down from the theorem to be proved or bottom-up from the given information. The graph shows that a proof is a structure with certain properties: It is a connected graph of assertions, with each assertion supported by known theorems or given information.

From the perspective of diagnosis, this proof is *not just an arbitrary belief dependency graph*; it is an object having a particular, preferable structure:

- There should be just one graph containing all abnormal findings (a single-fault assumption).
- Abnormal findings (terminal nodes) are causally explained by hypotheses they are said to support (e.g., headache is explained by a meningitis process).
- Hypotheses and hence explanations are more specific higher in the graph (e.g., acute bacterial meningitis is more specific than meningitis) and hence more specific characterizations of what is going on in the system

being modeled are incorporated in the model (e.g., an infectious process explains the fever; acute bacterial meningitis accounts for the fever being high and its duration).

- The root diagnosis (shown at the top) should be specific enough to select among competing therapies (e.g., acute bacterial meningitis is not specific enough because bacteria respond to different antimicrobial drugs).

Thus, the particular relations of cause and subsumption are oriented so higher nodes cause or are subsumed by lower nodes. The form of the graph as a single, connected network, with the root being specific enough to be useful for repair of the system being diagnosed, relates to the task-specific nature of the inference process: We are attempting to construct a specific enough model of what is occurring in a particular physical system so we can account for all our observations and be able to modify the system to make its behavior normal again. These constraints can be formalized and used for explanation, teaching, or to generate the inference procedure itself (Section 6).

For our purpose in describing inference, the idea of a proof is not as important as the idea of a *a model graph constrained to have a certain form.* The form—specified as constraints on the structure of the graph—arises because the graph is not an arbitrary network, but is a representation of processes occurring in a physical system, constructed for a purpose. (In particular, if our purpose were scientific description rather than medical therapy, we would want to know what causes viral meningitis and would not say that the model is satisfactory when that node is the root.) The idea that the graph describes a process (e.g., in the CNS) was not emphasized in ABEL or most other programs, and is an important step for realizing what is fundamentally new in AI programs: Systems are modeled not just in terms of numeric measures, rather they are internally described in terms of spatial, causal, and temporal relations among objects and events. Simply put—to make the distinction with traditional numeric programs—expert systems model processes qualitatively.

The key idea for our argument is that this internal description, the SSM, is inspected by the program itself during reasoning, and its partial state, viewed structurally in global terms (as opposed to looking only for specific assertions), is used to drive the inference process. Specifically, we can view NEOMYCIN's subtasks as operators that examine and modify links in the SSM (Fig. 16). Figure 14 already suggests that inference subtasks are not just arbitrary procedures. Rather, *subtasks can be viewed as operators that traverse different types of links or traverse them in a particular order.* The more powerful perspective we are suggesting here is that these operators are placing nodes and links in the SSM, and the process of searching the general model (e.g., the disorder taxonomy) is secondary, a matter of finding the processes that might be occurring in this case.

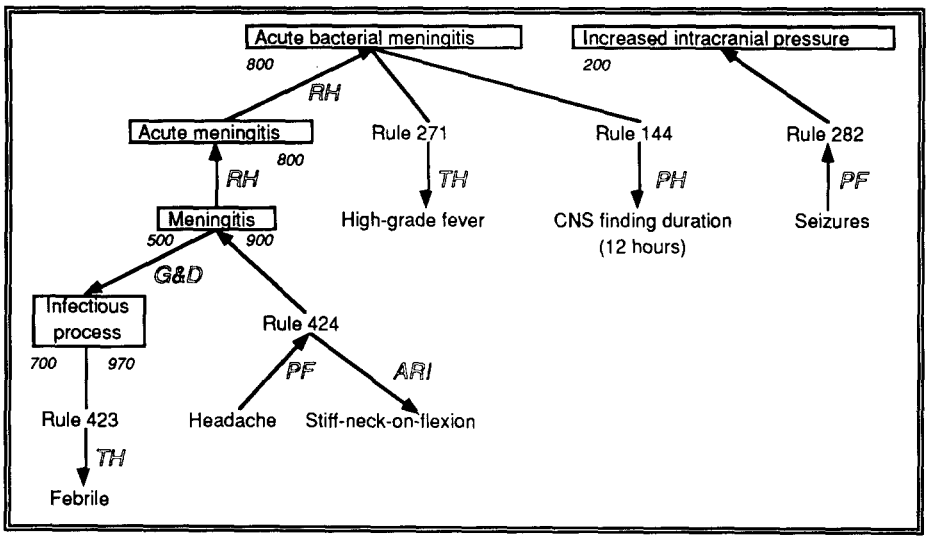

Fig. 16. Diagnostic subtasks shown as operators for constructing a situation-specific model (arrows indicate the order in which nodes were linked; see text).

Subtasks in Fig. 16 are indicated by abbreviations. The example presented in Figs. 9–11 is presented here in terms of constructing a model of processes causing the patient's complaints. Information about headache triggered Rule424, which led a question about stiff-neck-on-flexion to be asked and meningitis to be hypothesized (subtasks PROCESS-FINDING and APPLY-RULE!). The subtask GROUP-AND-DIFFERENTIATE looked for support for meningitis by considering categorical evidence (for infectious process, a more general process description). To use the node and link language more explicitly, TEST-HYPOTHESIS grows links downwards, here placing febrile and a causal link (Rule 423) below infectious process. Similarly, REFINE-HYPOTHESIS grows subtype links upwards, here placing acute meningitis and then acute bacterial meningitis in the SSM.

The idea of inference operators is developed well by Pople [75] in CADUCEUS and also in ABEL. However, the more powerful formulation we advocate involves a third shift in perspective, namely shifting from talking about inference in terms of the knowledge base (as in CADUCEUS) to *talking about the specific model the program is constructing of the particular system it is diagnosing* (as in ABEL). PROCESS-HYPOTHESIS makes a connection to previously known information, growing a link to the CNS duration (which was itself placed in the SSM by PROCESS-FINDING, in abstracting the headache duration). Similarly, we see PROCESS-FINDING growing a link to increased intracranial pressure, at this point a disconnected, competing explanation of what is occurring in this patient.

The explanatory value of the SSM for characterizing the inference process is particularly clear when there are two or more disconnected graphs, as in Fig.

16, where the incomplete nature of the solution is evident. Specifically, the work to be done is made explicit by the form of the SSM: Can a link be drawn between seizures and the acute bacterial meningitis graph? In terms of the relations required by the operators, could seizures be caused by acute bacterial meningitis? Could increased intracranial pressure cause acute meningitis? Of course there are many such questions that might be asked. The important point is that looking at such a representation of the inference process suggests *questions about the general model* that could improve the current solution.

More specifically, it is our knowledge of the form of an adequate solution—the constraints listed previously—that allows us to criticize an SSM. It is our knowledge of operators and link types that allows us to translate these gaps into general questions. A student looking at Anderson's proof graphs can ask similar questions. For example, how could I prove that AB is congruent to CD? How could I use the information that angle ABC is equal to angle ABD? When we consider how we rely on a situation-specific model for calculations as well-defined as subtraction or division, we realize that portraying the diagnosis process by a linear sequence of questions, typical of consultation programs, is showing just the superficial behavior of the program.

Probably the biggest surprise is that the *subtasks can now be formally described in terms of the particular nodes or links they place in the SSM* (with more-abstract operators controlling how these primitive operators are applied). This is quite a big step from the original implementation of the metarule premises as LISP code, with abstract terms like ESTABLISH-HYPOTHESIS-SPACE being the only theoretical language for describing what the subtasks were doing. This dimensional analysis is presented in Section 6.

The idea that NEOMYCIN's subtasks are constructing a model may appear to be at odds with the claim that it selects a model by classification. Section 7 resolves this possible confusion by contrasting selection of a process description (such as acute bacterial meningitis) with construction of a new process description that accounts for (or designs) process interactions.

Once we realized the value of the SSM for detecting the adequacy of the program's diagnosis, we began to print it routinely and use it as the starting point for knowledge acquisition discussions with our physicians. We also developed a program called GUIDON-DEBUG that detects gaps in an SSM and reformulates them as questions about missing propositions or rules in the knowledge base (Section 6).

Before reviewing how the system-model-operator perspective applies to other programs (Sections 4 and 5), we show how different perspectives on the nature of inference are manifested in different research emphases.

3.4. Drawing the elephant: piecing together different perspectives

The system-model-operator perspective provides a way of integrating how

different researchers have described their programs and their research objectives. We consider designs for modeling tools, spaces for describing inference operators, inference procedures contrasted with inference engines, and ways of classifying expert system tasks.

3.4.1. Modeling tools

Alternative views about the nature of inference strongly influence our beliefs about what a tool might do for us and hence influence the tools that we design. Tools can be viewed as using different *perspectives for asking questions about a representation*, ranging from a microlevel of nodes and links to a macrolevel of paths and subgraphs:

- *node*: terminological consistency (e.g., subsumption in NIKL, where should a new concept be placed in a classification?);
- *path*: inference nets (e.g., use/conclude browsing in EMYCIN, what rules conclude that the organism causing the patient's infection is E.coli?);
- *subgraph*: logical dependency (e.g., common to belief maintenance systems, includes "what-if" reasoning, backtracking, alternative models/worlds, detecting contradictions, endorsement/justification relations);
- *graph*: model construction (e.g., operators for detecting completeness/coverage or adequacy for task/specificity, scenario-generation programs, database discovery).

The system-model perspective suggests that these are not different kinds of programs per se, but operations that are conceivably desirable for any expert system over the range of its development, maintenance, and use for multiple purposes. This analysis suggests that we might gain leverage by integrating tool capabilities rather than pursuing them in isolation. For example, consider the benefit of relaxing NIKL's classifier to modify ABEL's general model in order to improve a particular diagnosis.

3.4.2. Operator spaces

Figure 17 illustrates a second way of piecing together alternative views. We can understand different program descriptions in terms of the space that researchers have used for defining inference operators. Programs like CENTAUR [1] and CASNET are described in terms of operators for searching the domain theory space of subtype and causal relations.[3] In NEOMYCIN, ABEL, etc., operators are described with respect to the form of the situation-specific model. Other researchers have focused on the problem of deciding what operator to apply; for them inference is described in terms of impasses,

[3] CENTAUR is a kind of half-breed between MYCIN and NEOMYCIN. The disease classification is explicit, but strategies are domain-specific (e.g., "after confirming that the patient has an infection, determine the specific disease").

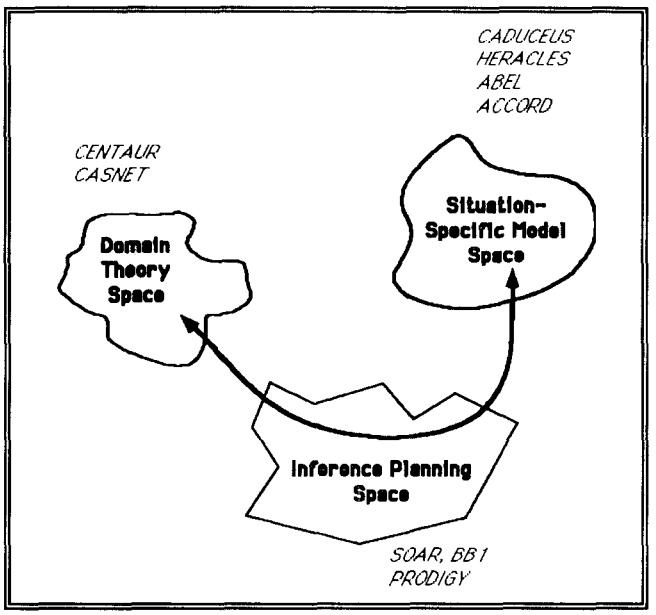

Fig. 17. Alternative spaces for defining inference operators.

agenda, a control blackboard, operator preferences, etc. As Fig. 17 shows, these are not three kinds of programs per se or specific to kinds of problems (e.g., design contrasted with diagnosis). Rather these are points of flexibility open to every program in every problem. By making explicit that these different perspectives exist, we can begin to study how they interrelate and how specific strategic knowledge in each space relates to the constraints posed by the structure of the domain theory, the structure of the situation-specific model, or the structure of the environment in which inference must take place (e.g., resource limitations that make planning useful). Indeed, the study then moves to characterizing *types of structures* and how they relate to *types of systems* to be modeled (Section 7).

3.4.3. Levels of interpretation

A third integration is possible by reconsidering how the idea of *interpretation* is used for describing inference. Figure 18 illustrates the view that an inference procedure constructs a situation-specific model of some system.

It may not be obvious at first that this diagram is not specific to diagnosis; rather it describes *what every expert system does*. Specifically, for design problems the general model describes particular structures and how they can be assembled to produce particular functions or processes. The data includes constraints on the cost, manufacturing process, environment of use, etc. The SSM is a description of the designed system. In this way, we can characterize

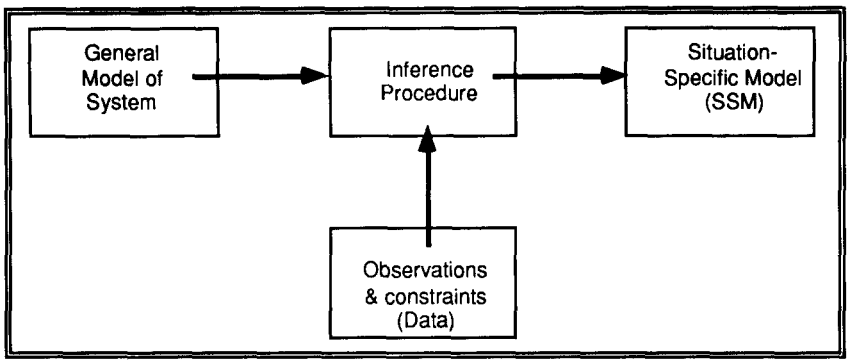

Fig. 18. An inference procedure is a program for gathering information about system behavior and the environment (data) in order to make assertions about the system producing this behavior (diagnosis), the system that could produce this behavior (design and control), or how a system will behave (prediction).

diagnosis as an analytic problem (describing processes in an existing system), and describe design as a synthetic problem (designing a new system). For the task we call control, the SSM describes how the system should be configured (and possibly what the input should be) to produce desired behavior. For assembly, the SSM describes a containing system, a manufacturing process, that will produce the desired system. A dimensional analysis of these possibilities relating to input/output relations is provided in [24].

Figure 18 suggests that *the idea of an inference engine interpreting a knowledge base is inadequate for describing inference*, particularly for contrasting inference procedures for different kinds of tasks. Figure 19 suggests that the inference engine idea, exemplified by EMYCIN's backward-chainer, has probably evolved from how we view traditional computer programs, which are interpreted or compiled by other programs, which itself developed from the idea of "programs as data".

It is intriguing to realize that just as MYCIN's inference procedure is lost inside its domain rules, the inference procedure in programs like CASNET is lost inside LISP code. Viewed this way, the frame–rule controversy of the 1970s was partly a matter of competing perspectives: Researchers using frames realized the importance of stating domain knowledge as propositions (viewed as networks of concepts), separately from the programs that use the knowledge. Researchers using rules realized the value of stating all knowledge in a stylized language so it could be annotated and hence interpreted by different programs for multiple purposes.

The description of inference operators in ABEL and CADUCEUS is a major step towards integrating these views, but these operators are still coded in a way that is not interpretable in multiple ways for explanation, student modeling, etc. *The major contribution of NEOMYCIN is to identify the*

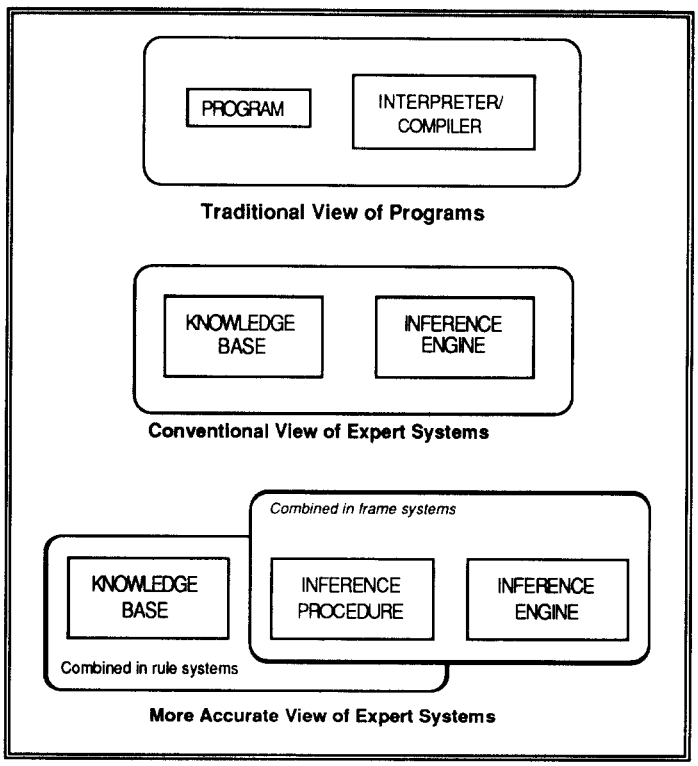

Fig. 19. An inference procedure interprets a task-specific representation; an inference engine is a programming-language interpreter.

inference procedure as something that must be represented in a stylized way, and by its formalization can be studied as an object in its own right. As we will see, other researchers view strategic knowledge as structured, but have not represented all levels in a stylized way. For example, strategic knowledge sources in blackboard systems are generally LISP functions or rules with LISP premises (Section 4). NEOMYCIN's architecture suggests three tiers: a knowledge base expressed as a set of propositions, an inference procedure that references the relations of the knowledge base, and an inference engine that applies the subtasks and metarules in which the inference procedure is encoded. Procedural knowledge in XPLAIN is coded as stylized procedures, but the particular statements are not objects in their own right, which can be annotated, flexibly interpreted, or reordered, an advantage of the metarule formalism (Section 8).

As indicated in the discussion of GUIDON-MANAGE, it is the ability to store facts about inference procedure constructs (subtasks, metarules, premise relations) that enables it to be used for multiple purposes. Just as the inference procedure relies on the classification of domain terms to interpret the domain model, a program like GUIDON-MANAGE relies on the classification of sub-

tasks and metarules to generate hints (e.g., subtasks not to mention when generating hints include APPLYRULE! and APPLY.ANTECEDENT.RULES). Similarly, the compiler that produces LISP code from the predicate calculus representation of metarule premises uses a classification of relations that describes how they are implemented in the underlying LISP (e.g., SUBSUMES is implemented as a property list attached to the first term of the relation; see Appendix A.4). Other researchers have emphasized the relation between control knowledge and classifications (e.g., [41]); the model construction perspective reveals *the role relations play* in selecting operands of SSM operators.

The idea that an inference procedure in effect asks questions about the terms it is manipulating is illustrated by Fig. 20. The inference procedure for GUIDON-MANAGE asks questions about subtasks and the metarule compiler asks questions about domain relations. Although the inference engine also asks questions about the inference procedure, these are part of the language (e.g., the idea of an iterative subtask) and never introduced by the knowledge engineer. In contrast, a knowledge engineer can modify existing inference procedures, adding new relations by which the knowledge base is defined. This distinction is of course just relative, consistent with the idea that a programming language (such as the subtask/metarule language) is relatively fixed and not modified by its users, a constraint that enables programmers to share tools.

3.4.4. Process modeling

To summarize, an SSM is a model of processes occurring in some system. Many researchers call the SSM a *blackboard* and emphasize that domain knowledge can be organized in terms of changes to the blackboard. In NEOMYCIN, we make explicit that these changes are made by inference operators, which reference relations by which the domain knowledge is expressed and hence structured. Several key ideas come together here that in some respects violate common ways of describing expert systems:

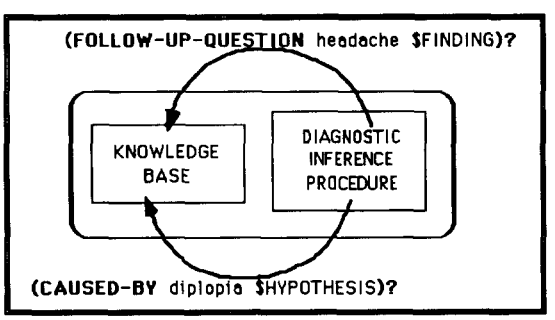

Fig. 20. An inference procedure inspects the general model by seeing if particular relations hold between the terms it is manipulating (e.g., domain terms, subtasks, metarules).

- A knowledge base contains a general model of some system (a description of an abnormal process, such as a disease, is a kind of model).
- An inference procedure constructs a situation-specific model of the particular structures and/or processes occurring in a desired or actual system.
- Both the general and situation-specific models (and indeed the inference procedure) can be represented as relational networks.
- Inference procedures can be described in terms of operators for placing nodes and links in the SSM.

The strong claim here is that all expert systems have an SSM, although researchers may not visualize the program's assertions this way (and hence the SSM's structure may be unarticulated or nonsystematic, as in MYCIN, Section 4.5). The advantages of the SSM perspective for tracking the completeness and consistency of a program's "solution" suggest that every expert system should have a blackboard; that is, the SSM should be routinely displayed to reify the program's operation. The SSM can be studied to compare and criticize inference procedures (Section 6). The important realization that enables this form of analysis is that *all expert systems are constructing and using models*; *all expert systems are model-based*. This term has been used too restrictively in the literature to refer to programs with simulation, as opposed to classification, models (Section 7). Indeed, there is strong reason to believe that classification models are necessary for modeling open systems and cannot be reduced to structure-function simulation models.

Generalizing a step further, we can *characterize the general inference structure of expert systems in terms of a sequence that involves constructing a model and then using it for some purpose* (Fig. 21).

Broadly speaking, for design we start with a description of desired behavior, produce a system design, and possibly an assembly plan. For diagnosis we start with actual behavior and identify what system is producing it, often followed by a modification (repair) or control plan. Either design or diagnosis might make predictions about behavior for testing a design (to see if it meets specifications), hypothesizing findings to confirm disorders, or monitoring a therapy plan. Of course, the inference process can iterate over these steps, for

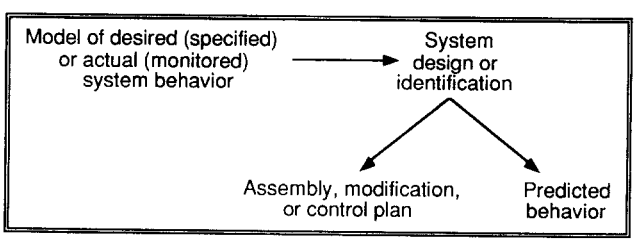

Fig. 21. General inference structure for reasoning about systems.

example, moving from a hypothesized design back to inquire about further specifications.

This diagram shows that planning depends on having a model of a system to be assembled, modified, or controlled. Indeed, it violates all common sense to say that MYCIN could prescribe appropriate therapy without having a model of the patient. Furthermore, planning is not something to be viewed in isolation, as something done by a particular class of programs, rather it is an integral aspect of all expert systems. (Otherwise, there would be no reason for constructing the SSM; see Section 9 for a discussion of decision support systems, which fit this description.) We might say the model is not the usual *solution* of expert systems, however plans themselves are models of systems that will behave in some way, often involving some configuration of processes that will interact with the object system (e.g., a therapy plan may include a combination of drugs, food, exercise, and later visits to the physician). (See [24] for related discussion.)

This diagram can serve as a template for describing any given expert system. *The idea behind task-specific architectures is that particular problems can be viewed as recurrent specializations of this diagram.* This seems clear enough given the examples of diagnostic and design architectures to date. For comparison, contrast this diagram with early CAD systems, which did not have the capability to construct the electronic or architectural model from input specifications and were not fed into automatic manufacturing or construction-planning systems. It is the internal SSM and inference operators for constructing it that enables this cascading of model construction and use in expert systems.

Our next step is to study operators for constructing SSMs, study the structure of SSMs, and determine how operators and SSM structure relate. If SSM structures recur, then presumably operators will recur. We will then be able to characterize a generic system, say a diagnostic expert shell like HERACLES-DX, in terms of the structure of its SSM and the corresponding operators of its inference procedure.

4. Blackboards and operators

To illustrate the central importance of the system-model-operator perspective, we review here several programs with an explicit SSM-operator design. This reveals the usefulness of the perspective for describing reasoning, the prevalence of the perspective in 1980s' research, and how the subtask/metarule formulation adds to previous accounts. We begin with CADUCEUS, ABEL, and ACCORD, which provide diverse examples of model construction operators. We then analyze HASP to show how the traditional blackboard architecture alone does not make explicit the structure of the SSM or the

operators; blackboards and knowledge sources are shown to have a regular structure in process-graph-operator terms. Finally, we use our new-found perspective to resolve an old puzzle—what is MYCIN's context tree?

4.1. *CADUCEUS's model construction operators*

Pople [75] describes diagnosis as a process of constructing a "diagnostic task" (an SSM) via operators that combine problem descriptions (disease hypotheses and findings) (Fig. 22). The essence of differential diagnosis is constructing sets of competing diagnoses, in which each set accounts for the symptoms and contains one or more disease descriptions (allowing for multiple faults). Thus, each differential diagnosis is an SSM. Diagnosis consists of searching the space of alternative SSMs. Thus, Pople emphasizes that given a large set of disease classes which can be intricately interrelated (both causally and in heterarchies), there are potentially dozens if not hundreds of SSMs consistent with a given set of findings. Diagnostic search must be conceived at this level, not just in terms of deciding what operator to apply to a given SSM. Differential diagnosis is not just a matter of deciding between a flat list of disease descriptions, but of combining and comparing subgraphs that represent relations between disease processes. Indeed, Pople emphasizes that his synthesis operators *construct* a differential diagnosis, viewing a "differential diagnosis" as an object, not merely a name for the process of asking questions.

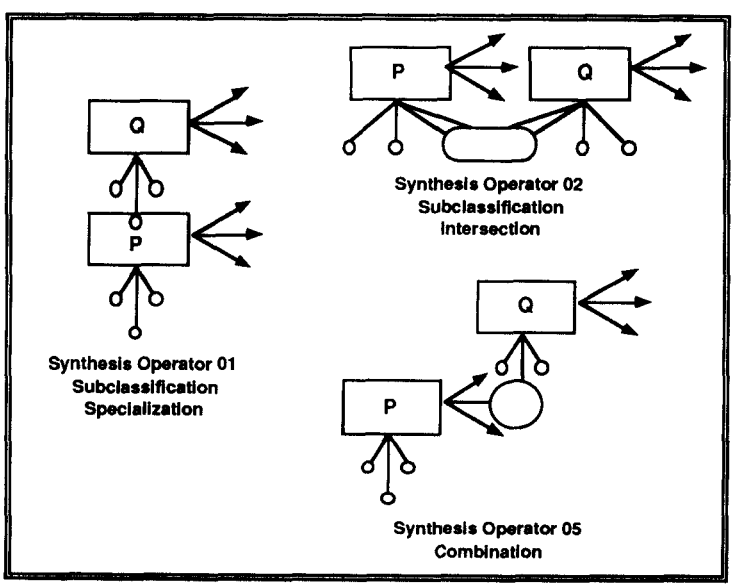

Fig. 22. Synthesis operators from CADUCEUS by which "multiple task definitions may be combined into unified task complexes" (excerpted from [75]). (Arrows indicate causality, vertical lines indicate subtype.)

The examples in Fig. 22 illustrate how two disease hypotheses P and Q can be discovered to be related via subtype (operator 01), via common subclasses (operator 02), or via an intermediate disease which is caused by P and is a subtype of Q (operator 05). Similar operators include causal specialization, intersection, and more complex intermediate connections. These operators transform networks of disease hypotheses from competing subgraphs into single graphs, constituting a single causal story of processes occurring in the patient (by which one disease causes another or is manifested in a particular way). Thus, *Pople's formulation makes explicit the idea of operators constructing graphs; however he does not emphasize that these graphs are models of processes.* As mentioned previously, the process-model perspective is useful because it focuses our attention on how different types of networks can be used to represent processes in different ways (e.g., the different forms a disorder taxonomy takes in NEOMYCIN and CASTER).

4.2. ABEL's model construction operators

Concurrent with the development of CADUCEUS and NEOMYCIN, Patil was formalizing diagnosis in ABEL in terms of operators for constructing a patient-specific model (Fig. 23).

The operator "projection" is what Pople calls "causal specialization". However, the other operators are different because they operate on different levels of detail. Aggregation and elaboration relate a composite, abstract term (a disorder name) to a subgraph that details the causal and component relationships of this process. Summation and decomposition relate quantities and

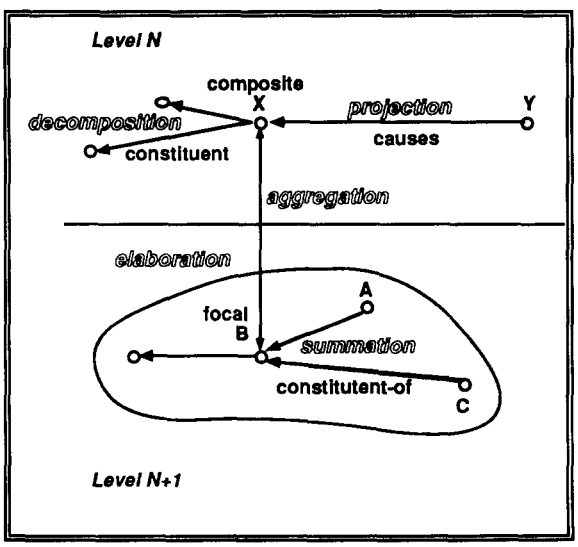

Fig. 23. "The PSM is created by instantiating portions of ABEL's general medical knowledge and filling in details from the specific case being considered" [74].

sources of substances within a process description at a particular level of detail (e.g., A and C might correspond to water loss and sodium loss, both constituents of Lower-GI-fluid-losses, node B). With this representation, ABEL has the capability of reasoning about interactions of processes. For example, when one process produces a substance that another consumes, ABEL can hypothesize the resultant quantity of the substance. The advantage of ABEL over CADUCEUS and NEOMYCIN is therefore its representation of the SSM at different levels of detail, by which it can construct descriptions of processes that are not pre-enumerated in the knowledge base. CADUCEUS can account for processes that co-occur, but process interactions are all preclassified via the causal and subtype specialization relations in the knowledge base. ABEL is not just dealing with disorder names (corresponding to the P and Q nodes in Fig. 22), rather it *constructs subgraphs that it treats as new process descriptions and relates these to each other* (corresponding to the line around the nodes at level $N + 1$ of Fig. 23). However, it is also true that ABEL's ability to aggregate and elaborate process descriptions is limited by the general model, as well as its ability to find higher-level relations among the summations and aggregations it has made.

4.3. ACCORD's model construction operators

Following the development of ABEL, CADUCEUS, and NEOMYCIN, Hayes-Roth reformulated the representation of control knowledge in BB1 in order to make explicit the task-specific operators and blackboard structure used for configuration problems [47]. The resultant framework is called ACCORD; it nicely builds on the work of Sowa [84] to make explicit the domain relations used by each operator.[4] Operators are viewed as verbs, with the subject and object relations corresponding to types of domain terms. For example, the operator YOKE links two subgraphs in the SSM, corresponding to a spatial constraint between partial configurations of the system being designed (Fig. 24). Operators are abstracted, so control knowledge can be written in terms of types of operators. For example, control rules can reason about when and where to attempt a POSITION operation.

The operator language in ACCORD is a state-of-the-art representation, with the advantage of reifying inference operators as objects which have properties, are abstracted into hierarchies, and can be independently applied. The careful choice of names for these operators is also a major contribution to the community's goal of accumulating libraries of operator definitions and control knowledge. Terms like "yoke", "dock", and "anchor" are general and could be usefully applied to temporal as well as spatial aspects of an SSM. For example, these operators could be respecialized for combining portions of a

[4] BB1-ACCORD is thus analogous to HERACLES-DX. BB1-ACCORD is a task-specific specialization of BB1, and HERACLES-DX is a specialization of HERACLES.

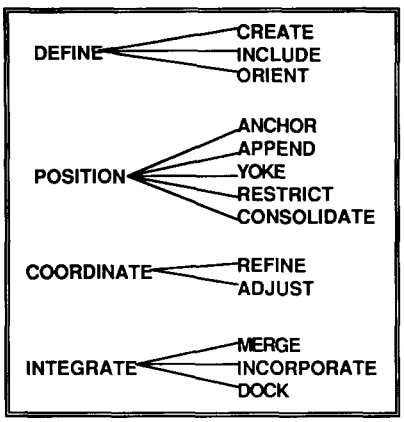

Fig. 24. Operators in ACCORD, abstracted according to the effect each has on the evolving SSM (from Hayes-Roth et al. [47]).

process description in diagnosis (e.g., "position" might be interpreted in terms of a place on a time line; "append" would show two temporal sequences to be contiguous). Besides the clarity of its design, *ACCORD reveals the generality of the idea that an SSM is a model of a system and provides a systematic vocabulary for building a library of operators*. The limitation of ACCORD's design, common to every system we will consider in this paper, is that no dimensional analysis is offered to argue that this set of operators is in any sense complete.

4.4. HASP's blackboard reconsidered

Part of the difficulty of synthesizing past research is that it is not just a matter of identifying one previous conception as being most correct or useful, and then subsuming everyone's work under that. The blackboard model of control provides a major foundation for retelling the story of expert systems, however an adequate synthesis requires viewing blackboards themselves in a new way. For example, it should be clear that ABEL's PSM is a blackboard: It is a network that the program uses to post a solution, and it uses the partial state of this solution to decide what data to gather and assertions to make next. However, it should be equally clear that the idea of multiple levels is useful, but not an essential characteristic of a blackboard, as illustrated by NEOMYCIN and CADUCEUS. Furthermore, the medical examples suggest that we *view the blackboard as a representation of some system being modeled, not just a data structure*. Finally, the idea that the data structure is "common" or shared crosses the line from a description of how a model is constructed to a programming language view.

Figure 25 provides a starting point for relating blackboard research to a more general conception of qualitative process modeling. The system being modeled

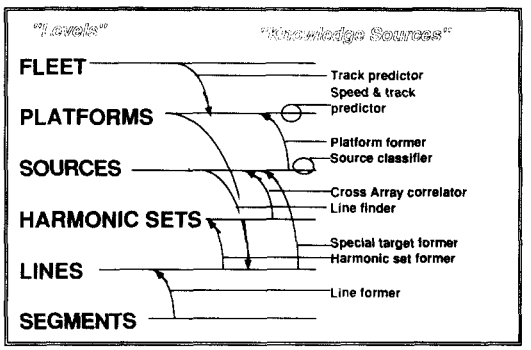

Fig. 25. HASP's blackboard (from Nii [73]).

in HASP consists of ships and submarines in the ocean. The task is to identify what system is present, the particular combination of platforms (vessels) and their combination into fleet(s). The system being modeled is not a static structure, but a set of loosely interacting parts with temporal characteristics (the vessels are usually always moving).

Figure 25 shows how very low-level behavioral characteristics of the system (chiefly sounds emitted by propulsion subsystems) are abstracted to identify source types and ultimately related to specific vessels. Thus, HASP's problem is essentially a design problem, given that the system being identified usually has novel configuration and process characteristics (to be contrasted with NEOMYCIN's selection of complete system descriptions such as "Acute-bacterial-E.coli-meningitis" from the disease taxonomy).

HASP's operators are called "knowledge sources". As is evident from their names, these operators can be abstracted according to the type of change they make in the SSM. "Formers" aggregate descriptions (e.g., a line former abstracts segments in the sound data to line patterns). "Predictors" work in the opposite direction to elaborate a description in terms of substance or process details that should be occurring in the specific system being modeled and thus form expectations for hypotheses or findings that should appear in this SSM. In short, these are not merely "knowledge sources" modifying "hypothesis elements" in a blackboard. *They are operators that correlate, form, predict, find, etc. process representations in order to construct a space–time model of a system.*

With this system-model-operator perspective in mind, we now realize that important characteristics of the model have been left implicit in its blackboard implementation. Most importantly, what are the spatial, temporal, causal, and subtype relations between the levels? If these are made explicit, it will be easier for the program to be modified and enhance its explanation ability. Crucial for the objective of developing task-specific frameworks, we want to know how the blackboard is structured so we can model similar systems using

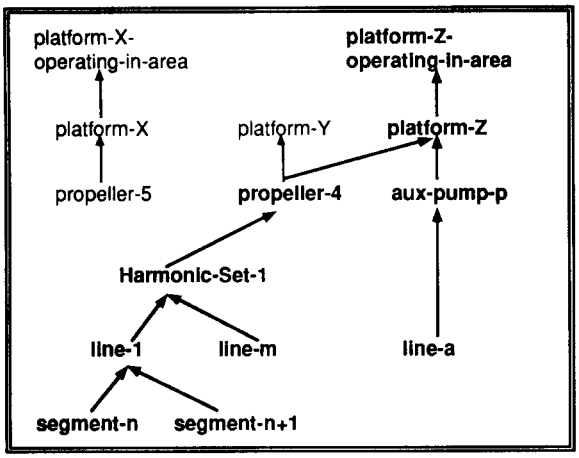

Fig. 26. "Partial lofargram analytic model" (from Keeler, 1987).

the same design. This design is relatively clear in ABEL and is a major contribution of ACCORD; in both systems it is the subtype, causal, temporal, and component relations between blackboard elements and not the levels per se that drive the inference process.

After reviewing the SSM description of NEOMYCIN [26], Keeler[5] redescribed HASP in terms of operators constructing a relational network (Fig. 26). Adopting the language used to describe NEOMYCIN's SSM, Keeler says:

> The process of lofargram analysis is the construction of a proof tree relating the acoustic spectrum and sources that could have generated these signals.... The target has a three-bladed prop (propeller-4); what platform could have that component? An auxiliary pump, type p, is unique to platform-Z types, and we direct our attention from above in searching for a line-a.

This example shows the generality of the SSM-operator perspective, as well as how the blackboard model of control can be reformulated to make explicit the model-graph-operator characteristics of inference.

In summary, *we need to move blackboard terminology from the programming-language level to the process-modeling level.* Words like panel, knowledge source, events, rules, focus of attention, and scheduler emphasize how a model and inference operators are encoded in a programming framework. Missing are mathematical concepts—such as relation, operator, graph, link, edge direction—and system-modeling operations—such as explain, locate, sequence. Nodes on the blackboard can be described in terms of an ontology of substances, processes, states, structures, and events (in the process being

[5] Larry Keeler, Personal communication, NTSC, Orlando, FL (March 1987).

modeled). Finally, a global perspective of the blackboard as a model of a system with certain space–time characteristics is essential for arguing about the adequacy of a solution and for describing the space of operators represented by a given set of "knowledge sources". Relating the structure of the blackboard as a model to inference operators provides a way of systematically designing or looking for knowledge sources.

4.5. MYCIN's context tree reconsidered

One of the defining characteristics of EMYCIN's design is the *context tree* [12], a means of organizing dynamic information during a consultation (Fig. 27).[6] The tree's design, dating from 1972, exhibits many features we associate with object-oriented programming today:

- a hierarchical class structure, with distinction between classes and instances;
- dynamic generation of an instance hierarchy, with provision for nonhierarchical associations[7] (e.g., infection/organism, organism/recommendation, culture/current therapy);

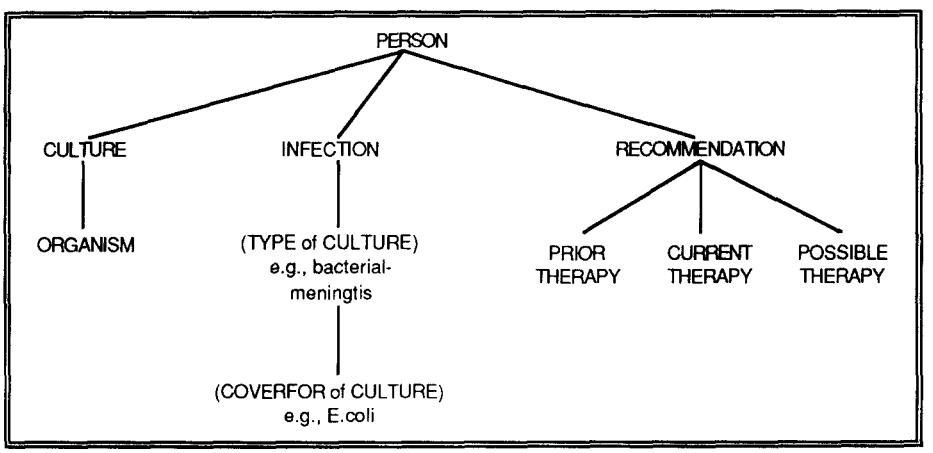

Fig. 27. MYCIN's context tree (circa 1977), with culture "parameters" shown as infection subtypes.

[6] In the original program, TYPE and COVERFOR are parameters of a CULTURE context. The values of TYPE are infection categories (e.g., bacterial meningitis). The values of COVERFOR are subcategories of these (e.g., E.coli bacterial meningitis). As I have said, the community of computer scientists and physicians who designed MYCIN did not talk in terms of disease classifications. The representation of the disease classification is distributed across several object types and parameters: INFECTION, "name of infection"; CULTURE, "type of infection", and CULTURE, "organism that therapy should cover for". To make explicit these relations, I have shown the parameters TYPE and COVERFOR as contexts below INFECTION.

[7] The value of an attribute can be an instance name or list of instance names. For example, "current therapies affecting a culture" is a parameter of a culture context and has current therapy instances as values.

- inheritance of class and instance properties;
- methods for filling in slots of instances (EMYCIN's parameters and rules for concluding about them);
- rule primitives for collecting, filtering, sorting sets of instances according to arbitrary predicates (e.g., a rule might refer to "the positive cultures from sterile sites with gram-negative rod organisms").

In effect, the context tree provides a place for posting information about the current case. The tree structure shows how objects are related (e.g., the organisms observed on a particular culture are grouped as children of that culture). Examining Fig. 27, we observe that there are three kinds of relations: *spatial* (organism located in a culture), *subtype* (kind of infection), and *componential* (drug therapy part of a recommendation). This systematic relational structure reveals that the three parts of the context tree are modeling three systems:

(1) The *physical structure and history of the person*, including cultures and organisms from different parts of the body.
(2) *Pathophysiological processes* (in the meninges and blood), that is, the disease process causing observed symptoms and positive cultures.
(3) The *therapy plan* (a list of drugs, dosages, and methods of administration).

It is now apparent that MYCIN's context tree corresponds to three situation-specific models; that is, it is a blackboard with three panels (Fig. 28). Each panel has different relations between the levels (as we observed in HASP, Fig. 25). The instances or objects—cultures, organisms, disease processes, drugs—are copied over from the general model, which describes what instances are possible (e.g., names of all sites where cultures can be taken) and how they can

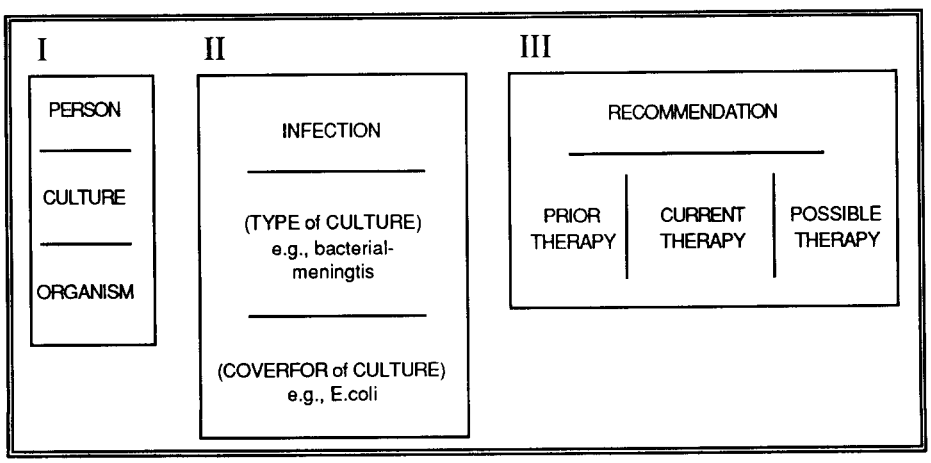

Fig. 28. MYCIN's context tree shown as a blackboard with three panels.

be related (e.g., what drug combinations are allowed and what organisms they are effective against). As discussed in Section 2, some of this information, such as the disease hierarchy, is implicit in rules.

Throughout, I have emphasized that how we visualize program structures influences how we talk about them (e.g., the three views of inference). This lesson is particularly clear when we replace the hierarchical lines of the context tree by boxes—suddenly, MYCIN has a blackboard! A number of observations leap out:

(1) The three panels loosely correspond to three stages in the consultation. The PERSON/INFECTION link is accomplished by heuristic classification. The INFECTION/RECOMMENDATION link is accomplished by generate and test from a grammatical description of valid drug combinations [23].

(2) The PERSON-CULTURE-ORGANISM panel models the system being diagnosed as a whole, setting up its physical and historical structure. This pattern is seen also in SACON (in terms of a building and reasons for failure analysis) and TOPO, an experiment in using HERACLES for configuration of computer networks (organizational structure and physical layout of working groups). Examination of other programs suggests that the first step in design or diagnosis modeling of a system (e.g., a person) is to *model the physical structure of the system* (its parts and their properties) *and the historical environment of the system* (in the case of diagnosis, its creation and where it has been; in the case of design, constraint specifications for assembly and future use).

(3) Therapies are related temporally, shown here as three sections at one level in the third panel. Temporal reasoning is very important in MYCIN, in order to determine how culture results are affected by current and prior therapies (i.e., drugs may mask the effect of an infection, obscuring the symptoms, without curing the patient). This reasoning mainly concerns the overlap of events (e.g., did a symptom occur more than three days after therapy began?).

(4) Broadly speaking, the operators (implicit in rules, EMYCIN's interpreter, and the therapy program) describe the patient's physical/historical structure (the task "specify", panel I), develop a top-down description of the disease process (the task "diagnose", panel II), and construct a therapy recommendation (the task "configure", panel III). In effect, NEOMYCIN's subtasks and metarules (Section 6) make explicit and generalize the operators in panels I and II.

It is no surprise that it took so long to realize that the context tree corresponds to a blackboard. First, the idea of a blackboard was developed years after MYCIN was designed; the context tree picture was firmly entrenched. Second, general descriptions of "blackboard systems" appropriately

emphasize opportunistic reasoning, which does not occur in MYCIN. If the idea of a blackboard as *a place for posting a model* of a system—an overarching theme of this paper—had been emphasized over the use of the blackboard for driving the reasoning process, researchers might have more readily seen that the context tree is a blackboard. In this respect, it is interesting that both EMYCIN and HASP fail to make explicit the relations between objects. This lack of attention to structure (the representation of a model) and overemphasis on programming (EMYCIN's rule interpreter, HASP's scheduler) played a major role in obscuring the commonalities of the designs. Ironically, a third obscuring factor is that the *reasoning about spatial, temporal, and component relations in MYCIN is more complex than the simple "levels" idea in blackboards*. That is, the blackboard conception alone does not have the complexity required for reasoning about the relations among cultures, organisms, infections, therapy recommendations, and drugs.

As stated at the beginning of this section, synthesizing research is not a simple matter of mapping onto one existing conception. In identifying the blackboard idea as a good orientation, we have moved back and forth between showing what it clarifies and what it fails to bring out. Our study of NEOMYCIN, HASP, and MYCIN in particular shows that if researchers simply dropped their own representations and adopted the blackboard conception (of say, BB1), important distinctions would be lost. Specifically, a synthesis shows that the relations among objects, levels, and panels need to be made explicit in order to make explicit how processes/systems are modeled. Furthermore, recurrent structures emerge (i.e., the first panel is similar in several programs), which we will want to build into knowledge acquisition tools and convey to knowledge engineers as guidelines for designing blackboards. These patterns are discussed further in Section 8, when we consider knowledge acquisition tools. But first we consider what kinds of processes are modeled in programs in general, and then the patterns in NEOMYCIN's subtasks and metarules.

5. AI programming as process modeling

Having established the value of the blackboard and operator perspective, we now consider more broadly how blackboards can be used in an AI program. This brings out the different kinds of processes that a program might model and what flexibility the blackboard perspective offers over other architectures.

Not surprisingly, we generally describe a new kind of computer program in terms of a programming language. Thus, papers about MYCIN emphasized the production rule language and interpreter. Papers about blackboard systems emphasized the relation between knowledge sources and schedulers. Generally, the news to report at first is how to get certain performance out of a

program or how to code something so it can be easily modified. In this respect, object-oriented programming has emerged from AI research as an important contribution to computer science, even to the extent that it is sometimes identified with intelligent programs in the popular literature. How is the blackboard conception related to object-oriented programming?

As shown by the examples in this paper, programming-language descriptions (e.g., HASP's "blackboard events") are inadequate for describing expert systems because they combine process-model characteristics with how the model is encoded and interpreted as a running program. Programming-language descriptions have in fact so disguised the expert system enterprise that the underlying model structure and construction operations are even denied by researchers in the misleading "shallow" versus "model-based" distinction. Nevertheless, programming-language descriptions can be useful for comparing alternative views of inference, making explicit the contribution of the blackboard perspective. This analysis is particularly valuable for describing the methods of AI programming to other computer scientists.

5.1. An object-oriented view of inference operators

Consider the following sequence of programming languages, more or less corresponding to the evolution of techniques for encoding inference procedures: procedural code in a conventional program, inference rules as in MYCIN, an object-oriented domain model as in CENTAUR and MDX, an object-oriented method hierarchy as in NEOMYCIN, methods viewed as operators for constructing an SSM graph as in ABEL, and finally the use of a scheduler for controlling methods as in BB1-ACCORD.

5.1.1. Procedural code

A traditional computer program combines a general model with an inference procedure, coded as conditional statements. Runtime ordering of statements must be explicitly coded. Typically, the system being modeled is implicit; attributes are represented as variables.

Representation in C code:
Boolean FEVER, INFECTION;
if (FEVER) then INFECTION = TRUE;
If there is a fever, then there might be an infection.

5.1.2. Inference rules

In a typical rule-based system, attributes or relations are distinguished from objects, which are signified by variables (e.g., CNTXT in MYCIN). Attributes are represented by literals. Facts about the system being modeled are treated as global data; they are structured objects (e.g., propositions), not just values of variables. Conditional statements are called rules, which can be controlled in

a data-directed way, not just linearly executed like program steps. Rules are generally written in a language with a restricted syntax amenable to interpretation by different programs (translation, explanation, knowledge acquisition, student modeling, tutoring). Thus, conditional statements are treated like data.

Representation in EMYCIN:

```
(PATIENT Mary)

(IF (AND (PATIENT $CNTXT) (FEVER $CNTXT))
THEN (INFECTION $CNTXT))
```

If the patient has a fever, then the patient has an infection.

Uncertainty is represented by second-order relations, e.g., the following is a formal representation of EMYCIN's rule predicate, MIGHTBE:

```
(IF (AND (BELIEF ($RELATION $CNTXT) $CF)
         (GREATERP $CF -200))
THEN (MIGHTBE $RELATION $CNTXT).
```

5.1.3. Object-oriented domain model

In a typical object-oriented system, such as CENTAUR or MDX, domain objects are represented in a hierarchy and inference methods are attached to them. This approach makes explicit the classification nature of the domain knowledge, but often domain relations that could be stated in process-model language are stated procedurally. For example, in OCEAN (developed by Cimflex-Teknowledge, Inc. as a reformulation of R1 for NCR) structural-component relations in a computer system being configured are expressed by the "FROM" and "TO" relations, indicating the way in which the model is interpreted to refine the SSM. Also, methods are often stated using domain terms, rather than abstracted, as illustrated by an "if-confirmed" method from CENTAUR (using the same infection example).

Representation in CENTAUR:

```
(MORE-SPECIFIC infection (disease meningitis))

(IF-CONFIRMED infection
  (DETERMINE disease of infection))
```

5.1.4. Object-oriented method hierarchy

In NEOMYCIN there is both a domain object hierarchy (the disease taxonomy) and a method hierarchy (the subtasks/metarules). The methods do not reference domain terms directly; they are abstract. Although this discipline could be applied by moving all methods to the most abstract objects in the domain hierarchy, as is done in MDX, representing methods in a separate object hierarchy makes clear that they are not just associated with abstract domain objects (such as "disorder" or "finding"), but they are themselves

hierarchically abstracted. Figure 4 illustrates how the infection example is handled in NEOMYCIN; see Section 8 for further examples of method abstraction in generic systems.

5.1.5. Methods as SSM-graph manipulation operators

In HERACLES-DX, the generalization of NEOMYCIN into a diagnostic shell, the object hierarchy is viewed as a process classification. The method hierarchy is viewed as operators for constructing an SSM. Specifically, the state of the program's reasoning is reified as an object in its own right, the SSM, with dynamic properties that drive the application of methods (e.g., "an unexplained abnormal finding exists in the SSM"). This perspective is in most respects adopted in ABEL, CADUCEUS, and ACCORD. ACCORD's use of abstraction for describing methods is illustrated by Fig. 24. The important shift is from talking about object-oriented programming in general to talking about a particular kind of computer program in which the objects represent processes (and indeed, this applies to both the domain and method object hierarchies).

5.1.6. Agenda/blackboard and scheduler for posting and selecting methods

In BB1/ACCORD, methods are not executed directly, but an intermediate reasoning step involves posting the methods that could be applied on an agenda. In other variations, such as Hayes-Roth's model of planning [46] and Lesser's HEARSAY variants [58], goals and methods are posted on a control blackboard. That is, the inference process itself is represented on an inference SSM, paralleling the posting and reasoning about alternative processes in the domain system being modeled (sometimes called the "data blackboard").

In this sequence we have compared diverse programs like MDX, EMYCIN, and BB1/ACCORD, using the idea of object-oriented programming as a backbone for understanding the shift from directly executing programming statements to representing them as objects with properties. In parallel, what is a simple conditional statement in a FORTRAN or C program becomes two or more statements: the causal, subtype, temporal or spatial relation of the domain model and the operator that places this relation in the SSM.

Crucially, the program's output is viewed not just as values for variables, but a structured representation of processes occurring in some system being modeled. This representation takes the form of a list of diseases, called the differential in the early NEOMYCIN. It then becomes a graph, relating substances, states, and processes, called the SSM. A similar transition occurs in moving from a linear agenda, which simply orders operators that can be applied (a kind of differential), to a control blackboard in which alternative inference processes are described, elaborated, and contrasted.

5.2. Advantages of a control blackboard

We now understand that the advantage of BB1's architecture over HERACLES is that it allows control of inference operators to be represented in a more general way, directly analogous to the advantage of NEOMYCIN's subtask/metarule representation over rule-clause ordering in MYCIN's rules. In fact, representing alternative inference processes is precisely what ODYSSEUS does (and must do) when modeling a student's diagnostic strategies. Alternative strategic abstractions of a sequence of patient data requests are posted as alternative lines of reasoning, that is different explanations for why the student requested the information in this order [93]. The same idea is exploited in Murray's [69] BB1-tutor in which alternative instructional plans are posted on a blackboard. Figure 29 summarizes how the idea of an SSM can thus be generalized for representing processes in the world (the system model and plans for assembling, controlling, or modifying this system), processes in the program (the inference process), and processes of interaction between the program and the world (e.g., discourse or instructional plans).

The innermost two ellipses in Fig. 29 correspond to the domain or general

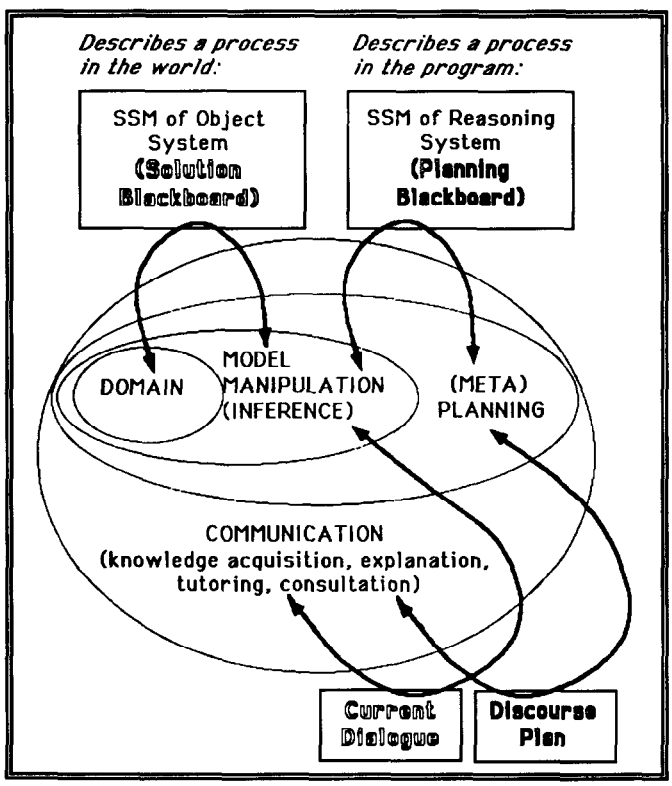

Fig. 29. General view of AI programs showing nested representations of domain, inference,

model (the knowledge base) and the inference procedure, which we have been discussing in this paper.[8] The SSM we have focussed on is shown at the top left. The next layer corresponds to the control or planning blackboard and scheduler in programs like BB1. The reasoning process is represented by combinations of operators on the planning blackboard, which may be elaborated into alternative lines of reasoning. In essence, this is where ODYSSEUS posts alternative strategic models of a student's reasoning. Stefik's MOLGEN [86] provides another good example, in which there are explicit planning operators (called metaplanning operators because the object SSM is an experimental laboratory plan). Stefik's inference planning operators include GUESS-UNDO and LEAST-COMMITMENT. Of course, these are model-manipulation operators, too. They manipulate the model or representation of the inference process, in contrast with inference operators, which manipulate the representation of processes in the system being modeled.

The outermost layer concerns processes by which the program interacts with its environment. For example, in GUIDON and BB1-tutor these are instructional discourse processes. Again, there are two layers. The first, here labeled "current dialogue", is a representation of the interaction as it has been occurring (e.g., the student has just asked for help). The second SSM, here labeled "discourse plan", is another planning blackboard; here the program posts alternative dialogue interactions it might have with the student (e.g., generate a hint, provide a simpler example). Objectifying the discourse process in this way facilitates interruption and return to a previous state of the dialogue, as well as lookahead (e.g., to determine whether an example can be suitably elaborated and related to the current situation or to estimate duration of an interaction and take resource limitations into account). A similar pair of blackboards could be used for acquiring data from sensors, allowing interrupts from multiple sources to be posted and selectively attended to.

In both GUIDON and NEOMYCIN, which lack planning blackboards, these considerations have to be coded in an ad hoc way. For example, after GUIDON selects a candidate domain rule to present, it must look ahead to determine if the rule can be adequately discussed (perhaps its syntax is too complicated). In general, a planning blackboard is useful for determining whether operators can successfully apply to a candidate focus operand in the object SSM. An analogous problem occurs when an abstract operator selects a line of reasoning; a control blackboard provides a systematic way of posting the selection criteria so the lower-level operators can refer to it when instantiating the plan. For example, GUIDON might select a domain rule for presentation because the student forgot to apply it in a previous case. However, the discourse procedure for discussing a domain rule is separate from the proce-

[8] Figure 29 is not a Venn diagram. Outer processes reason about and/or control inner processes, e.g., the inference procedure reasons about the domain model.

dure for selecting what rule to discuss, and thus the selection criteria is not available for biasing the presentation method, except through ad hoc use of flags and other bookkeeping records. Put another way, levels in a blackboard represent support for planning decisions and make selection criteria available for subsequent processing—an important theoretical principle in controlling reasoning revealed by our analysis.[9]

Similarly, NEOMYCIN's inherited end conditions lead it to interrupt a line of reasoning (e.g., pop up from TEST-HYPOTHESIS back to ESTABLISH-HYPOTHESIS-SPACE to reconsider what hypothesis to focus on). However, it retains no explicit description of the line of reasoning it interrupts (just the SSM and a record of the contexts in which subtasks have been applied). In general, a planning blackboard enables posting which subtasks have been started, but are incomplete, enabling a program to deliberately compare alternatives and pick up where it left off, rather than beginning entirely new lines of inquiry (which could confuse the student/user).

In summary, the representation of inference and discourse in NEOMYCIN and GUIDON respectively is abstract (using no domain terms) and represented via hierarchical operators, enabling these operators to be represented non-redundantly and flexibly applied. However, without a control blackboard for posting the current, interrupted, and alternative lines of reasoning (operator sequences), *communication between operators* concerning the state of the inference process (for constructing a domain SSM and constructing a dialogue SSM) must be handled in an ad hoc way. The use of a network representation for lines of reasoning in ODYSSEUS and a blackboard for posting alternatives, in contrast with IMAGE in which lines of reasoning are represented as a flat agenda, enables the ODYSSEUS student modeling program to contrast different lines of reasoning and argue for completeness of its explanations (e.g., a given strategic model encompasses the full sequence of student data requests). A similar conclusion concerning explanation dialogue planning is reached by Moore and Swartout [67].

5.3. The generality of process modeling

Figure 29 captures a recurrent result in diagnosis, explanation, and cognitive modeling research. We need to treat the diagnosis, the dialogue, and the inference process as explicit structures. In all cases, the SSM makes explicit the model construction nature of the inference process, provides a place for posting and reasoning about alternative models, and by an analysis of the form of the model allows arguments about its incompleteness or inadequacy for the purpose at hand to drive the inference process.

[9] It is worth mentioning again that I use the term "blackboard" because it makes explicit the idea of an SSM. I am not promoting wholesale acceptance of the programming style of a "blackboard architecture", which focuses on symbol-level manipulations, illustrated by the analysis of HASP (Section 4.4).

In each case, *a process is treated as a structured object which can be reasoned about*, rather than being something that is executed or interpreted directly as code (e.g., the contrast between metarules that post subtasks on a control blackboard versus metarules whose actions directly apply subtasks). Posting and reasoning about processes enables parallel execution, suspension/resumption, and dynamic ordering (contrasted with selection of predefined action sequences). This idea is very general and has important implications for computer science in general: *AI programming methods enable representations of processes to be constructed and manipulated by different programs (themselves represented as structured processes)*. Indeed, it should be evident that the nesting of domain model, inference process, planning, and communication is very general; it could be argued that this covers the gamut of what programs must know and do.

It is worth recalling that we developed this picture by beginning in NEOMYCIN with the idea that *different processes should be represented separately*. From this method, we have demonstrated a range of programs with new capabilities: modeling a student's diagnostic strategies, generating strategic hints, detecting that a solution is incomplete, generating hypotheses about missing domain facts. Figure 29 and the process-modeling perspective also provide a good starting point for teaching about AI programming. For it should be clear that we have stepped beyond expert systems per se to encompass the full sweep of AI research concerns, including planning, natural language generation, and learning. The value of this perspective is obviously not in resolving all open questions, but rather in providing a single architecture for integrating work in different areas.

We have also learned that what makes software reusable by different interpreting inference procedures is the ability to classify process constructs in different ways. For example, domain relations in NEOMYCIN's metarules are classified so the compiler can replace them by LISP code. Other properties of domain relations are used by the explanation program (e.g., if one relation implies another, it may be sufficient to state just the more specific clause). The subtasks are classified in one way for GUIDON-MANAGE's hint generator and classified another way for the explanation and student modeling programs (e.g., GUIDON-MANAGE does not suggest subtasks below the level of subtasks in the student's menu). IMAGE must know how metarules can be reordered or omitted when generating advice from a student model. At a lower level, the arguments of subtasks (called foci) are also classified (into finding, hypothesis, and domain rule), so the explanation program can determine whether a type change has occurred in a line of reasoning (arbitrary variable names would not allow this). In short, *different interpretation procedures require different views of the model/process being reasoned about*. These views take the form of new relations by which elements of the model/process are classified, hence new structure. These relations appear in conditions for

applying operators for constructing an SSM using the model (e.g., the compiler's SSM is the LISP representation of the inference procedure). In essence, these *relations act as filters* by which different elements of the process model are preferentially collected and sorted for incorporation in the SSM (cf. the description of inference in SOAR).

We have also discovered a recurrent set of operations that are performed on processes: specification, design, assembly, diagnosis, etc. In general, the relations among these operations shown in Fig. 21, which was presented as a description of how a domain model is created and used, applies equally to the inference, planning, and communication processes. For example, the inference planning process is a design task; the compiler is an assembly task; the knowledge acquisition program includes both diagnosis and repair tasks. Given that the representations of all these processes are similar (using compositions of hierarchical and state-transition networks), the possibility arises that there are common inference procedures. That is, if the relational network representation is similar (see Section 7), the diagnostic operators for constructing a patient-specific model should share some similarities with the operators for diagnosing what has gone wrong in an instructional dialogue. Indeed, everything would collapse to the inner two ellipses of model and model-manipulation procedure, as shown in Fig. 18, with different representations and inference procedures for different types of systems (e.g., CNS versus LISP program) and different tasks.

This analysis suggests that we should now study the relation between types of systems, process model structures, SSM structures, and inference procedures. In Section 6 we present a dimensional analysis of NEOMYCIN's subtasks; in Section 7 we study the types of relational networks used for representing processes. These ideas are then applied in Section 8 to improve our descriptions of knowledge acquisition tools.

6. Formal analysis of subtasks and metarules

Subtasks are essentially functions; here we adopt three different perspectives for describing them:

(1) as operators placing oriented edges in the SSM graph;
(2) as operators manipulating typed sets (the SSM-graph nodes—findings, hypotheses, and domain rules);
(3) as operators satisfying constraints on the form of an adequate SSM.

In these three perspectives we view subtasks according to how they change the SSM, viewed at the different levels of edge, node, and subgraph. The purpose of this section is to show that an inference procedure can be systematically described using simple graph, set, and logic distinctions. This description provides a convenient means of determining the completeness of a

given inference procedure, as well as a common language for comparing and collecting libraries of inference operators.

6.1. *Placing oriented edges in the SSM*

Figure 16 suggests that at least some subtasks can be described according to the types and directions of the links they place in the SSM. Table 1 describes many of NEOMYCIN's subtasks in this way. For example, FINDOUT and CLARIFY both place an edge between findings, moving downwards from a finding to a more specific finding (e.g., FINDOUT grows a link from medications to aspirin; CLARIFY grows a link from headache to the headache's duration). The key on the far left indicates that this is a supporting link, an attempt to find evidence for or details about a more general process.

We also see that FINDOUT can grow links in the opposite direction, from a known specific finding to an unknown general finding (e.g., if the patient has undergone neurosurgery, we can conclude that the patient has undergone surgery, and hence go on to ask about cardiacsurgery). Whether the focus findings are known or not is a secondary consideration; there are systematic patterns in the metarules, so they are noted here.

Notice that the subtasks are described in this table by several dimensions, in terms of:

Table 1
NEOMYCIN's subtasks viewed as operators that place oriented edges in the SSM.

SUBTASK	FINDOUT CLARIFY (gen'l -> unknown specific)	x	TEST CONSIDER H (new H -> known F)	GROUP
(support) From node I ▼ To node	*(specialize)* F I ▼ F	F I ▼ H	H I ▼ F	*(generalize)* H I ▼ H
RELATION	subsumes	causes	causes	subsumes
To node ▲ I From node *(explain)*	F ▲ I F *(generalize)*	F ▲ I H	H ▲ I F	H ▲ I H *(specialize)*
SUBTASK	FINDOUT (known specific -> unk. general)	x	CONSIDER F (new F -> any H)	REFINE

- the type of the subtask focus (e.g., hypothesis, indicated by H, is the focus type for TEST-HYPOTHESIS),
- the type of node to which the subtask seeks to grow a link (e.g., TEST-HYPOTHESIS seeks to link its focus hypothesis to a finding node),
- the relation linking these nodes (e.g., "causes" for TEST-HYPOTHESIS), and
- the direction in which the graph is being modified (e.g., TEST-HYPOTHESIS grows a "support link", from a hypothesis node to a finding node lower in the graph).

Further distinctions can be made regarding whether the nodes represent known findings (or previously considered hypotheses). Findings are never linked directly above hypotheses (column 2) because by definition, a finding is an observed system behavior that is caused by or subsumed by an internal state, structural change, or process (which are called hypotheses because in diagnosis we are attempting to infer the description of the system). For findings to be placed above hypotheses, the direction of causality or subtype would have to be reversed, and there would not be a consistent interpretation of the SSM in terms of support (that a process is occurring from below) and explanation (for particular behaviors from above). This underscores the point that an SSM is not an arbitrary graph, but is a representation of processes, a model that is interpreted for the purpose of the tasks at hand, such as diagnosis and repair.

This table only shows operators for extending a given graph by adding one link between findings and hypotheses. Other operators incorporate laboratory tests (a kind of finding), relate hypotheses in different subgraphs (such as DIFFERENTIATE), or simply place a node in the SSM (e.g., an unexplained new finding).

It should be apparent that this kind of table provides *a dimensional analysis* that merely listing operators (as in Fig. 22 (CADUCEUS) and Fig. 24 (ACCORD)) does not provide. The graphic presentation of ABEL's operators (Fig. 23) shows the kinds of links the operators grow and their direction, but there is no way of seeing quickly what the possible operators are.

The subtasks listed in Table 1 are *primitive operators* because they directly place links in the SSM. More abstract operators control the order in which these primitive operators are applied. They are called *procedural operators* (Fig. 30). The metarules for such subtasks constitute *ordered, conditional steps* in a procedure. In contrast, the metarules for primitive subtasks are *alternative methods* for accomplishing one thing—placing a certain kind of link in the SSM graph. A typical procedural operator is PURSUE-HYPOTHESIS. Its two metarules invoke TEST-HYPOTHESIS and REFINE-HYPOTHESIS in sequence. Procedural operators represent preference between primitive operators. For example, to accomplish a top-down, breadth-first search of the

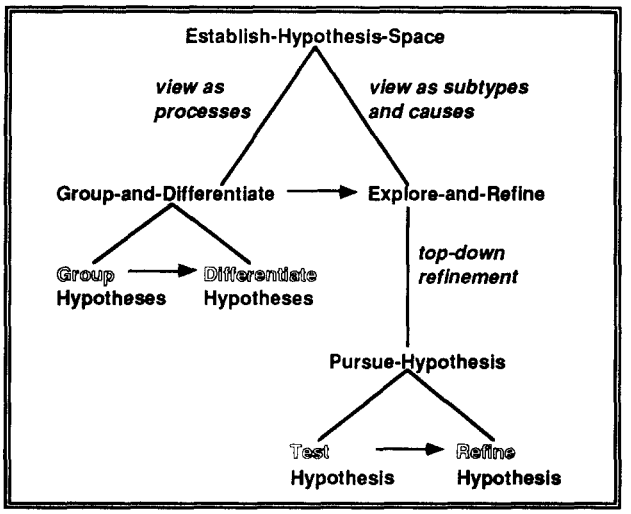

Fig. 30. Procedural subtasks order the application of primitive operators (GROUP, DIFFERENTIATE, TEST, REFINE) or other procedural subtasks GROUP-AND-DIFFERENTIATE, EXPLORE-AND-REFINE). Arrows indicate preference relations represented by the order of metarules for a given subtask.

disease taxonomy in NEOMYCIN, we want to test a hypothesis before refining it. If we refined hypotheses first, we would test the most specific disease first and lose the benefit of pruning the search at a higher level in the disease taxonomy.

GROUP and DIFFERENTIATE are not implemented as explicit subtasks in NEOMYCIN; our analysis (Table 1 and Fig. 30) shows that these are two distinct operators. Thus, one benefit of this analysis is making clear how subtask decomposition can be done in a principled way. Specifically, the dimensional analysis shows how to coherently group metarules into subtasks that accomplish one kind of operation (by edge type and whether they are primitive). It was a surprise to realize that the GROUP-AND-DIFFERENTIATE metarules fell into two disjoint groups, which constituted two distinct subtasks that might have been explicitly labeled, analogous to TEST and REFINE. Metarules for other subtasks, such as FORWARD-REASON and PROCESS-HARD-DATA, not shown here, can be abstracted and regrouped in a similar way.

Figure 30 also shows that procedural subtasks themselves can be characterized in terms of modeling processes. In particular, we see that "looking up" in GROUP-AND-DIFFERENTIATE entails a categorical view of the model in terms of processes; "looking down" corresponds to Szolovits' probabilistic phase, viewing the model in terms of specific causal and subtype relations.

In practice, these preference relations are enforced by the use of HERACLES' ability to interrupt a subtask and return control to a higher-level procedural subtask. Preconditions and goals, called end conditions, can be

associated with each subtask; they are in effect inherited as interrupt conditions by all subtasks on the stack. A true end condition signifies that a subtask precondition is violated or a subtask goal is achieved; the subtask interpreter aborts the current subtask and returns control to the highest subtask on the stack for which no interrupt condition holds. In particular, EXPLORE-AND-REFINE has an end condition termed "wider differential", which means that a new hypothesis has been added to the SSM which is not subsumed by any existing subgraph. This requires return to categorical reasoning (GROUP-AND-DIFFERENTIATE). This simple mechanism served well in NEOMYCIN, but as indicated previously such end conditions themselves need to be objects that can be posted and reasoned about for student modeling and discourse management (which forced the use of a control blackboard/agenda in both ODYSSEUS and GUIDON-MANAGE).

6.2. Manipulating sets of SSM nodes

From a programming-language viewpoint, it is interesting to describe the subtasks as set-mapping functions. It is surprising at first to realize that there is just a small group of possible set operations that are performed by the 84 NEOMYCIN metarules. This analysis has value for explanation programs (summarizing a line of reasoning in terms of subtask focus changes) and certainly for knowledge acquisition (providing templates for modifying and acquiring new metarules).

In particular, we find that subtasks have either a single node as a focus (argument) or a set of nodes of a single type. For example, Fig. 31 shows that CLARIFY-FINDING maps between a single finding and a list of findings. Here we are not viewing the subtask in terms of the kinds of links it places in the SSM, but computationally in terms of function (subtask) arguments. In effect, CLARIFY-FINDING collects a set of findings and invokes FINDOUT on each of them. Similarly, FORWARD-REASON sorts its arguments, mapping from a set of input findings to ordered invocations of PROCESS-FINDING. DIFFERENTIATE maps from a set of hypotheses (the roots of disjoint SSM graphs) to a discriminating finding. *In effect, each subtask is focusing the operation of the subtasks it invokes by collecting, ordering, and filtering arguments that are passed on.* In each case, we find a relation in the premise of a metarule that takes the focus as a given term and seeks possible bindings from the general domain model in order to carry out these collection, ordering, and filtering manipulations (Fig. 20).

As we have seen (Figs. 4, 6, and 8), there is a striking pattern by which each new metarule (not just each new subtask) tends to require a new relation by which nodes in the SSM can be collected, sorted, or filtered. This is an important result for the design of knowledge acquisition and learning programs: It means that changes to the theory (the relational language in which the domain model is expressed) are tied to new metarules that use that

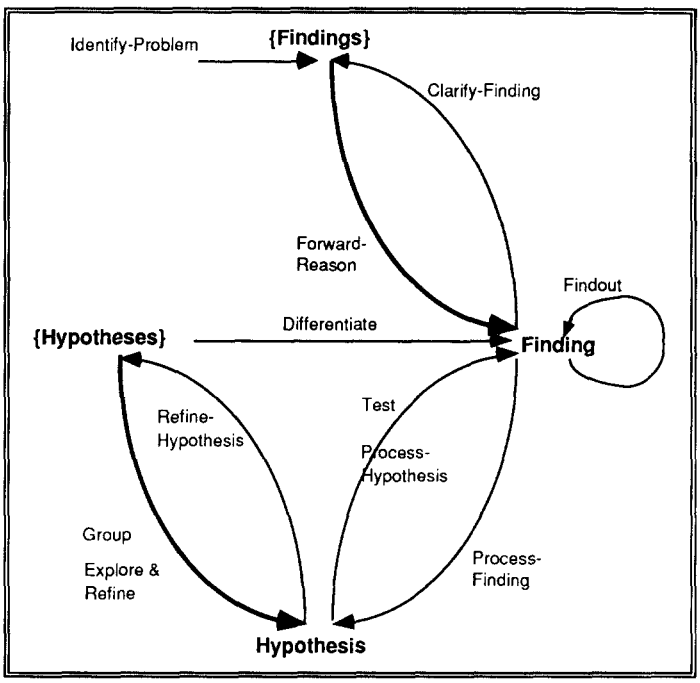

Fig. 31. Subtasks viewed as functions manipulating types of SSM nodes (boldface arrows indicate a selection or filtering process).

distinction in order to selectively apply an existing operator to a set of SSM nodes. We can expect the types of nodes, links, procedural operators, and even most of the primitive operators to be relatively stable. Acquiring a new kind of knowledge (a new relation)—at least in our experience in the development of NEOMYCIN—generally involves acquiring a new metarule for an existing operator. More commonly, we acquire only new domain propositions, which themselves can change the program's behavior; the strategy is unchanged because the relational language and metarules remain the same.

An example of set-mapping analysis applied to the initial specification of an expert system, very much in the spirit of the approach advocated here, is provided by Alexander et al. [2]; for example, for a meeting-planning program, the SELECT-ARBITRATOR operator is defined as

$$\text{Meeting} \times \text{Purpose} \rightarrow \text{Arbitrator},$$

a notation with the advantage of representing the intermediate relations (i.e., Purpose) that will qualify the operator's focus (Meeting) in order to select an output variable (Arbitrator). In this notation, we might write:

TEST-HYPOTHESIS:
Diagnostic-hypothesis × Causal-effects
→ Conjectured-finding.

Ontological analysis provides a formal specification for subtasks and metarules; the subtask/metarule language in turn provides a stylized procedural implementation for Alexander et al.'s knowledge-level specifications. In related work, Hartley [43] shows how Sowa's conceptual structures, using a case-frame representation of "actors", can be used to represent diagnostic operators that manipulate sets of symptoms, tests, etc.

Another direct application of set-mapping analysis is for collapsing a line-of-reasoning explanation. By examining subtasks in terms of set-mapping relations, the explanation program can detect potentially confusing focus shifts. For example, in Figs. 9–11 we observe a complicated shift from data-directed to hypothesis-directed reasoning. Rather than traipsing through the networks link-by-link, the explanation program could say, "Besides meningitis, papilledema might be caused by an intracranial mass lesion. Before jumping to that conclusion, I need to gather additional evidence that there is increased intracranial pressure." The possibility of composing TEST-HYPOTHESIS-(Meningitis) with TEST-HYPOTHESIS(Increased-Intracranial-Pressure) in the first sentence lies in the analysis of the focus mapping:

$$\mathrm{TH}(\mathrm{H1}\rightarrow\mathrm{F1}) \rightarrow \mathrm{PF}(\mathrm{F1}\rightarrow\mathrm{H2}) \rightarrow \mathrm{TH}(\mathrm{H2}\rightarrow\mathrm{F2})\,.$$

Thus, explanation patterns need not be specified in terms of particular subtask sequences, but in terms of sequences of foci (H1 to F1 to H2 to F2). The number of such combinations that lead to clearer explanations have not been fully explored in NEOMYCIN; surely the example shown here will occur in many programs.

The set-mapping view of Fig. 31 has the advantage of showing how the diagnostic work of NEOMYCIN gets done, in terms of mapping from findings to hypotheses. The figure provides a simple dimensional analysis as well; for example, we discover that no subtask maps from sets to sets. Unfortunately, this figure is a white lie; it is an abstraction that leaves out the domain rules. For example, TEST-HYPOTHESIS does not literally invoke a subtask with a finding argument. It invokes APPLYRULES with a list of domain rules as an argument. Can a literal description of the subtasks provide additional interesting information?

Table 2 shows literal relationships betwen subtask foci. A given subtask accepts the first column (indicated by o--) as a focus and calls a subtask with a focus indicated by the destination column (indicated by →).[10] Possibilities that are logically already covered by preceding cells are indicated by ×. Subtasks that map from a list of rules only invoke APPLYRULE!, so this category is omitted. Mappings to a list of rules ({R}) are rare, but other missing associations are commented in italics.

[10] In the column headings, F corresponds to finding, H to hypothesis, R to rule and sets of these, which are the domain and range of the subtasks listed on the left. The table is designed to enumerate every possible domain–range combination.

Table 2
Literal relationships between subtask foci and metarule actions.

SUBTASK	{F}	F	{H}	H	{R}	R	{F}	F	{H}	H	{R}	R
Forward-Reason Generate-Questions	o	→						✘	✘	✘	✘	✘
(indirectly via rules)	o	-	→					✘	✘	✘	✘	✘
??? analog of Differentiate	o	-	-	→				✘	✘	✘	✘	✘
???	o	-	-	-	→			✘	✘	✘	✘	✘
Generate-Questions	o	-	-	-	-	→		✘	✘	✘	✘	✘
??? analog of Est-Hyp-Space	o	-	-	-	-	-	→	✘	✘	✘		
(indirectly via rules)		o	→						✘	✘	✘	✘
(indirectly via rules)		o	-	→					✘	✘	✘	✘
Process-Finding Apply-Evidence-Rules		o	-	-	→				✘	✘	✘	✘
Process-Finding		o	-	-	-	→			✘	✘	✘	✘
Elaborate-Datum		o	-	-	-	-	→		✘	✘	✘	✘
Findout Clarify-Finding Elaborate-Datum Find-Relevant-Tests Findout-Table Get-Test-Associate-Results Process-Finding		o	-	-	-	-	-	→	✘	✘	✘	✘
Explore-&-Refine Group Forward-Reason Process-Hard-Data			o	→						✘	✘	✘
???			o	-	→					✘	✘	✘
??? like Generate-Questions			o	-	-	→				✘	✘	✘
(indirectly via rules)			o	-	-	-	→			✘	✘	✘
Differentiate			o	-	-	-	-	→		✘	✘	✘
Establish-Hypothesis-Space			o	-	-	-	-	-	→	✘	✘	✘
???				o	→						✘	✘
Test-Hypothesis Process-Hypothesis				o	-	→					✘	✘
??? expectations?				o	-	-	→				✘	✘
Process-Hypothesis Ask-for-Hard-Data				o	-	-	-	→			✘	✘
Refine-Node Refine-Complex-Node				o	-	-	-	-	→		✘	✘
Process-Hypothesis Pursue-Hypothesis				o	-	-	-	-	-	→	✘	✘
Applyrule!						o	→					
???						o	-	→				
Applyrule!						o	-	-	→			
???						o	-	-	-	→		
???						o	-	-	-	-	→	
???						o	-	-	-	-	-	→

Several "missing" associations are accomplished indirectly (e.g., FORWARD-REASON accomplishes {F}→{H} indirectly, via the composition of PROCESS-FINDING [{F}→R] and APPLYRULE! [R→{H}]). Other missing associations seem logically possible but have not been accounted for in NEOMYCIN's metarules. For example, we could add a metarule to GENERATE-QUESTIONS (a set of heuristics for improving an SSM which is too incomplete for the program to determine what to do next) by adding a

mapping from {F}→H similar to the DIFFERENTIATE metarule. *Detecting such gaps is precisely the value of a dimensional analysis.* This analysis also provides another perspective for detecting composite subtasks (e.g., ELABORATE-DATUM has metarules with different destination arguments (both {F} and F). Finally, this analysis provides a mathematical basis for relating operators from different programs. For example, what is {F}→F called in MDX? In CADUCEUS? Does ABEL have an operator corresponding to {H}→F? Consider how filling in such a table could focus a workshop discussion.

Finally, it may seem odd that rules are mentioned here at all. Why not describe the subtasks in terms of findings and hypotheses alone? The reason is that this is a literal description of the argument types of the subtasks. We are describing the computation each subtask actually carries out, not what it logically (or cumulatively) accomplishes through its subtask invocations. This is important because we want to map the subtasks, as computer programs, onto the SSM graph. We want to know precisely where the code is for each operator. Furthermore, in so far as rules are links in the general domain model, subtasks with rule arguments are referring to the general model and its reflection in the SSM. From this we realize that there are operators that examine and determine the logical applicability of the general model (APPLYRULE!), and not just operators that examine and change the SSM directly.

6.3. Satisfying constraints on SSM form and purpose

In order to argue about the completeness of a given set of subtasks and metarules, we must go beyond consideration of structural changes to the SSM. We must consider the purpose of the SSM and redescribe the subtasks functionally in terms of how they satisfy the program's overall goals.

Part of this analysis was provided in Section 3; viewing the SSM as a model of some system, we can list the constraints on the form of this model for it to be adequate for the task at hand. Specifically, for infectious-disease diagnosis and therapy we want to minimize the number of drugs, so we attempt to consolidate the diagnosis in terms of a single-disease process (single-fault). We can refer to therapeutic actions available to determine whether diagnoses need to be more specific in order to select a single action.

Other constraints follow from the form of the general model. Supporting a disease hypothesis requires supporting its ancestors in the disease taxonomy, which are essentially frames that collect properties common to each subtype (e.g., every disease process under infectious process causes a fever, however they may specify the type of fever or conditions that mask or prevent its occurrence). It is the structure of the disease taxonomy that leads to the particular group, discriminate, explore, refine, and test metarules in

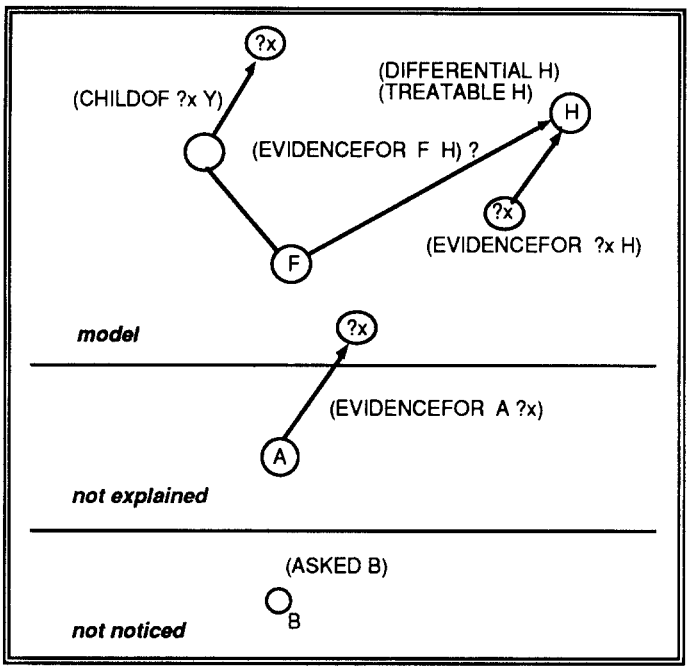

Fig. 32. Domain model relations corresponding to SSM nodes and links. Findings can be in the model, known but not explained, or available but not noticed by the program.

NEOMYCIN. Figure 32 summarizes how these constraints are related to the nodes and links of the SSM.

Three types of nodes are shown:

- findings/hypotheses that are part of SSM subgraphs (the "model");
- known abnormal findings that have not been explained (i.e., not linked to any process subgraph); and
- findings in the case record that have not been noticed by the program.

Only the link between hypothesis Y and finding F is established here; the arrows and (ASKED B) indicate unsatisfied constraints. For example, if it can be established that F is evidence for H, all the hypotheses will appear in one graph, which is desirable. Thus, the propositions shown here are not the constraints themselves (e.g., to have one graph), but links found to be missing when the constraints are applied to a given SSM. For example, the constraint that all abnormal case data should be discovered by the program might be written,

$$(\forall F\ (\text{AND}\ (\text{KNOWN}\ F)\ (\text{ABNORMAL}\ F)) \Rightarrow (\text{ASKED}\ F))\,.$$

(ASKED B) represents a particular SSM node that is missing. The figure also

shows two propositions that must be satisfied by any "final diagnosis": it must be a subgraph root of the SSM (DIFFERENTIAL H) and it must be treatable.

In the program GUIDON-DEBUG, these constraints are applied to the final SSM to generate a list of missing nodes and links. Heuristics order and minimize the number of hypothesized nodes and links. In particular, the hypothesis node that is the root of the subgraph containing the largest number of abnormal finding nodes is chosen to anchor the analysis. If we call that node H_{root}, then (EVIDENCEFOR F_i H_{root}) links are only generated for known F_i that are not in this subgraph. (EVIDENCEFOR ?x H) propositions are also generated for any leading hypothesis H that is not directly supported by a finding it causes. Table 3 summarizes these constraints and the corresponding SSM propositions they detect to be missing. The constraints are expressed in terms of gaps or inconsistencies, which in order to be satisfied specify bindings for propositions that must also be satisfied.

Probably the most surprising result is that GUIDON-DEBUG discovers findings in the case record (shown as B in the figures) that were not vol-

Table 3
Violated SSM constraints and corresponding unsatisfied propositions.

SSM CONSTRAINT VIOLATION	SSM PROPOSITION UNSATISFIED
(AND (KNOWN B) (ABNORMAL B) (NOT (REQUESTED B)) *a known abnormal finding B was not requested*	(ASKED B)
(AND (DIFFERENTIAL Y) (NOT (TREATABLE Y)) *a subgraph root Y (one of the leading hypotheses) is not a treatable diagnosis*	(CHILDOF ?x Y)
(AND (STRONGLY-BELIEVED H) (NOT (CONSIDERED H)) *a hypothesis H with certainty greater than 400 was never put in the SSM*	(DIFFERENTIAL H)
∀$F (KNOWN $F) => (AND (DIFFERENTIAL H) (NOT (CAUSED-BY $F H)) *a subgraph root H is not directly supported by any known finding*	(EVIDENCEFOR ?x H)
(AND (BEST-HYP H) (KNOWN F) (ABNORMAL F) (NOT (EXPLAINED F H)) *the best hypothesis H does not explain a known finding F (special case: F is explained uniquely by hypothesis H1, but not H)*	(EVIDENCEFOR F H)
∀$H (DIFFERENTIAL $H) => (AND (ABNORMAL A) (PRESENT A) (NOT (EXPLAINED A $H)) *a known abnormal finding A is not explained by any subgraph root H*	(EVIDENCEFOR A ?x)

unteered or requested during the consultation. Without this constraint-satisfaction perspective, we would not have thought to systematically analyze NEOMYCIN's diagnoses in this way. Indeed, by our previous perspective, the name of the diagnosis at the top of the SSM was all that was printed by NEOMYCIN; if this was correct from the expert's perspective, we said that the program had the right answer and moved on to the next case.

The propositions shown in Fig. 32 and Table 3 can be systematically related to subtasks that place such nodes or links in the SSM. These relations are summarized in Table 4.

Working backwards from violated constraints to generate a list of unsatisfied propositions (Table 3), GUIDON-DEBUG invokes the explanation program to determine why the corresponding subtask(s) did not succeed or why it was not applied (Table 4). For example, FINDOUT(B) might have been invoked, but B might never have been asked because a FINDOUT metarule with a different action applied. If FINDOUT(B) was never invoked, then the explanation program checks metarules that have FINDOUT in their action and determines under what conditions binding B would be passed to the action and why those conditions did not apply. In general, this can be a huge search problem because we are looking for gaps in the domain model. That is, we not only want to know why a given metarule failed, but what changes to the domain model would have allowed it to succeed with the appropriate bindings. Heuristics indicating what domain relations are likely to be wrong or incomplete are especially useful here. Figure 33 illustrates an example of this analysis for a case in which NEOMYCIN failed to discover that the patient has a cranial nerve dysfunction.

GUIDON-DEBUG finds that Metarule611 never invoked FINDOUT/CRANERVE because the subtask FIND-RELEVANT-TESTS was never invoked with the focus CRANERVE ($FINDING). The question then becomes,

Table 4
SSM propositions and subtasks that satisfy them.

SSM PROPOSITION UNSATISFIED	SUBTASK(FOCUS) FAILED
(ASKED B)	FINDOUT(B)
(CHILDOF ?x Y)	REFINE-NODE(Y)
(DIFFERENTIAL H)	PROCESS-HYPOTHESIS(H)
(EVIDENCEFOR ?x H)	TEST-HYPOTHESIS(H) PROCESS-HYPOTHESIS(H)
(EVIDENCEFOR F H)	TEST-HYPOTHESIS(H) PROCESS-FINDING(F) PROCESS-FINDING(H)
(EVIDENCEFOR A ?x)	PROCESS-FINDING(A)

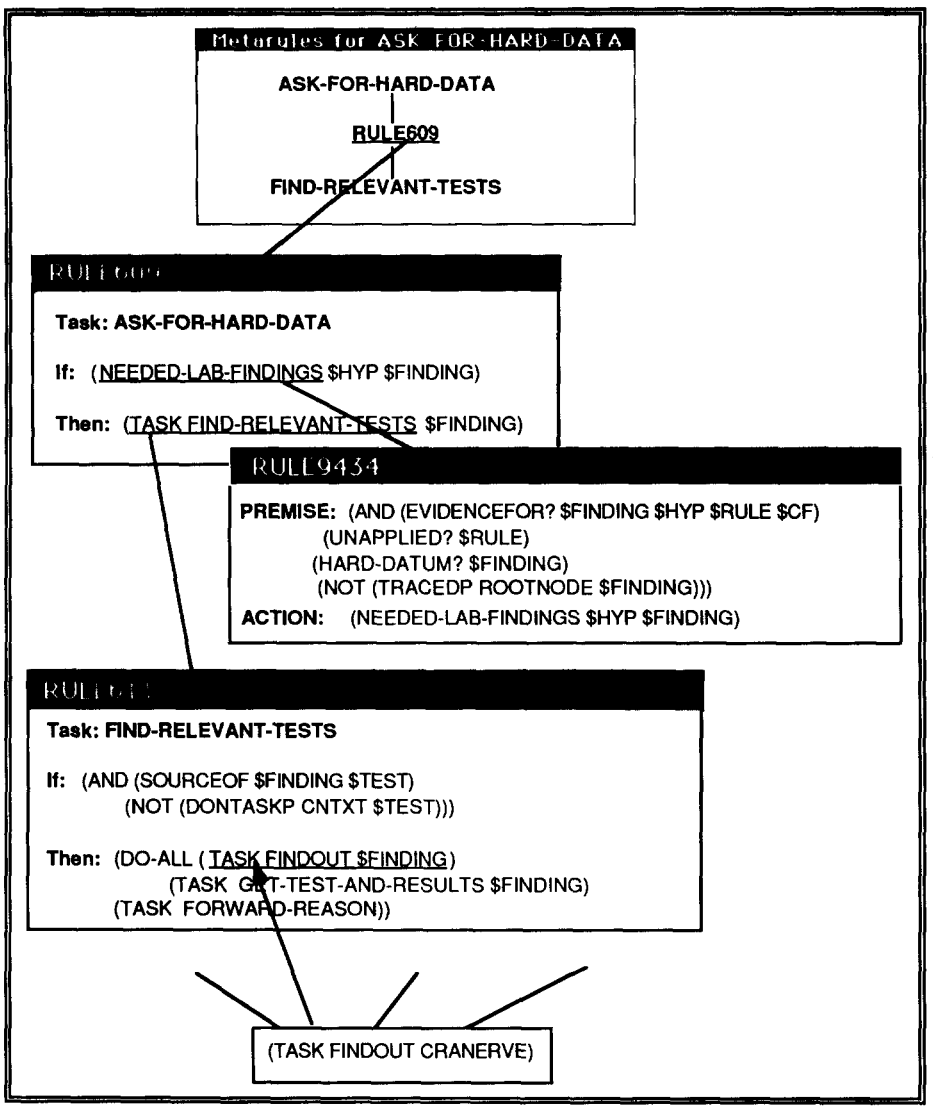

Fig. 33. GUIDON-DEBUG attempting to explain "Why wasn't CRANERVE asked during the consultation?"

"Why wasn't FIND-RELEVANT-TESTS(CRANERVE) invoked?" Looking at the history, the program finds that Metarule609 was tried with $HYP bound to INCREASED-INTRACRANIAL-PRESSURE but the first clause of definitional Rule9434 could not be satisfied; in particular, (EVIDENCEFOR? CRANERVE INCREASED-INTRACRANIAL-PRESSURE $RULE $CF) is unsatisfiable. Examining Rule9434 further, the program discovers that (HARD-DATUM? CRANERVE) will never be true (because CRANERVE is circumstantial evidence, not a direct observation of internal workings of the

CNS). Generalizing, the program says, "Metarule609 will always fail (to bind $F to CRANERVE) (independent of subtask focus) because (HARD-DATUM? CRANERVE) cannot be satisfied." There are several possibilities: CRANERVE is hard-data, but this is not represented correctly in the domain model. Or perhaps Metarule609/Rule9434 should be modified. After noting these, GUIDON-DEBUG goes on to find other reasons why FINDOUT/CRANERVE was not invoked.

Observe that this method of generating explanations is much more general than the approach used in MYCIN, where every type of question requires a specialized function (e.g., "Why not conclude X?", "Why ask F?"). In GUIDON-DEBUG all inquiries are mapped onto one question and its complement: "Why (not) do subtask X?" Possible subtasks include ask, conclude, hypothesize, test, refine, etc. In effect, the design of MYCIN's inference engine was redundantly represented in the explanation system. Because all backward-chaining in NEOMYCIN is deliberately controlled, all questions can be tracked back to metarule actions that apply rules and ask for case information.[11] Furthermore, explanations are in terms of inference subtasks (e.g., inquiring about a test which is the source of hard data relevant to a hypothesis), not just in terms of specific domain rules. However, we found in our investigation of strategic explanations that the specific domain rule, stripped of all its procedural and screening clauses required in MYCIN, often provides a cogent explanation (see the domain-specific hints in Fig. 12).

In summary, GUIDON-DEBUG detects problem solving failures and tracks them back to gaps in domain knowledge. The approach is similar to that used in apprenticeship and explanation-based learning [51, 66, 83]. However, there are some basic differences:

- *Goal regression involves directly interpreting the inference procedure itself*, rather than a representation redundantly encoded within the learning program.
- *The operationality criterion is described in terms of the form of an adequate solution*—constraints a good diagnostic model should satisfy—rather than in terms of computational efficiency. Thus, the SSM provides a coherent way of specifying what are sometimes called "operationality criteria", means for determining the usefulness of new concepts. (Much research in explanation-based learning focuses on deriving a relation that is implicit in the domain model; the goal of learning is to make the program able to solve a problem that was previously too time consuming.)

[11] The interpretation of domain and metarules is still implicit in code. In particular the way in which APPLYRULE! leads to invoking FINDOUT and CONCLUDE is implicit because this is done by the EMYCIN interpreter when domain rules are applied. But any task that leads to applying rules goes through APPLYRULE!, so this logic need only be coded once in the explanation program and is therefore tolerable.

- *Learning is based on explaining problem solving failures, as detected by the program itself*, not in explaining why a supplied example is correct. That is, the method involves determining what needs to be learned in order to more adequately solve problems, not just to increase problem solving speed. Thus, this method bridges a gap between explanation-based learning and cognitive models of failure-driven learning [80].
- *The problem solving procedure uses a schema model of the world* (the domain relations) which constitutes an incomplete theory, in contrast with the axiomatic theories of domains like calculus.

Finally, Fig. 32 and Tables 3 and 4 suggest that we might generate the subtasks and metarules from a list of constraints they must satisfy. For many subtasks, the relation is obvious. For example, every metarule for TEST-HYPOTHESIS has a proposition of the form

(EVIDENCEFOR $FINDING H)

in its premise, where H is the focus of the subtask; the only reason there are multiple metarules is to order the domain rules passed on to the APPLY-RULE! subtask. Although we have not had reason to do this, inspection of other subtasks indicates that the metarules might be generated from the domain relations and preferential constraints (e.g., test before refining) in the same manner code is generated in XPLAIN.

Our analysis shows that a syntactic description of the form of an SSM provides a very powerful means for building knowledge acquisition and explanation programs, and possibly deriving the inference procedure itself. However, it is important to realize that not all inference procedure constraints reduce to structural properties of the general model, the SSM, or the relation between the SSM and action plans. Other constraints relate to the environment in which problem solving itself will occur. For example, questions are generalized by FINDOUT in order to shorten the duration of the consultation; if data were available online this metarule might be unnecessary. Furthermore, the pruning effect of asking about surgery first, for example, relies on the fact that most patients have not undergone surgery. If most patients in the population being diagnosed had neurosurgery, we would actually save a question (about surgery) by being specific.

This is an important result: The metarules are written with respect to a particular *case population*. Probabilistic assumptions about the prevalence of findings are implicitly factored into the subsumption relations. To relate subtasks and metarules from different programs, we will need to make explicit these environmental assumptions in the domain model and refine the metarules to take environmental relations into account. Specifically, for the program to be reusable in different environments (and not just applied to different cases) we need to distinguish between processes that all systems being diagnosed

share (commonalities in the human body), environmental processes that affect the case population, and processes in the immediate problem solving environment that affect the inference process itself (e.g., urgency, availability of data). Again, the process-modeling perspective provides a useful framework for describing the generality of the knowledge base, its components, and how it might fail.

We should not underestimate the role that social considerations play in the definition of an appropriate inference procedure. In particular, it is now obvious that teaching a student about medical diagnosis involves teaching not only the disease knowledge and diagnostic process, but relating these to the environment in which problem solving occurs. In NEOMYCIN, we have viewed reasoning as essentially something that goes on in the head, omitting the social perspective that makes everything in the knowledge base reasonable and appropriate. In short, justification of knowledge should not be viewed narrowly in terms of causal relations, but also in terms of the purpose of the model and the circumstances under which it will be applied. Because social considerations are involved, there is no definitive, optimal design for the subtasks and metarules. The adequacy of a given set of metarules cannot be proved from objective properties of structure-function models or causality. The metarules embody social values and these will vary depending on the social circumstances. We can generate the primitive operators from mathematical properties of model graphs, but preference relations for applying primitive operators will depend on assumptions about the world.

6.4. Does NEOMYCIN understand what the subtasks mean?

Social and environmental considerations aside, do NEOMYCIN's metarules constitute a solution to our original problem of representing strategic knowledge? The question arises when we recall our original criticism of MYCIN's rules: clause ordering implicitly coded a top-down refinement strategy. Now we have a collection of strategy words like "explore and refine" and "differentiate". But still we use words like "look up" and "depth-first" to describe NEOMYCIN's behavior, and these words do not appear anywhere in the program. Have we actually represented the "look up before you look down" strategy? What is the difference between the metarules for EXPLORE-AND-REFINE and a definition of what the subtask name means?

To begin, notice that the definitions of procedural subtasks, such as ESTABLISH-HYPOTHESIS-SPACE (EHS), cannot be equated with compositions of primitive subtasks. Descriptions such as "top-down refinement" concern *how reasoning will appear to an observer*; they are at a different level than the primitive operations that modify the SSM. Procedural concepts describe how the program's behavior will appear *over time* as it interprets a domain model with a certain global structure, in a particular problem solving environment ("it

looks up before it looks down"). Such concepts cannot be replaced by local, moment-by-moment operations (such as TEST-HYPOTHESIS(x), REFINE-HYPOTHESIS(y)). Of course, this does not mean that we cannot control the program in a way to produce behavior that looks this way; for this is obviously what the EHS metarules do.

The concepts used for describing what a subtask like EHS does are inherently *abstractions of a sequence of program behaviors*. If you look at a sequence of data requests, a sequence of hypotheses tested, a sequence of links put in the SSM, etc., you will find a pattern that describes what repeated, regular calls to such subprocedures are accomplishing: They are looking up, they are doing top-down refinement, they are pruning unnecessary questions. Such accounts are not reducible to internal structures, states, or mechanisms. Top-down refinement is not captured by any particular set of subprocedures statically described, but is a description of the pattern of changes made to the SSM, an abstraction taking the frequency and relations between sequences of subtasks into account.

The relationship between a historical account and the inference mechanism does not appear to be intuitive. The main idea is that strategic descriptions account for *patterns in behavior* by naming and grammatically relating them. In essence, NEOMYCIN's abstract subtasks and metarules constitute a grammar. This is precisely how the subtasks are used in ODYSSEUS for explaining a student's sequence of patient data requests (each specific request can be viewed as a word; the whole problem solving session is composed of phrases that may or may not be related at the level of the most abstract subtasks).

If we do not understand this relation between an observer's description of historical trends and the mechanism of the program, we might search for formalisms that are not possible or have a misunderstanding about the nature of the formalisms we design. For example, we described GUIDON in terms of strategic abstractions such as "review frequently", and "opportunistically introduce new material". We say that such terms are "operationalized" [68] when we implement a program with such properties. However, we do not find the words "review" or "opportunistic" in the discourse procedures themselves. Concepts like "review" are temporal abstractions of sequences of program behavior. Reviewing is something we can say the program is doing at a particular time, but that is just relative to what it did before. A single procedure could be used to present a domain rule the first time it is encountered (e.g., as a quiz) and to present it again later, as a "review". Again, historical accounts characterize structural and temporal patterns in surface behavior; they need not necessarily correspond to mechanisms that invoke them.

Is this a shortcoming in our procedural languages? It is if we want ODYSSEUS to be able to reflect on a student's or expert's behavior and notice new patterns not captured by the current set of subtasks and metarules. The blocks

world provides an example of how difficult this might be. The strategy of stacking blocks from the bottom up might be encoded by the rule:

```
IF (AND (CANDIDATE-GOAL (ON X Y))
        (CANDIDATE-GOAL (ON Y Z)))
THEN (PREFER-GOAL (ON Y Z) OVER (ON X Y)).
```

Simply applying this rule multiple times will result in stacking blocks bottom up. But where are the concepts "bottom", "stack", or "bottom-up"? They need not be used in the mechanism that produces the desired behavior. In order for an agent that observes the program (perhaps the program itself) to say that "the blocks are being stacked bottom-up" (and understand what this means when we say it), the agent must *track the spatial relations between blocks and the temporal relations between stacking operations*. The words "bottom up"—just like the words "look up" or "group"—may be tied to rules like this, just as subtask names label the metarules, but they stand for concepts that are not represented.

Turned around, a program that can write the blocks stacking rule (or NEOMYCIN's procedural subtasks) must be able to represent and abstract patterns from changes in the world or SSM over time. Similarly, in order to convince yourself that NEOMYCIN's metarules for EXPLORE-AND-REFINE accomplish breadth-first search, you cannot simply look at the metarules, you must envision the effects of running the metarules over time. Specifically, writing a program that can acquire subtasks and metarules like those in NEOMYCIN—by watching a person or a program solving similar problems—requires moving from a description of how the program will appear to an observer to internal operations whose recurrent execution will produce these behavioral effects.

Indeed, this analysis leads us to realize that most automatic programming systems are not reasoning about "top-down refinement" and similar concepts in the way required for recognizing these patterns; they merely generate code from templates associated with undefined labels like TOP-DOWN-REFINEMENT or EXPLORE-AND-REFINE. We see this for example in Mostow's operationalization of heuristics for playing Hearts. "Avoid taking points" is represented by templates like

```
AVOID = (LAMBDA (E S) (ACHIEVE (NOT (DURING S E))))
```

which Mostow translates as "avoid an event throughout a scenario means try not to let it occur during the scenario". But "try not" suggests both a preference and a frequency. In recognizing this pattern, we will admit that some number of points is acceptable (though perhaps demonstrably non-optimal). After some interval, we will say that the player is not trying. Recognizing the pattern of avoiding Hearts—being able to use the concepts the way Mostow does while watching other players—requires being able to

abstract from non-optimal behavior over time, a capability that is not built into Mostow's generation routines. In short, the Heart's operationalization program does not have the same understanding of Heart's playing that a person has, just as NEOMYCIN does not have the same understanding of top-down refinement that a person has. This distinction between representations required for generation and recognition is crucial for any knowledge acquisition or student modeling system that does not receive a well-formed specification, but must abstract it from a sequence of observed behaviors.

But the problem is even more difficult than this if a program is not merely to recognize a grammatical performance, but must learn the grammar itself. Besides characterizing temporal properties in sequences of system behavior, a strategic explanation must also tease apart the interactions between the inference procedure, the domain model, and the environment. For example, suppose that NEOMYCIN is observed to ask two follow-up questions every time it receives a new finding (e.g., "What is the headache's duration?, What is its location?"). We cannot tell from the program's outward behavior whether the CLARIFY-FINDING metarules are deliberately generating two questions or whether the domain model always contains two propositions of the form (FOLLOW-UP-QUESTION F $Y), for every finding F. This example illustrates that strategic explanations are characterizing the *product* of the inference procedure's interaction with a domain model. A given behavior sequence can be abstracted in any number of domain model/inference procedure combinations. A similar example might show that the program always generalizes its inquiries when the user starts the consultation by supplying volunteered information (which perhaps the program interprets to be a sign of the user's urgency). But if the environment is always the same, we will not discover that generalizing inquiries is conditional in this way.

In summary, the meaning of subtask names constitutes a knowledge-level specification of NEOMYCIN's reasoning, derived from patterns we observed in physician behavior. The metarules "operationalize" strategies, but do not define them. Furthermore, we cannot objectively prove that an inference procedure is correct because every strategic theory is relative to the frame of reference of an observer. These conclusions require a reevaluation of how learning by being told and learning by watching are related. In particular, we must consider the shift in representations required for moment-by-moment action versus reflecting on behavior over time (abstracting behavior sequences).

To put this to immediate application, we might focus on automating failure analysis in programs like NEOMYCIN or the Heart's player, given the benefit of a representation with distinct domain model and inference procedure components. For example, to test the generality of the inference procedure, we could vary the case population, the domain model, and the problem solving environment (e.g., can avoiding points be achieved better with knowledge of

opponent strategies?). This suggests that we should attempt to *record our assumptions* about case population, model content, and environment as boundary conditions against which the program could verify the appropriateness of its inference procedure in specific cases, as well as when it is reapplied by a knowledge acquisition tool to new domains. The process-model perspective suggests that boundary testing be organized by a matrix of systems, modeling methods, and communication environments.

7. Types of process representations

In this section we are concerned with the recurrent types of networks used for modeling systems. In contrast with the usual research emphasis on the details of causal modeling or inheritance in hierarchies, we want to focus here on recurrent, *macrostructures of relational networks*. A high-level perspective reveals that there are just a few types of relational network representations for processes, such as classification hierarchies, state-transition networks, and functional composition hierarchies. This observation provides the license for defining AI programming in terms of qualitative modeling. A typology of process representations also provides a direct basis for teaching knowledge engineering techniques. This perspective has been generally ignored, partly because it is so obvious and partly because we are immersed in the details of specific representation techniques.

We have seen that inference procedures are expressed in terms of domain relations, by which nodes and links are placed in the SSM. The generality of inference procedures therefore depends on the generality of the relations we use for specifying the domain model and the SSM. Certainly subtype and causal relations are basic. What other generalizations can be made about how hierarchies and causal networks are used to represent processes in expert systems?

7.1. Types of hierarchies used for modeling processes

In NEOMYCIN, disease processes are represented by a subtype hierarchy (Fig. 14). Each node is intended to be a description of the complete system being diagnosed. For example, ACUTE-BACTERIAL-MENINGITIS and INFECTIOUS-PROCESS can both be used to describe a process occurring in the CNS; the first supplies more details about the temporal, spatial, and causal characteristics of the process (i.e., it is an infectious process that has been occurring for a relatively short time, located in the meninges of the brain, and bacterial agents are producing structural changes there). Each node is essentially an encapsulation of a historical process, something that began in the past and is still affecting the patient. That is, each process concept in the hierarchy

is essentially a causal story (e.g., bacterial infectious processes begin with bacteria entering the patient's body and moving to some location where they proliferate because normal control by white blood cells is suppressed). Figure 34 shows that NEOMYCIN's disease taxomony can be viewed as just one kind of process hierarchy, termed an *interactive-historical abnormal process hierarchy*. An abnormal process hierarchy can also model the stages in a process, as in CASTER, in which top-level nodes correspond to stages in the sand casting process. NEOMYCIN's diagnostic inference procedure is represented by a compositional hierarchy of functions, often called a *procedural net* [79]. In BUGGY [14] inference processes are represented by a network of mutually exclusive variations of composed functions; thus functional composition can be broken down further.

An additional classification could be provided for types of transitional networks (e.g., causal networks [10] and discourse state networks [27]). The distinction between classification and simulation models can also be related to the "bug library" versus "generative" distinction in student modeling programs [25].

The point of this diagram is not to show a complete classification of existing methods. Rather it reveals the kind of organization that is now possible and how this perspective is fruitful for relating representations used in AI pro-

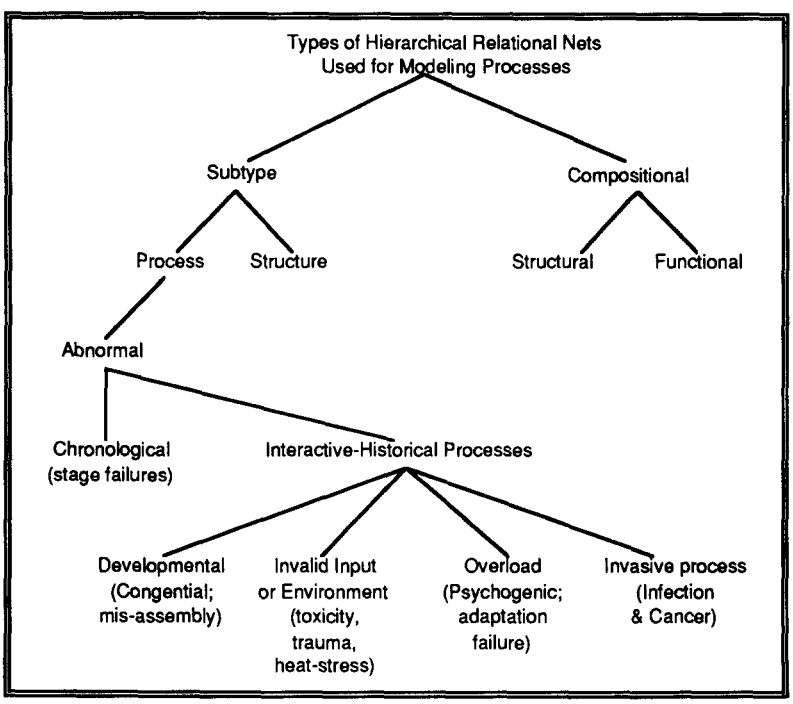

Fig. 34. Types of hierarchical networks used for representing processes.

grams. For example, the distinction between an abnormal process taxonomy organized into chronological stages of a normally-operating system (as in CASTER) and a hierarchy organized according to the kinds of historical interactions it can have with its environment (as in NEOMYCIN) is the kind of macrostructure—well-removed from implementation issues involving rules, OOP, etc.—that serves to orient a knowledge engineer early in the design phase for a new knowledge base. We can take existing programs and classify them in this way, according to the types of hierarchies in the domain model. We can build a knowledge acquisition tool around this classification, using it to index a library of expert system examples, representational templates, and inference procedures.

Indeed, this perspective provides the solution to a riddle that plagued the initial presentation of heuristic classification [24]. If NEOMYCIN is supposed to be selecting solutions from a pre-enumerated list, how can we describe its inference procedure in terms of *constructing* something? It is now clear that every diagnostic program must construct a model, in the sense of finding evidence that a process is occurring, linking findings to new hypotheses, and contrasting alternative explanations (formulated as competing subgraphs). In the simplest programs this might involve just a weighting scheme by which a list of diagnoses are matched against evidence and rank ordered. However, even this process involves constructing a list of diagnoses and ordering them—a primitive kind of SSM, but constructed from the general model and copied over into a situation-specific record nevertheless.

To understand the difference between programs like NEOMYCIN and ABEL we must look at what nodes in the SSM can represent. In a program using heuristic classification exclusively, nodes in some sense represent the entire system being modeled; they are processes in NEOMYCIN, system names in a PC advisor, etc. In ABEL, nodes stand for states, structures, or subprocesses and need to be assembled into system descriptions in order for there to be a model of a process (a story recounting how the current state of affairs began and how the manifestations were produced). In essence, ABEL is doing more than selecting, supporting, comparing, and refining off-the-shelf process descriptions; it must assemble a new process design. NEOMYCIN and CASNET use a hybrid approach, which does not involve designing new system models, termed *causal classification* [24]. Internal states and causal relations between substances are considered, but only to implicate the predefined processes in the disease hierarchy. That is, system models are selected from named descriptions (the disease hierarchy). In contrast, ABEL assembles a network of interactions on multiple levels of detail. Taken as a whole, the network ABEL constructs constitutes a new system description (albeit made out of predefined components).

This is a substantial clarification of a distinction that proves difficult without the process-modeling perspective. In particular, we need to look at what SSM

nodes stand for and how they are related in order to distinguish different modeling methods. This just underscores the obvious: In describing expert systems we are describing methods for modeling systems using relational networks.

7.2. The structure of blackboards

In our analysis of HASP, we observed that the relations between SSM nodes is not explicit. Figure 35 shows what such an analysis looks like, using NEOMYCIN's SSM as an example (refer to Fig. 15). Disease taxonomy relations appear on the left branch as subtype links between abnormal processes. Structural descriptions, such as Intracranial-mass-lesion may appear as well, with their subtypes above them (e.g., Intracranial-tumor). These abnormal structures are linked by bold arrows to the abnormal processes that they cause (e.g., Increased-intracranial-pressure). There is a simple characteristic

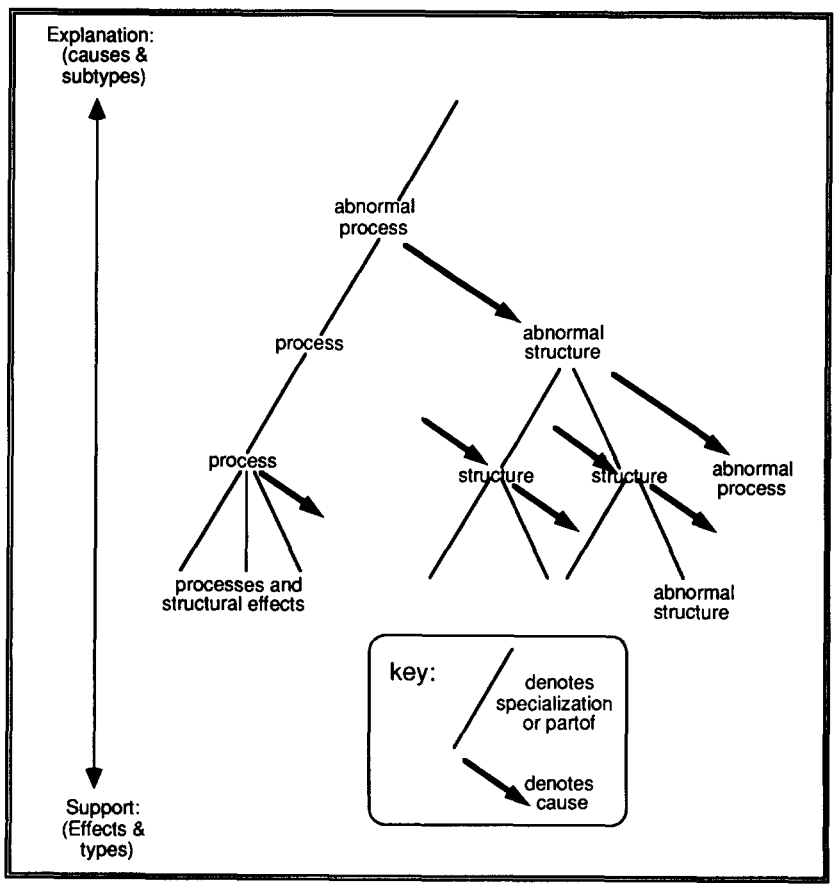

Fig. 35. Process-model relational structure of NEOMYCIN's SSM.

form (if the domain model is so organized) of abnormal structure causing abnormal process, etc. Again, it did not occur to us to describe the SSM in this way until we realized that *as a model of a system being diagnosed, it must be relating structural and process descriptions.* It is not just an "inference net" or a "dependency graph". For the program to be building up a coherent model, there must be systematic causal, temporal, and spatial relations between its inferences.

Figure 35 also makes explicit (and justifies) the pattern we claimed before, by which higher nodes in the SSM explain lower nodes: Higher nodes are the causes and subtypes of lower nodes—they specify what it is happening in more detail. In contrast, lower nodes support higher nodes—they constitute evidence for effects and substance/process categories. This is not just a "causal network". It is an oriented graph with a form that can be interpreted like a proof.

7.3. The process classification/assembly spectrum

The idea of "causal classification" used in NEOMYCIN and CASNET suggests that we should not look for black and white distinctions between process-modeling methods. Rather, it would be fruitful to align different programs on a spectrum. Figure 36 summarizes the programs we have been studying. Protean [45] and DART [37] are added as examples of programs that design new system descriptions out of primitive structure-function relations. ABEL is viewed as less general because it does not have specific representations for spatial and temporal modeling; however, in some respects its capability to reason about quantities of substances exceeds the other programs. CADUCEUS lacks the ability to summarize or decompose quantitative effects, but its ability to integrate orthogonal process hierarchies places it beyond NEOMYCIN.

Figure 36 reminds us how far this analysis has taken us from simplistic views of "shallow" versus "deep" reasoning. As stated in the introduction of this paper, expert systems are more appropriately and fruitfully described in terms of how relational networks are used for modeling processes. Hierarchies can be used to represent abnormal processes (e.g., diseases in NEOMYCIN), functional composition, etc. Network nodes represent internal states, structures, functions, or processes. Some programs have internal descriptions of entire system models, others assemble components into new system models. This analysis leads us to say that all expert systems do qualitative reasoning.

Furthermore, the same analysis used to distinguish between historical trends and mechanisms of the inference procedure (Section 6.4) shows us that abnormal process classifications cannot be replaced by or reduced to structure-function simulation models. For example, *many disease descriptions constitute the product of recurrent interactions between the system being diagnosed and its*

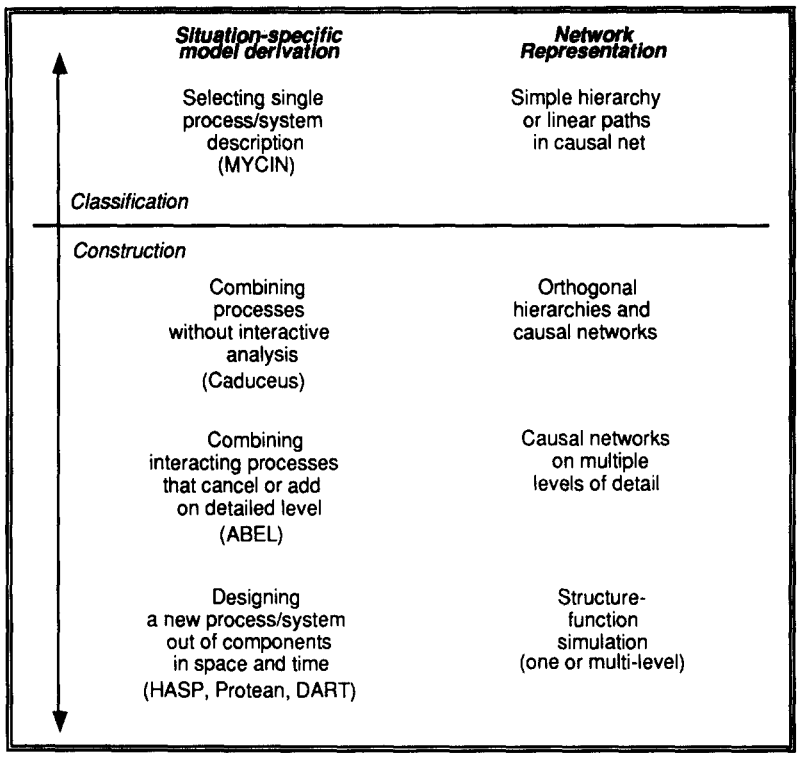

Fig. 36. Spectrum of methods for modeling processes.

environment. The structural damage a physician sees in tennis elbow could not be predicted from a blueprint of the human body; indeed, the patterns of abnormal structures and processes that will occur change with social habits and are thus inevitably open. Not only are classification models not an inferior, lazy way of modeling, they are necessary and irreplaceable (see [29] for further discussion). Disease taxonomies are not just "knowledge hierarchies" or "frames" or "concept classes". Again, until we look and study how processes are modeled and the different kinds of processes that can occur in different domains, we cannot adequately make claims about the generality or superiority of modeling methods.

In summary, the levels we have found to be useful for studying qualitative modeling are:

(1) *environment*—task constraints and assumptions:
- how the model will be used (form or level of detail required for taking action),
- case population and world data assumptions (biases),
- interactional data and communication constraints (aspects of the inference process);

(2) *qualitative calculus*—system-model representation:
- conceptual primitives (a typed-set manipulation view of classes, instances, attributes),
- relational networks (a graph construction view of hierarchies, transition networks),
- process models (a modeling view of causal, temporal, spatial, and subtype descriptions of states, substances, structures, and processes);

(3) *implementation*: programming language: (e.g., frames, rules, objects, knowledge sources, agenda, metarules).

Recalling Fig. 29, we are emphasizing once again that comparisons of modeling tools must be placed in the context of use (the environment), which determines model content and constrains the inference process.

Table 5 contrasts this with Brachman's analysis [6]. His linguistic, conceptual, epistemological, and logical levels provide one way of describing a qualitative calculus. Our system-model perspective constrains the linguistic and conceptual levels by specifying what concepts and expressions represent, including context-of-use concerns. Furthermore, we view networks of atoms and pointers—more generally, graphs—as more abstract than programming implementations in particular knowledge representation languages, and give them epistemological status. That is, useful epistemological distinctions include not only relations, but *macrostructures constructed by the systematic replication*

Table 5
Restatement of system-model perspective in terms of Brachman's [6] "levels of semantic networks".

Level	Brachman [6]	System-model distinctions
Linguistic	Arbitrary concepts, words, expressions.	A model of some system for some purpose, in some environment.
Conceptual	Semantic or conceptual relations (cases), primitive objects and actions.	Specifically, causal, spatial, temporal, etc. relations between structures, states, processes, etc.
Epistemological	Concept types, conceptual subpieces, inheritance and structuring relations.	Same, except includes types of relational networks.
Logical	Propositions, predicates, logical operators.	Same, except includes set and graph operators
Implementational	Networks of atoms and pointers.	More specifically, the programming languages that people use (e.g., EMYCIN).

of relations (e.g., hierarchies and transition networks). These macrostructures are manifested at the conceptual level as recurring conceptual networks (e.g., types of abnormal process hierarchies). We view the logical level as including set and graph operators, following Sowa's approach of integrating propositional, set, and graph notations. Finally, we describe the implementation level in terms of programming languages (specifically relegating "rules" versus "frames" arguments to this level).

8. Task-specific knowledge acquisition tools

The system-modeling perspective provides a good basis for developing knowledge acquisition tools. To illustrate this, we will consider examples from some leading programs, showing how these programs can be generalized and more easily related to each other if we express strategic knowledge abstractly. Restating the reported methods should not be viewed as criticism, so much as building on previous work, in the same way that NEOMYCIN would not have existed without TEIRESIAS. Furthermore, researchers emphasize different aspects of systems modeling, exemplified by the trivial treatment of causal representations in NEOMYCIN.

We will show how domain principles in XPLAIN [87] contain implicit system-modeling inference rules. Analysis of KNACK/WRINGER [52, 53] will show how object instantiation is handled by abstract control rules (and how much of this was handled by EMYCIN's inference engine). Finally, we will relate Generic Tasks [15–17] and role-limiting methods [63] to the terminology used in this paper, and examine the problem of defining a useful level of abstraction.

8.1. XPLAIN: abstracting domain principles

Like NEOMYCIN, XPLAIN was developed to improve the explanation program of a medical program. The work complements NEOMYCIN by emphasizing how procedures can be written by an automatic-programming system from a domain model and set of prototype methods. Figure 37 gives a well-known example from the program.

The prototype method does not fit the definition of abstract control knowledge used in NEOMYCIN because it mentions the term "drug dose". "Finding" might refer to any domain from car diagnosis to legal reasoning. However, the domain suggested by "drug dose" is much narrower. While the system being treated is not specified (it could be a sick house plant), practically speaking this prototype method will only be useful for medical diagnosis. This example illustrates that there is no sharp line between "domain-specific" and "abstract"; it is just a judgement that we make based on the systems we know

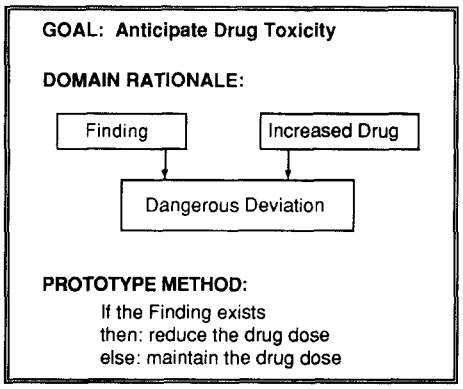

Fig. 37. Example of a domain principle from XPLAIN.

about. The final part of this section will consider principles for selecting a useful level of abstraction.

A second observation is that the domain rationale does not express the process-model relations between its terms. Figure 38 illustrates one way of formulating this domain principle in a NEOMYCIN-like metarule. Reinstantiating this rule in terms of our example, drug dosage is a quantitatively-variable action that causes an undesirable effect (a dangerous deviation) in the system being treated, proportional to the amount of the drug. In addition, another cause ($FINDING) of this effect is currently present. Therefore, to avoid increasing this undesirable effect further, the amount of the drug should be reduced.

Obviously, there are many other ways of writing this rule. The example is intended to illustrate how abstraction can bring out process-modeling relations (e.g., proportional cause) that are implicit in the domain rationale and prototype method. Although it is not worked out here in detail, notice also how the metarule is written in terms of modifying an SSM of a system-modification plan, namely reducing the amount of the quantitatively-variable

```
TASK: ANTICIPATE-PLAN-SIDE-EFFECT
FOCUS: $PLAN-ACTION

IF:     (AND  (QUANTITATIVELY-VARIABLE $PLAN-ACTION)
              (CAUSES $PLAN-ACTION $EFFECT)
              (PROPORTIONAL-CAUSE $PLAN-ACTION $EFFECT)
              (NOT (DESIRABLE $EFFECT))
              (CAUSES $FINDING $EFFECT)
              (SAME CNTXT $FINDING))

THEN :        (TASK REDUCE-AMOUNT $PLAN-ACTION)
```

Fig. 38. Example of a domain principle from XPLAIN.

action (called "therapy" in medicine). Moving from the domain principle of Fig. 37 to the abstract metarule of Fig. 38 exemplifies the abstraction process of dropping domain-specific terms and expressing the inference procedure in terms of a process model and some action that depends upon it. A knowledge engineer with this vocabulary will have a great advantage in analyzing other problems. With a metarule like this in the toolkit (contrasted with the domain principle in Fig. 37), the next program should be much easier to build. In particular, the idea of a quantitatively-variable system-control action is very basic to system modeling and should be taught as part of knowledge engineering methodology.

8.2. KNACK: distinguishing system models and object instantiation

KNACK is a knowledge acquisition program for a class of expert systems that generate reports. KNACK acquires an adequate domain model by interacting with the knowledge engineer to generalize sample reports. Figure 39 illustrates this, revealing how a knowledge acquisition program that develops a domain model from cases is essentially the inverse of the inference procedure, which generates an SSM (e.g., a sample report) from a general model (recall Fig. 18). Applying our modeling orientation, we can describe KNACK's task in terms of operators for completing a sample report, abstracting it, and checking the generalization for completeness and consistency. Other operators, not shown here, complete and abstract the domain model that constitutes the subject material of the report. Once again, a listing of these operators would be of value for a wide variety of knowledge acquisition programs that reason about cases. For example, how do KNACK's operators relate to those used in apprenticeship learning? Research is not reported this way, therefore it is difficult to tell from the literature.

Each expert system developed by KNACK is called a WRINGER. Of the seven WRINGERs constructed to date, we will consider here the three WRINGERs that generate environmental reports. Figure 40 shows that there are actually two general domain models in each WRINGER, a model of an electromechanical system and a model of a report. This is not untypical. Recall

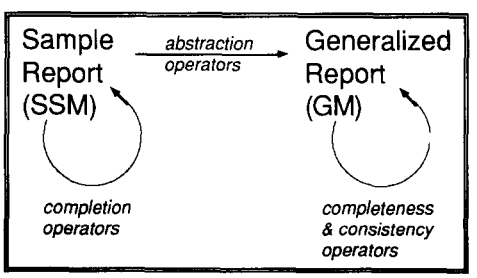

Fig. 39. KNACK constructs a general model of a report from a situation-specific model.

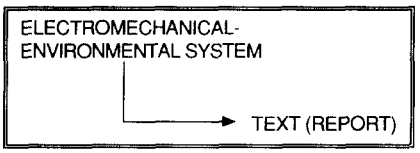

Fig. 40. A WRINGER contains models of two kinds of systems.

that SACON has general models of some physical structure (e.g., an airplane wing) and of computer programs (i.e., possible configurations of structural analysis programs) [24]. GRUNDY has general models of people and of books. MYCIN has general models of patients, diseases, and therapies. Indeed, every engineering problem (broadly including medicine) will involve relating models of different systems, in which the environment or human actions constitute systems (as shown by Fig. 21). Of particular interest here is that a report is a static system; it has no process characteristics. Nevertheless, a report has structure, and in KNACK types of texts are modeled by a taxonomy of chapter and section relations, a qualitative model.

Klinker et al. [53] give a good description of the electromechanical–environmental system in system-modeling terms.

> A ⟨X⟩ *system* performs a set of *functions* and comprises a set of interrelated *components*.
>
> An *environment* produces a set of *conditions* under which a ⟨X⟩ system must function, each of which may affect system components via a set of *media*.
>
> The effect of a condition on system components may be modified by some *provisions*, each of which can . . . be affected by a set of conditions via a set of media.

With domain-specific terms replaced by variables (e.g., conditions, media), it is clear that the authors intend to abstract their domain model beyond the problem at hand. The result is a very suggestive, general perspective that improves our understanding of the nature of the system being modeled, in terms of what is known about its parts, functions, and its operation in some environment. This is precisely the kind of coherent, system-model perspective we found missing in XPLAIN, which of course predates this research by almost a decade.

KNACK generates domain-specific control rules for specific WRINGER expert systems (Fig. 41), which can be viewed as instantiations of its underlying abstract systems model (like the relationship between PROTEGE and OPAL [70]). Because the abstract systems model is implicit in KNACK's OPS5 rules-for-writing-rules and its graphics editor for acquiring a domain model, the capability shown by NEOMYCIN for explanation, student modeling, strategic

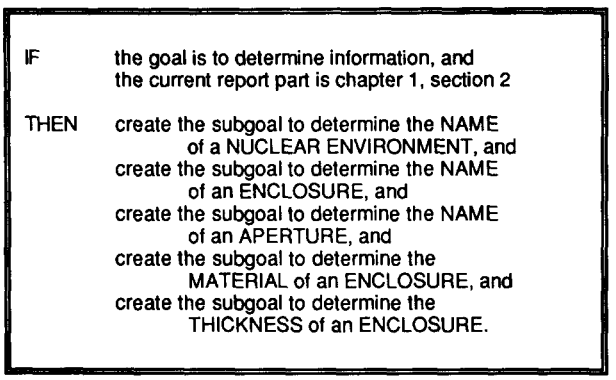

Fig. 41. Domain-dependent strategy in a WRINGER expert system, generated by KNACK.

tutoring, and explanation-based learning cannot be realized as easily. There is nothing inherently wrong with compiling an abstract inference procedure into domain-specific rules (just as MYCIN's rules might be recovered by compiling NEOMYCIN's metarules with respect to the given domain model). However, an operator-process model analysis (Sections 6 and 7) requires a more structured representation than either KNACK or published papers about it provide.

How might we use abstract subtasks and metarules to represent the system-model and control strategy that KNACK uses to generate the rule shown in Fig. 41? This is a kind of reverse engineering, since the papers show the compiled rules, not the general models and inference procedure from which they were generated. We want to replace domain terms such as "nuclear environment" by variables and explicit domain relations between the subgoals in the five action statements. We want to define a subtask for modifying an SSM, which here represents a text. The specific operator corresponding to the rule given in Fig. 41 should acquire initial information about a report part and set up the corresponding text in the report. Figure 42 shows how this would be done in a surprisingly simple way, transferring most of the structure in the WRINGER rule to the domain model and exploiting the object-instantiation primitives of NEOMYCIN (inherited from EMYCIN).

The metarule shown in Fig. 42 begins with a report part (e.g., CH1S2) and for every subtopic (e.g., NUCLEAR-ENVIRONMENT) invokes a subtask that sets up an instance corresponding to the subtopic in the SSM. In EMYCIN, the instantiation procedure (coded in LISP) uses general information about object relations to prompt the user about the existence of instances, as well as to acquire initial information about them. Figure 40 shows these relations as part of the general model, precisely as they are expressed in EMYCIN/HERACLES. For example, an APERTURE object is associated with enclosures, so an instance of that must be set up first. Initial information about the enclosure name, material, and thickness is requested at this time.

```
Subtasks and Metarules

SUBTASK:     DETERMINE-INFORMATION
FOCUS:       $REPORTPART
TASKTYPE:    ITERATIVE
LOCALVARS:   ($TOPIC)

IF      (SECTIONTOPIC $REPORTPART $TOPIC)
THEN    (TASK CREATE-CONTEXT $TOPIC)


Domain Facts

<< represent report structure as a classification>>
CH1S2
 PARENT:            CHAPTER1
 CHILDREN:          (CH1S2SS1 CH1S2SS2 CH1S2SS3)
 SECTIONTOPIC:      (NUCLEAR-ENVIRONMENT APERTURE)

<<represent the general system as a tree of contexts>>
ENCLOSURE
 ASSOCWITH:         (SYSTEM)
 OFFSPRING:         (APERTURE)
 PROMPT1ST:         ("Please list the enclosures of the *")
 INITIALDATA:       (ENCLOSURE-NAME MATERIAL THICKNESS)

APERTURE
 ASSOCWITH:         (ENCLOSURE)
 OFFSPRING:         (APERTURE-PROTECTION)
 PROMPT1ST:         ("Please list the apertures of the *")
 INITIALDATA:       (APERTURE-NAME)
```

Fig. 42. HERACLES version of KNACK/WRINGER DETERMINE-INFORMATION strategy.

The prompts shown here are those used by the WRINGER. This tree of instances, which is part of the SSM, corresponds to an EMYCIN context tree.

Further comparisons reveal that general object-instantiation operations are collected and abstracted in EMYCIN's interpreter and context-tree mechanism. Figure 43 shows a KNACK-generated domain-specific object-instantia-

```
IF    the goal is to integrate a strategy result, and
      the result is a value for the NAME of an ENCLOSURE, and
      a SYSTEM with some NAME is known

THEN create a concept ENCLOSURE with a NAME characteristic,
            and instantiate it with that value,
      create a link that the SYSTEM COMPRISES the ENCLOSURE.

<< This is handled by the EMYCIN interpreter
      when setting up the ENCLOSURE-1 instance >>

ENCLOSURE-1
      CRE8:                 ENCLOSURE
      UPPOINT:              SYSTEM-1
      ENCLOSURE-NAME:       (S-280C)
```

Fig. 43. WRINGER rule for connecting an instance to its parent is handled by EMYCIN's interpreter.

tion rule, which EMYCIN's interpreter handles automatically. In comparing knowledge acquisition tools, it is important to realize that what is expressed as a rule in one system (e.g., KNACK) might be part of the interpreter of another system (e.g., EMYCIN). Indeed, the implementation of HERACLES as a specialization of the EMYCIN architecture (Appendix A.2) shows that EMYCIN provides a more general and useful frame language than is often realized.

Other strategy rules given by Klinker et al. can be abstracted; however, we are unable to develop a full specification of a WRINGER in terms of SSM operators. We can see from the KNACK paper that the SSM could be expressed as a context tree of chapters and sections, generated from a general model that classifies the structure of reports. The text to be printed is modeled in this SSM by attributes of chapters and sections, for example, the heading of a section. Subtasks and metarules will set up the text SSM and finally print out the text associated with each node.

However, the published strategy rules written by KNACK have clauses whose relation we cannot easily infer. For example, one rule states:

> if . . . a NUCLEAR-ENVIRONMENT with NAME EMP is known, and an ENCLOSURE with some NAME is known, then print . . . ⟨ENCLOSURE.NAME⟩ . . . .

What is the role of the first clause? In so far as we cannot infer the structure of the domain model from the published literature, we do not have an adequate description of KNACK for the work to be replicated or applied to similar problems. A basic claim of this paper is that the reformulations into HERACLES given here are at a level, using system-modeling language, that provides more adequate communication of knowledge engineering research.

8.3. How abstract should control knowledge be?

Abstracting inference procedures has allowed us to compare control strategies from different programs, express domain knowledge more concisely, and make control common to many programs (e.g., instance creation) separate and available for reuse. The role of knowledge is viewed uniformly in terms of system models (e.g., a SECTIONTOPIC of a TEXT) and operators for constructing an SSM (e.g., VALIDATE-INFORMATION, DEFINE-FRAGMENT).

But abstraction is relative. Are there any guidelines for developing new subtasks and metarules? Generalization beyond what the examples at hand can support might produce obscure code that is more difficult to understand and modify. Common sense suggests that the metarules be specific enough to facilitate reuse (i.e., the language should be suggestive of other problems for the same system-modeling task) and general enough to be reusable (i.e.,

replace primitive domain terms by variables). The examples in this paper are intended to support two general claims:

Claim 1. *Strategies mentioning domain terms are unnecessary*; they can always be rewritten by introducing "system-model" variables and relations between domain concepts (i.e., causal, temporal, spatial, and subtype relations).

Claim 2. *The resulting abstract inference procedure is a kind of grammar ordinarily reusable in a variety of domains*; it can be interpreted for both *recognition* and *generation* purposes (e.g., consultation, explanation, knowledge acquisition, student modeling).

Figure 44 shows how, for a given task, *control knowledge may be specialized depending on the process characteristics of the system being modeled.* Configuration generally views a system in terms of its structures; planning views a system in terms of its processes. In particular, designing a building or text report is generally viewed as a problem of configuring structures. Static systems are generally described in terms of spatial orientation of parts, as opposed to causal and temporal relations between structural changes. (Of course, civil engineers will model stress and fatigue from use; service designers will model dynamic subsystems such as heating and lighting.)

The point of this diagram is that *inference operators can be viewed with respect to a classification of system types*, with general configuration operators (e.g., for instance creation) applying to all systems, but other operators specialized depending on the relations of the domain model which will be incorporated in the SSM. A WRINGER text has no causal/temporal relations, so the inference procedure for configuring a text uses no operators mentioning them. One can imagine building a future expert system by including and specializing operators from a library organized according to Fig. 44, with the description of causal, temporal, and spatial relations in the general model available as specifications that drive the choice of relational network (Section 7), SSM structure, and inference operators.

In this respect, the content of strategic knowledge is intricately tied to an

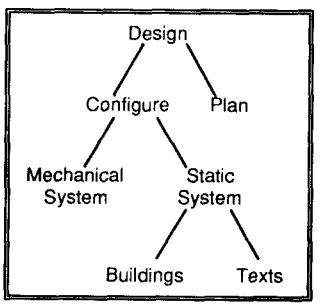

Fig. 44. Alternative levels of abstraction for task-specific architectures.

ontology of system types, such that learning to be a knowledge engineer involves learning how configuring a text is different from configuring a stereo and how constructing a configuration model differs from constructing a diagnostic model. Depending on the frequency of problems a particular client encounters, there might be knowledge acquisition tools for "configuring static systems" or even one specialized for configuring environmental reports.

Figure 45 shows more specifically how we abstract from given expert systems to create a family of tools. Using the HERACLES subtask-metarule language, we developed a program called TOPO for configuring computer networks (e.g., workstations, servers, local-area network, modem links).The program models some *physical-organizational structure* (POS) (e.g., the sites, buildings and working groups of a regional golf association) and the *information processing* required (e.g., membership rolls, due payments, and game statistics).[12] This model is mapped onto the design for a computer network that provides the required services, a *service network*. The configuration inference procedure, as in HERACLES-DX, is domain-general. Together with the language for expressing a POS and network configuration, the subtasks and metarules constitute a task-specific shell, which we call HERACLES-CX.

What kinds of expert systems could be built with HERACLES-CX? The

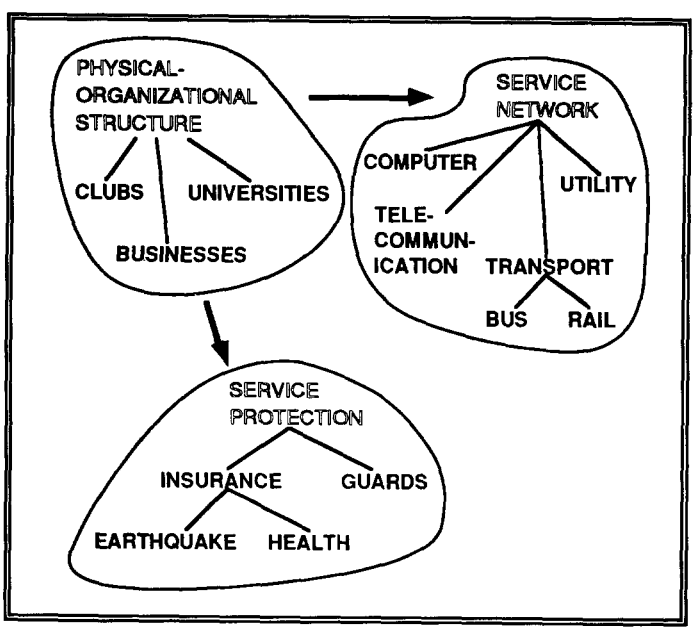

Fig. 45. Types of system configurations developed from a physical-organizational model.

[12] For simplicity, we assume the information-processing model is part of the POS in Fig. 45. TOPO is intended to be a front-end to XCON; it was developed with Monique Barbanson.

figure shows two possibilities: We could adapt the POS model (and inference operators) to layout other kinds of service networks (e.g., a telecommunication network). Or we could use the POS model to configure "protection services", such as insurance policies. The idea is that a model of buildings and site layout and the nature and location of working groups is useful for both configuration of a service network and configuration of an insurance policy. Certainly different information will be required (so the domain-modeling language will change), but it is plausible that the POS model developed for TOPO will serve as a good starting point. Indeed, as pointed out in the discussion of MYCIN's context tree (Section 4.5), we observe that *the first reasoning step in many expert systems is the creation of a POS model.* Are there just a few kinds of POS models (e.g., models of clubs, universities, businesses and subtypes of these), relative to the kinds of subsystems or processes that require a POS model in the first step of design (e.g., service networks, protection services)? This is the kind of recurrent structure we seek to discover and exploit in the development of knowledge acquisition tools.

Our chief interest at this time in the development of knowledge engineering is to represent specific domain models and control knowledge so that the common elements can be recognized and collected into more general tools. The idea of organizing modeling tools according to the process characteristics in the system being modeled (Figs. 44 and 45) is further exemplified by a study of student modeling programs [25]. In order to determine the generality of student modeling techniques relative to subject material domains, we classify domains (systems) in two dimensions according to the operators in the object system (*axiomatic* as in algebra versus *open* as in chemistry) and the SSM inference operators (*algorithmic* as in subtraction versus *heuristic* as in medical diagnosis). Without this kind of framework, it is difficult to appreciate the contribution of any given expert system (or student modeling program) to the development of qualitative modeling techniques. In short, a useful level of abstraction in a research report or program design relates the given program to a classification of tasks and domains, exemplified here by the descriptions of XPLAIN, WRINGERs, and TOPO.

8.4. What is the relation between inference operators, Generic Tasks, and problem solving methods?

Other researchers have advocated the analysis and decomposition of knowledge bases in terms of general or "generic" components. How do these analyses compare?

Chandrasekaran's *Generic Tasks* [16] are generally at the same level as subtasks in NEOMYCIN. Generic Tasks are procedural operators for modifying the SSM; they can be combined to configure a complete inference procedure for an expert system. Example operators, labeled by the names of the reusable code modules, are:

- CSRL: explore-refine classification,
- Hyper: prototype matching and evaluation,
- Pierce: abductive assembly and evaluation of composite hypotheses,
- WWHI: prediction by abstracting state changes,
- Idable: data retrieval and inference.

These operators are generally independent of the overall modeling task. For example, data retrieval and inference, corresponding to NEOMYCIN's FIND-OUT and PROCESS-FINDING subtasks, could be used for diagnosis as well as for design tasks. The important idea is that these operators have been abstracted from particular expert systems, and they can now be functionally composed and specialized in new application-specific programs.

In contrast, McDermott [63] describes knowledge acquisition tools at a higher level, in terms of the overall inference procedure for constructing an SSM:

- MOLE: cover-and-differentiate diagnosis,
- SALT: propose-and-revise design,
- KNACK: acquire-and-present design,
- SIZZLE: extrapolate-by-analogy design.

By this analysis, we would add the entire HERACLES-DX system as an example in this list (perhaps with the subdescription "propose-explore-refine, group-differentiate, broaden-confirm diagnosis"). McDermott calls these *role-limiting methods*, referring to the way in which an inference procedure specifies (and hence restricts) how knowledge is applied in solving a particular problem. McDermott thus emphasizes that to provide a problem solving method is also to structure the domain model in terms of roles particular relations and types of rules will play (cf. discussion of GUIDON-MANAGE and ODYSSEUS in Section 5). Role-limiting methods, in contrast with Chandrasekaran's Generic Tasks, tend to be specific to the overall system-modeling task, hence the use of the words "design" and "diagnosis" in the tool descriptions. Rather than being functionally composed, they are specialized to create application-specific programs.

We can now summarize the relation between NEOMYCIN's inference operators, Generic Tasks, and role-limiting methods as follows:

- A (*role-limiting*) *problem solving method* is a procedure composed of several *inference operators* that—through their controlled interaction—form a situation-specific model that satisfies task constraints. For example, NEOMYCIN's subtasks together implement a variant of MOLE's COVER-AND-DIFFERENTIATE method.
- Furthermore, a subtree in NEOMYCIN's subtask hierarchy (e.g., EXPLORE-AND-REFINE), which appears in multiple problem solving methods, is packaged and distributed separately in what Chandrasekaran calls a *Generic Task*.

In conclusion, the model-construction-operator perspective reveals that *different researchers have pursued different levels of generality in formalizing control knowledge*: single SSM operators correspond to a NEOMYCIN subtask; subprocedures of one or more operators correspond to a Generic Task; a complete inference procedure corresponds to a role-limiting method. A good property of Generic Task and role-limiting methods analyses is that they each adopt a single point of view, compared to lists that mix inference methods with modeling purposes (e.g., a typical "problem types" list, appearing in an expert systems tutorial, is "evidence gathering, stepwise refinement, stepwise assembly, design, and monitoring"). However, there is no way of arguing for the completeness of either the Generic Tasks or role-limiting methods lists or even comparing two such lists without a dimensional analysis of the primitives that underlie them, precisely the role of mathematical analysis. In particular, the set-mapping and SSM constraint descriptions (Section 6) provide a means for systematically describing and hence generating candidate Generic Tasks and role-limiting methods (e.g., as in Table 2). We might look for these operators or operator combinations in existing expert systems to fill out our library or use them for selecting a new application that will extend the capabilities of an existing tool.

9. Historical perspective

One of the main ideas of this paper is that expert systems incorporate models of systems that are mostly nonnumeric in character, and by this technique AI programming has produced a new modeling method. We have termed this qualitative modeling; however, representational modeling or relational modeling are also appropriate names.[13] The important point is that expert systems as computer programs are using a form of modeling that distinguishes them from traditional programs.

9.1. Changing views of relational networks

To see more clearly how AI programming has introduced a new modeling method that is worthy of being named and promoted in itself—independent of how it has been applied to develop intelligent programs—consider how our

[13] Recalling Fig. 29, processes represented qualitatively include the domain system, inference, planning, data gathering, and discourse. A great deal of AI research has been concerned with "formal reasoning" (e.g., geometry, subtraction, theorem proving). From the perspective of system modeling, formal reasoning is degenerate because model notations are manipulated (e.g., a child's subtraction problem), but there is no specific system in the world being modeled. Nevertheless, the inference process is represented qualitatively (e.g., the subtraction process is represented in BUGGY as a procedural hierarchy [14]). Extensive discussion of formal versus physical systems modeling and algorithmic versus heuristic inference procedures appears in [25].

views of relational networks have changed in computer programming over the past two decades:

- Programmers have traditionally used node and link representations to represent processes; flowcharts are a leading example. In expert systems *the relational network is a representation, an object in itself, that is selectively interpreted for different purposes.* It is not executed or run so much as read, examined, and modified like a database.
- In early AI research, complex relational networks are used for associating arbitrary things; semantic nets are a leading example. In expert systems we find *systematic macrostructures (e.g., hierarchies, state-transition nets) relating states, substances, and processes.* It is not just a network where facts are stored, but rather a representation of a process or system.
- Traditional programs read and print numbers and symbols; FORTRAN and COBOL are leading languages for this. Expert systems create and manipulate linked-node graphs (the SSM), whose *intermediate structure often opportunistically drives the computation.* It is not just a data structure that is created and modified, but a model of structures and processes, usually on different levels of detail, whose form can be systematically interpreted for detecting incomplete or inconsistent facts.

Notice the shift from describing a process as something that runs directly to *something that is constructed by the program itself* and selectively interpreted, perhaps the hallmark of a representation. Notice also that the shift from semantic networks to the idea of classification (e.g., as developed in UNITS, KL-ONE, and CYC [57]) is in the direction of not just relating arbitrary things, but modeling processes (a point emphasized in CYC research).

A key distinction that has been little recognized is that knowledge bases are not just complex networks (let alone unorganized pots of rules), but have a macrostructure that can be viewed in terms of types of graphs, such as types of transition networks. We have made progress by focusing on what a node represents (state, substance, process) and what the links mean (cause, spatial connection, temporal sequence, functional subprocess). Early studies focused on subtype relations of concepts [7, 95], which unfortunately misses the process-modeling aspects. These studies tend to view a knowledge base as a collection of facts that simply exists like a database ("Clyde is a grey elephant"). In an expert system—or indeed any program that must reason about the world and take action—*descriptions of systems are represented and employed for some purpose; they are not arbitrary.* Finally, the key idea emphasized in this paper is that the program's reasoning can be usefully characterized in terms of set and graph operators for manipulating this model, thus relegating discussions of conceptual structures, rules, frames, blackboards, etc. to the level of notational style and program implementation.

Table 6
Evolving purposes for using graphs to represent systems.

Types of graph representations of systems	Purposes/views
Process classifications Causal (substance/process) networks Functional composition networks Structural composition hierarchies	Model of a system taken as an object itself, interpreted and modified by diverse processes (*links indicate causal, temporal, spatial, or functional relations, and inheritance of these*).
Object feature classifications (Entity-relation database)	Taxonomy of physical appearance and behavior of similar objects (*links indicate inheritance of static, inherent properties*).
Semantic networks	Description of meaning of concepts (*links indicate relations between objects and events*).
AND/OR subgoal tree Dataflow diagram	Derivational description of facts via inferences and calculations (*links indicate conditional transformations*).
Petri nets Flowcharts Influence diagrams (Decision trees)	Description of conditions and operations constituting states of a process (*links indicate flow of control*).

Table 6 provides a particular cut on the historical development of qualitative modeling, emphasizing what aspects of an object system's behavior or operation are represented by different types of networks. Notice in particular how Petri nets, decision trees, and influence diagrams are similar to flowcharts because they do not describe how a system is put together or how it works, so much as procedurally specify what happens under particular conditions. In practice, these networks are often viewed as types of causal-associational networks. AND/OR subgoal trees (e.g., Fig. 13) and dataflow diagrams represent a collection of facts as a stream that is transformed by mathematical operations or logical syllogisms. The development of relational databases—particularly its expression in the "entity-relation model" [8]—has paralleled the development of process modeling in AI and is close in spirit to early systems like UNITS and KL-ONE. This view emphasizes that collections of things are organized into a classification hierarchy, and primacy is placed on representing the relations that define this classification (corresponding to slots in frame systems). This is certainly an important part of the elephant's hide.

9.2. Decision support and critiquing models

The system-modeling perspective also provides an easy way of contrasting decision support programs and expert systems. Generally speaking, *decision support programs partially automate the modeling task, leaving the user to provide a general system model or to determine how an SSM should be applied.* For example, *risk analysis* involves predicting how a system being modeled will behave, using heuristics to determine whether undesired events ("risks") will occur. An *assessment* step involves determining which system attributes or environmental inputs are causally related to the risk/undesired event. For example, a decision support system for designing new detergents might accept a detergent design as input and use simulation to determine that there is a risk of excess suds when washing certain kinds of synthetic materials; assessment could track this back to a particular component of the detergent. Such a program helps a user test and evaluate designs; more typically an expert system (as defined by Fig. 21) would generate the initial detergent design or given a supplied design would go on to say how it might be manufactured.

The *critiquing model of consultation* (e.g., SOPHIE's hypothesis evaluator [9], MYCIN's therapy explanation system [18] and similar methods developed by Langlotz and Shortliffe [56] and Miller and Black [65]) are essentially decision support tools, accepting an SSM or action plan from the user and comparing to what is internally generated. Instructional programs often provide scenarios or cases for developing and exercising general models; they also provide representations by which the student can describe an SSM, as in Anderson's geometry program. Notice that in decision support, critiquing, and instructional tools the idea of modeling is salient because the model itself is an object of study or elaboration. In consultation programs like MYCIN the diagnostic model is not presented so much for the user to ponder over and use, but as a justification of the therapy (indeed, even the idea of an intermediate SSM is disguised by the goals, viz. "conclude that E.coli is one of the organisms *that therapy should cover for*").

A decision support or critiquing approach is valuable for problems in which it is difficult to automate action planning (e.g., controlling a company) or in which there are too many situations for the knowledge engineer to anticipate in the general model, using a fixed set of concepts. For complex design or planning (control) tasks, *the decision support system becomes a kind of knowledge acquisition program which helps the user develop a general model and evaluate it by simulating its behavior in different situations.* Critiquing has turned out to be especially valuable for medical diagnosis and therapy, both to remind physicians about the complexities of the general model and to keep them engaged and feeling responsible for patient care [56]. The main contribution of AI to decision support systems is in providing these qualitative modeling techniques, in contrast with previous probabilistic, nonrepresentational approaches [39], which limit the way models can be expressed.

9.3. *Grammars, nonlinear modeling, automata theory*

Another way of understanding qualitative modeling techniques is to track them back even further to their origins in the theory of computation and cybernetics. This is a chapter that AI textbooks consistently omit, and it is perhaps responsible for the perceived lack of coherence in AI research. The 1970s and 1980s generation brought up as computer programmers trying to create intelligent machines did not see itself as building on the nonlinear modeling and systems theory from which AI was born [82].

As an example, consider how control knowledge might be related to Post's original production formalism from linguistics [49]:

- A blackboard is a data structure for posting alternative parses of some expression or behavior (indeed, this is precisely what HEARSAY is doing, posting alternative parses of input sentences, starting of course at the level of sounds rather than words).
- An inference procedure expressed as subtasks and rewrite (meta)rules is a higher-order grammar for controlling the parsing/recognition and generation process (indeed, this is explicit in ODYSSEUS, which uses NEOMYCIN's subtasks and metarules to parse a sequence of student data requests).
- The relations for representing processes in the domain, inference, planning, and communication models (Fig. 29) constitute grammatical (more precisely, lexical) distinctions for organizing objects in the object system; for example, "substance" and "follow-up question" constitute grammatical distinctions in a domain model for diagnosis (Fig. 20), and sentences (propositions) are facts about the domain.

In short, it is easy to view blackboards, inference procedures, and relational languages as direct extensions of the production rule formalism developed for formally representing languages. Of particular interest is how the sentences generated and recognized in expert systems and tutorial programs are not just utterances in an arbitrary conversation, but are about some system that is being modeled for some purpose. Grosz's [35] research on task-oriented discourse is an early natural language effort that adopts this perspective. Indeed, how could there be a natural language program that carries on purposeful dialogue without a domain model and inference procedure inside it?

As indicated, ODYSSEUS provides a particularly good example of how an abstract inference procedure can be viewed as a grammar for the inference process. Figure 46 shows a sequence of three data requests, listed on the right-hand side, parsed in terms of subtasks and domain rules, in boxes. The program indicates alternative reasons why a question might have been asked, using a bottom-up analysis, then groups them in terms of higher-order subtasks that could now be applied given the state of the SSM, a top-down analysis. For

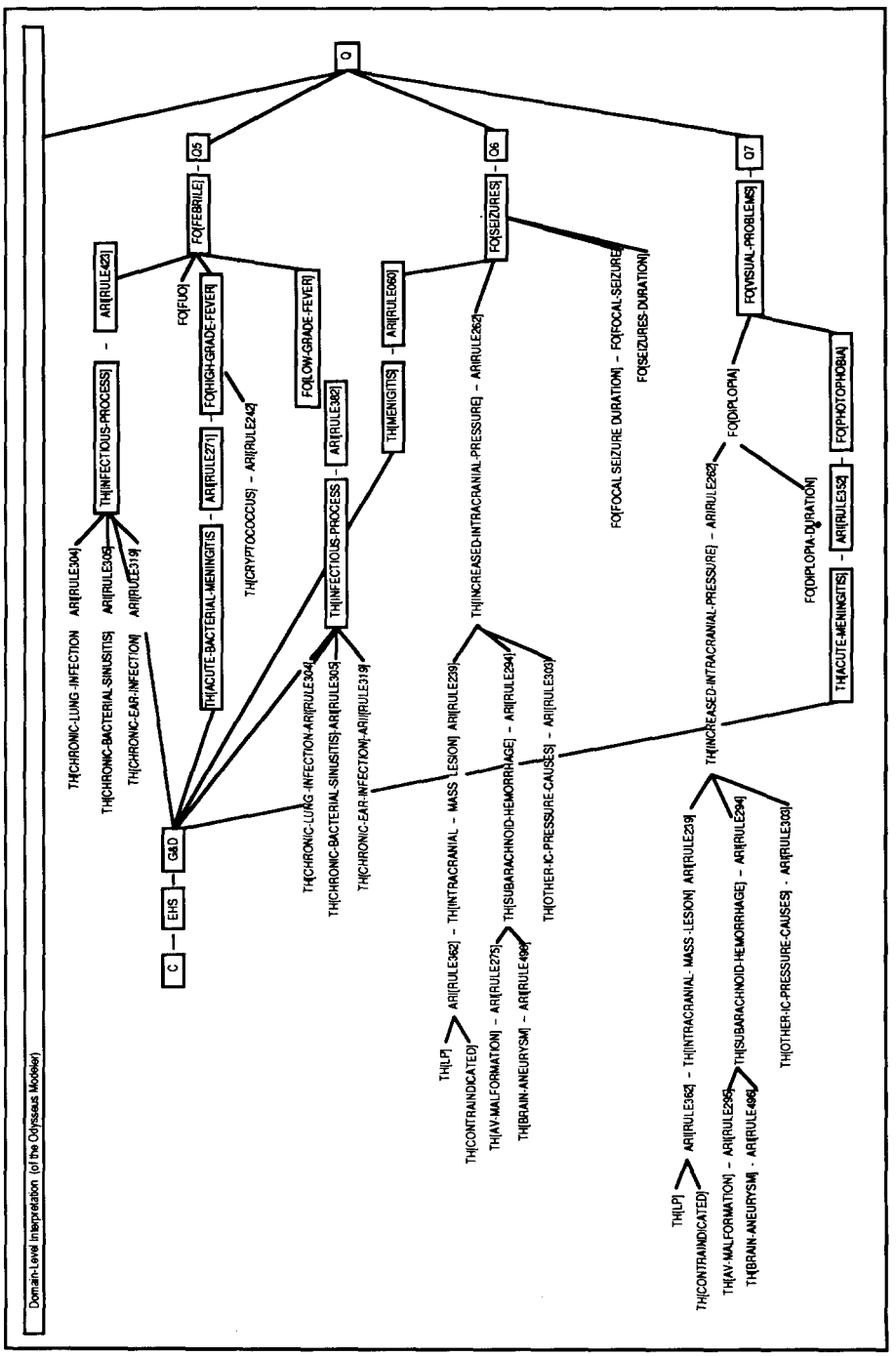

Fig. 46. ODYSSEUS parse of a student's data requests.

example, the student's inquiry about seizures (FINDOUT/Seizures, Q6) might have been asked to determine whether the disease is caused by an Intracranial-mass-lesion, Subarachnoid-hemorrhage, and so on. The successful parse indicates that the query is consistent with TEST-HYPOTHESIS/Meningitis, as part of the process of looking up in GROUP-AND-DIFFERENTIATE. By the same analysis, the question about fever (FINDOUT/Febrile, Q5) has three consistent interpretations; we cannot determine from this information alone whether the student is focusing on Acute-bacterial-meningitis or Infectious-process, however NEOMYCIN would select Infectious-process first.

This kind of automated protocol analysis is not possible using MYCIN's rules or even TEIRESIAS because *a grammatical analysis requires that the rules not mention domain terms* (analogous to using variables like "noun" and "verb" in natural language grammars, rather than specific words). The relations serve to classify the findings and hypotheses, in the same manner that a natural language lexicon classifies words (e.g., passive verb, demonstrative pronoun), determining which grammar rules will control their assembly into sentences. The subtask structure serves as a higher-order representation of the entire consultative interview, analogous to a grammar characterizing an essay or particular type of exposition [61]. In fact, the characterization of inference procedures as grammars plays a major role in criticisms about the adequacy of expert systems as models of intelligence, based on the idea that grammars are an observer's description of patterns of behavior and cannot be equated with the mechanism itself (Section 6) [30]. The explicitness of the grammar and parse in ODYSSEUS reminds up that expert systems and, more specifically, cognitive models share the power and limitations of natural language grammars as mechanisms for generating behavior.

Much more could be said about the origins of AI in nonlinear modeling and automata theory. The ideas of compartmentalization (near decomposability [82]), topology, adaptiveness, informational flow, state transition, and rationality have their modern beginnings in attempts within cybernetics to model open and nonlinear systems [72, 92]). We can better present qualitative modeling to scientists and engineers if we return to these origins, explain the path we have taken, and show how our representational methods solve some of the early problems. For example, Bertalanffy summarized the difficulty of representing discontinuities (nonlinearity) in system behavior:

> Representation by differential equations is too restricted for a theory to include biological systems and calculating machines where discontinuities are ubiquitous.

In our knowledge acquisition tools, we are implicitly claiming that we have found methods for representing just such a wide variety of systems. Kuipers' [54] qualitative simulation research is one good example where the relation between qualitative modeling and differential calculus is made explicit. Similar-

ly, relational modeling techniques can be fruitfully tracked through their use in describing formal machines (automata), to modeling reasoning processes in mathematical psychology, to representing states in physical systems such as the human body.

Connecting qualitative modeling to its origins in cybernetics has recently taken on new importance with the emergence of new mathematical and nonrepresentational techniques for modeling complex, dynamic systems [85]. As new methods are found for building intelligent machines there is a danger that knowledge engineering methods will be tossed aside. We must recognize that these methods are more general, and have wider applicability, than their developers first intended. For example, qualitative modeling could be used for representing neural processes. To identify expert system techniques as just "one way to model reasoning" is to confuse a modeling method with the particular theories it has been used to represent. If we place qualitative modeling in historical perspective, we are more likely to get the ideas disseminated for general use by scientists and engineers, and better articulate the tools we will need to advance our understanding of intelligence.

10. Conclusions

This paper can be viewed as the culmination of an inquiry that began by studying MYCIN's rules [21], led to generalizing patterns in expert systems [24], and now involves identifying methods that distinguish AI programming as a whole. A central claim is that it is productive to view AI programming in terms of a modeling methodology that represents causal, temporal, and spatial relations in systems. (Formal reasoning such as geometry problem solving is ungrounded model manipulation; the numbers, variables, lines and angles represent situations, but not particular systems in the world.) From this perspective, control knowledge consists of the procedures for constructing situation-specific models. Different representations, problem solving architectures, knowledge acquisition tools, specific expert systems, and even different areas of AI research can then be systematically related by this model construction perspective, in terms of types of relational networks, process models, inference operators, system domains, and modeling purposes (tasks).

Our exposition unfolded by studying NEOMYCIN's metarules, attempting to relate them to representations used in other programs and formalize principles by which metarules are written. We found that multiple views, incorporating basic mathematical and programming terminology, are useful for describing NEOMYCIN's subtasks: graph-manipulation operators, set operators, grammatical categories. The value of this formalization is exemplified by the GUIDON-DEBUG program, which uses constraints on the form of the SSM to detect problem solving failures and direct explanation-

based learning; the ODYSSEUS program which models diagnostic strategy by abstracting sequences of data requests; and the GUIDON-MANAGE program which allows a user to dictate diagnostic strategy and generates plausible actions (hints) in the same language.

The programming technique that enables these capabilities represents control knowledge as stylized procedures that order and control metarules, whose premises use a form of the predicate calculus (see appendices). *This representation reveals that each time we write a new procedure for interpreting a representation, we define new relations that classify its constructs.* Thus, we find that classifications and procedures are defined in terms of each other. The representation of strategic knowledge that results from stating control knowledge abstractly, using variables in place of primitive terms, is not domain-independent, but *domain-general*, in the sense that the language or relations can be made more general than any one system being modeled by using spatial, temporal, causal, and subtype distinctions (a perspective by which all systems can be described).

One of the surprises in our investigation is that control knowledge cannot be derived from the form of the SSM alone, but instead must always contain assumptions about the world (involving case population and constraints on the nature of the interaction between program and the world). We found that the blackboard approach for representing and reasoning about alternative control regimes provides flexibility for looking ahead and resuming interrupted operations, that must be handled in an ad hoc way in NEOMYCIN. Similarly, we found that NEOMYCIN's representation of methods as objects provides a way of making explicit the relations between SSM elements and inference operators ("knowledge sources") which are often coded in blackboard systems in an ad hoc manner.

A major step is realizing that model construction recurs at three nested levels: the domain, inference procedure, and communication with the world. Specifically, the idea of a blackboard or SSM recurs at each level, which is to say that each level involves some general model of processes and an inference procedure for constructing a situation-specific representation (e.g., a model of a patient's disease, a diagnostic plan, a discourse/explanation process for communicating with the user). To see this, we need to extend the idea of a model to include classification descriptions and not just simulations. We then realize that *a key contribution of AI programming is in using relational networks to represent processes.* Representation research develops types of classifications and state-transition networks that are linked and composed in taxonomies, causal-associational networks, and structure-function models.

The resulting picture is very general. Its power comes not from telling us specifically how to develop a good model for some purpose, but how to relate the diverse research that has attacked specific problems. We can relate blackboards to metarules, NEOMYCIN to ABEL, heuristic classification to

qualitative process simulation, XPLAIN to KNACK, and Generic Tasks to role-limiting methods. Put another way, we can now relate representational constructs, expert systems in a given domain, inference methods, knowledge acquisition programs, and reasoning strategies, all from a model construction perspective.

A key observation is that existing programs differ greatly in the grain size by which inference procedures are described. For example, in SOAR procedures are described very generally in terms of operator/operand application and caching; in NEOMYCIN procedures are described in terms of relations between operands and operators for constructing sets of operands; in BB1 procedures are described in terms of coping with resource limitations for applying applicable operators or observing available operands. These complementary views are integrated by the model construction perspective. Rather than continuing to talk past each other (SOAR: matching situations to achieve goals; NEOMYCIN: calling subprocedures to accomplish tasks; BB1: invoking knowledge sources to modify the blackboard), we can make progress by adopting a uniform graph-set-operator language.

In general, we see a remarkable shift from talking about rules in MYCIN to talking about the structure of graphs and types of models of processes in NEOMYCIN, made all the more dramatic by the use of the same examples through the sequence of papers [21, 24]. This study suggests that AI researchers should be wary of describing reasoning in terms of "answering questions" or "achieving goals". At its heart, *reasoning involves forming a model of some system in the world in order to take action* (Fig. 21). This perspective relates expert systems to AI in the large, systematically relating knowledge, inference, and planning (Fig. 29). Given any AI program, we can ask, "What are the systems being modeled?, What are the structure and process characteristics of this system?, What kind of relational network is used to represent these structures and processes?, What is the inference procedure for constructing a situation-specific model?, How is this model employed by later reasoning phases, evaluated, or conveyed to the user?" Rather than simply asking about a new problem domain, "Is there real-world knowledge that allows classification?" we might ask, "Must the system be modeled as open in its interactions with its environment?, Is there a known etiological hierarchy?, Are there stages or developmental descriptions involving trends and frequency of behaviors?, What experience have people had with this system in rebuilding, modifying, assembling it in different situations?" Thus, knowledge engineering is a form of systems analysis.

From these observations, we can conjecture where expert systems and knowledge acquisition research are headed. First, much more attention could be placed on the recurrent macrostructures in domain models. Hierarchies and transition networks need to be viewed as objects in their own right and classified (Fig. 34). Such classifications can then be mapped to a taxonomy of

system types, according to whether the system has dynamic properties, whether a classification of interactive-historical patterns is known, and whether a behavioral (causal–associational net) or functional simulation is possible and useful.

Paralleling the approach in software engineering, indeed merging with it [90], object libraries need to be collected and shared. These objects will include and be organized by the relational language, types of process model, system type, inference operators, form of the SSM, constraints on interactions with the world, and modeling purpose. In this respect, HERACLES-DX, BB1-ACCORD, KNACK, Generic Tasks, etc. are precisely the systems that must now be studied and integrated. Automatic programming could play a role here, specifically if the SSM graph-operator representation is adopted for relating the diverse approaches.

Researchers might begin by articulating and collecting the constraints that existing inference procedures place on the form of the SSM (Fig. 32 and Table 3) and a program's interaction with its environment. This would be useful for both automatic programming and failure analysis (GUIDON-DEBUG). Indeed, the model construction view is essential for formalizing the process of evaluating expert systems. Other research areas that could adopt this perspective include parallel computation (e.g., organize the modules as equivalence classes of operators, not just "knowledge sources") and learning (e.g., focus on how new relations are defined by new procedures for interpreting a representation). Reuse of representations is called *repurposing* in the multimedia domain; one could apply repurposing of knowledge representations by knowledge acquisition programs to the problem of indexing and composing new sequences of multimedia presentations.

Probably the greatest opportunity lies in integrating numeric simulation and qualitative modeling. Examples abound of engineering applications in which a qualitative model is used to aid in the design of a system [48], to generate scenarios to be simulated (including reasoning about boundary conditions as in SOPHIE), to make heuristic decisions as embedded models of human agents (e.g., in a manufacturing plant), and to analyze simulation results (particularly abstracting and explaining trends in numeric data). Applications to chemistry [13], biology [45], and genetics [36] are well known, but their contribution to scientific modeling has been obscured by the emphasis on "expertise". Similar applications in operations research [33] amply demonstrate how representational and parametric modeling complement each other.

In conclusion, a historical perspective suggests that we take care to *separate the modeling contributions of AI research from specific theories of intelligence.* In many respects, the techniques of knowledge engineering now have a life of their own, as they are adopted and developed by scientists and engineers who have no particular interest in cognitive science or automated reasoning, per se. Qualitative modeling should be studied, formalized, and presented as a

contribution to computer science and systems analysis. On this basis, links to software engineering and operations research will be more quickly realized and taught. Indeed, in many respects this generation of AI research may come to be viewed not so much for the particular capabilities of the programs that were developed, but for the generality of the methods, which as computational formalisms are arguably as novel and wide-sweeping in their impact as Newton's calculus.

Appendix A. HERACLES' architecture

The architecture is described here as a general shell, then more specifically in terms of how it is implemented on top of EMYCIN, followed by details about the subtask interpreter and the metarule compiler. The full relational language used in NEOMYCIN's metarules is also categorized.

A.1. Nesting of shells, application programs, and model-manipulation routines

HERACLES is a framework for encoding a domain model and an inference procedure. The language is called CTMR, standing for constraint, subtask, metarule, and relation (Fig. A.1). Out of these constructs, a specific inference procedure of subtasks and metarules can be written, which will mention relations by which the domain knowledge will be represented. HERACLES-DX is the expert system shell that contains the subtasks, metarules, and relations extracted from NEOMYCIN. CASTER, a rudimentary sand casting diagnostic system demonstrates the generality of this shell [89]. A new shell called HERACLES-CX is currently being implemented, with a new set of subtasks, metarules, and relations for configuration, based on a computer network layout expert system we are developing. Thus, HERACLES is not an expert system shell, but a framework for developing shells. As shown in Fig. A.1, the tutoring, explanation, student modeling, and knowledge acquisition programs are separate modules that rely only on the CTMR representation, not on the particular subtasks, metarules, or relations. Additional knowledge required for any of these programs is expressed in the same CTMR language (e.g., new relations required by the explanation program classify existing domain relations).

A.2. Metarules, relations, and subtasks

Originally NEOMYCIN's metarule premises were coded in LISP. In a hybrid system called MRS/NEOMYCIN [31], we represented metarule premises in MRS, a logic-programming language that provides a framework for multiple representations of knowledge and control of reasoning [37]. We also recoded the interpreter in MRS rules and placed a simple deliberation-action loop at

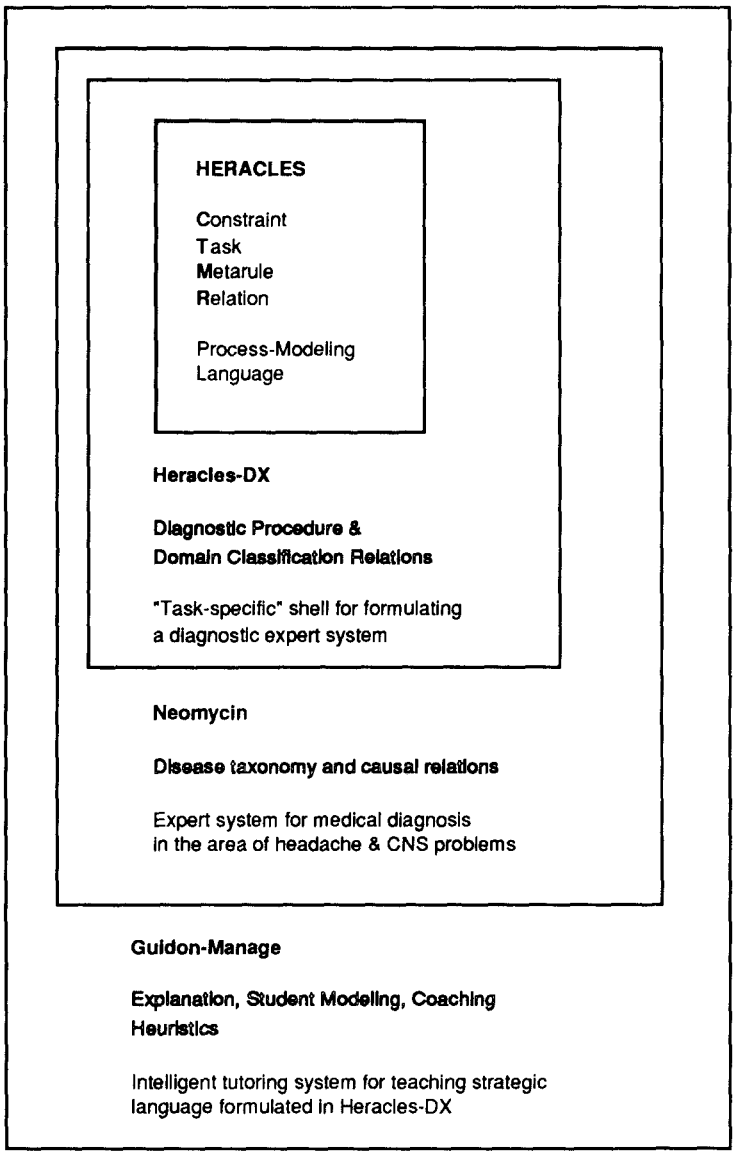

Fig. A.1. Nesting of language, task-specific shell, expert system, and communication procedures.

the top. Unfortunately, this slowed down the program by an order of magnitude and made the procedure too obscure to read or maintain. A compromise design works far better, with the interpreter written in LISP and the metarules coded in a variant of MRS, which are compiled into LISP. This provides the well-structured language required for bookkeeping and interpretation by the explanation, tutoring, and student modeling, knowledge acquisition programs, without sacrificing runtime efficiency. Figure A.2 shows how an

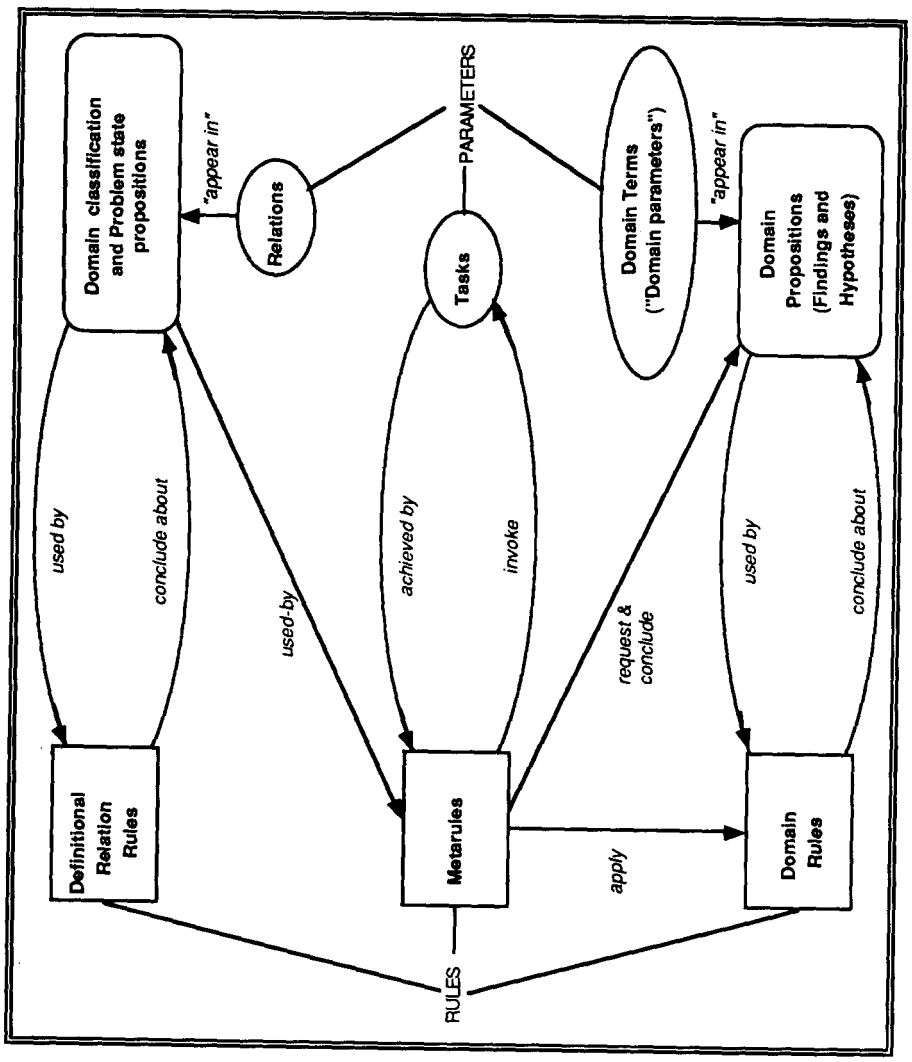

Fig. A.2. Representation of an inference procedure in HERACLES.

inference procedure is represented in HERACLES, which we view to be a major contribution of this research. HERACLES has three kinds of "concept" or "parameter" objects and three kinds of "rule" objects. We use the EMYCIN terms *parameters* and *rules* because the system is actually implemented on top of EMYCIN. Parameters are specialized as domain relations, control subtasks, and domain terms, conditionally inferred and invoked by definitional rules, metarules, and domain rules. We use "relation" in the mathematical sense to refer to both predicates and functions. Findings and hypotheses are two classes of domain term; more informally, they refer to propositions in the SSM (so informally we say that "the patient has meningitis" is a hypothesis). Subtasks are accomplished by an interpreter that applies metarules (described in more detail below). Propositions used by metarules premises (e.g., (EXPLAINED-BY $F $H)) can be inferred definitionally by rules or can be inferred by procedural attachment (accessing LISP structures). These propositions are both static and dynamic. They classify domain propositions and domain rules, as well as characterize the problem solving state (e.g., the contents of the SSM and bookkeeping information about subtask application). Additional relations that classify subtasks used by the subtask interpreter, and the interpreters in GUIDON-MANAGE, ODYSSEUS, etc. are not shown here. Metarule actions apply domain rules, request (from the user) or assert domain propositions (e.g., Fig. 6), or invoke other subtasks. In particular, the subtask FINDOUT, recoded and expanded from the original EMYCIN program, uses all of these methods to infer domain propositions. In HERACLES all domain rules are applied directly by metarules rather than by uncontrolled backward chaining. Only domain rules mention domain terms directly; other rules use variables. No rule may mention another rule by name.

To coordinate with the subtasks and metarules, the EMYCIN function CONCLUDE maintains a list of new finding and hypothesis assertions (part of NEOMYCIN's blackboard). A much more complicated certainty factor scheme distinguishes between inherited and direct belief (from rules); the cumulative CF of a hypothesis is defined to be its direct CF combined with the cumulative CF of its parent (because it inherits the slots and hence the evidence of its parent). Finally, EMYCIN primitives for setting up data tables, creating an instance in the context tree, etc. may be used directly in metarule actions.

A.3. The subtask interpreter

A HERACLES subtask is a procedure, represented as an ordered and controlled set of metarules. The control of a task's metarules is specified by properties of the subtask:

- The *subtask focus*, which is the argument of the subtask. Only one focus is allowed.

- The task's ordered *metarules*.
- The *end condition*, which may abort the subtask or any subtask when it becomes true. Aborting can occur only while the do-during metarules are being applied. The end condition is tested after each metarule of a subtask succeeds. A subtask may also be marked to prevent its abortion.
- Ordered rules to be applied before or after the metarules (used only for bookkeeping).
- The *subtask type*, which specifies how the metarules are to be applied. There are two dimensions to the subtask type: *simple* or *iterative*, and *try-all* or *not-try-all*.

The combinations of subtask types provide four ways of controlling the metarules:

- *Simple and try-all.* The rules are applied once each, in order (equivalent to a LISP PROG statement). Each time a metarule succeeds, the end condition is tested.
- *Simple and not-try-all.* The rules are applied in sequence until one succeeds or the end condition succeeds (equivalent to a LISP COND statement).
- *Iterative and try-all.* All the rules are applied in sequence. If there are one or more successes, the process is started over. The process stops when all the rules in the sequence fail or the end condition succeeds (equivalent to a pure production system).
- *Iterative and not-try-all.* Same as for iterative and try-all, except that the process is restarted after a single metarule succeeds (equivalent to a "for" loop).

In addition, each subtask can have local variables that appear in metarule premises, allowing a binding to be passed to the action as a subtask focus.

A.4. The metarule compiler

The metarule compiler converts a metarule premise, expressed as a conjunction of propositions, to a LISP function. Relations concluded by definitional rules are coded as separate functions. Primitive domain relations are compiled as direct LISP operations (e.g., GETPROP). The backtracking necessary to match the variables in the premise, including a capability for finding all possible matches is compiled as nested iterations and set collection operations.

Table A.1 lists the ways in which primitive domain relations can be implemented in LISP, with examples.

A miscellaneous category of general relations must be handled specially by the compiler, including the quadruple relation between a finding, hypothesis, domain rule, and certainty factor (EVIDENCEFOR) and relations that can be easily optimized by the compiler (e.g., MEMBER and NULL).

Table A.1
IMPLEMENTATION classification of domain relations.

Implementation	Example	Interpretation
FLAG	NEW-DIFFERENTIAL	T or NIL variable
VARIABLE	STRONGCOMP.WGHT	LISP variable
LIST	DIFFERENTIAL	LISP list
PROPMARK	ASKFIRST	T or NIL property
PROPLIST	CHILDREN	List-valued property
PROPVAL	PROMPT	Arbitrary property
FUNCTION	SAMEP	LISP function
METARULE-PREMISE-RELATION	TAXREFINE?	Determined by a definitional rule

To write a compiler for MRS-style rules in general is difficult. However, several features of HERACLES-DX metarules and a few simplifications made it easy to write the compiler:

- Only one rule concludes about each metarule premise relation. Where necessary, rules were combined into a single rule with a disjunction.
- Relations are either predicates or functions, rather than being used in both ways. For example, (CHILDREN $HYP $CHILD) is only used as a functional generator, never as a predicate to test whether a given candidate is a child of a given hypothesis. This was not a deliberate design choice—all of the 166 relations in HERACLES-DX satisfy this property.
- Functional relations are all single-valued (except EVIDENCEFOR). Consequently, backtracking to find matches for variables in conjunctions can be expressed as nested *find x suchthat* or *therexists an x suchthat* loops; failure of the inner loop and return to the next outer loop for a new variable match is equivalent to backtracking.
- Rule conjuncts are ordered manually so that a variable is found (by a functional relation) before it is tested (by a predicate relation). This is a natural way to write the rules.
- Inverse relations are defined so that the LISP atom with the property corresponding to the relation is the first variable in the relation. For example, the functional relation CHILDREN, as in (CHILDREN $HYP $CHILD), is used when $HYP is known. Again, this occurred naturally rather than being a deliberate design choice.
- Redundant clauses are used in disjuncts, rather than being factored out, i.e.,

  (AND ⟨common clauses⟩ (OR d1 d2 . . . dn))

  Consequently, a few rules are awkward to read.

In general, the compiler's code is easier to understand than the original LISP metarules from NEOMYCIN because it does not use constructs like *thereis* and *never*, which require mental gymnastics to logically invert and combine.

Besides the IMPLEMENTATION property, domain relations may have PREDICATE and MULTIPLEMATCH properties. PREDICATE indicates that the matched variable need not be saved. MULTIPLEMATCH means that the rule that defines the relation should be matched as many times as possible. In essence, the compiler changes

(*find* ⟨*var*⟩ *in* ⟨*list*⟩ *suchthat* . . .)

to

(*for* ⟨*var*⟩ *in* ⟨*list*⟩ *collect* ⟨*var*⟩ *when* . . .) .

Essentially, the task of the compiler can be viewed in terms of finding a binding for a variable, testing it, setting it, or simply checking to see if a value exists. In particular, the majority of metarules use some definitional relation of type MULTIPLEMATCH, returning a list, which is passed to the metarule's action. This pattern suggested the set-manipulation way of describing subtasks (Section 6).

In conclusion, the use of prefix predicate calculus notation as a specification language for metarules is convenient and allows efficient compilation.

A.5. Relations used in HERACLES-DX

The relations used in NEOMYCIN and CASTER metarules and definitional rules are categorized as domain, dynamic belief, dynamic search or focus bookkeeping, and computational. Inverses (e.g., caused-by) are not listed. Primitive terms are $PARM, $RULE, $CF, and $CNTXT (an instance of a domain class). All other terms and relations are defined in these terms. Indentation indicates hierarchical definition of new terms. For example, a nonspecific finding is a kind of finding. These relations are generally implemented as LISP structures; the dynamic belief and computational relations are implemented as LISP functions. The remaining relations are composites of the others, defined by rules written in MRS.

Relations classifying findings and hypotheses

```
(FINDING $PARM)
  (SOFT-DATA $FINDING)
  (HARD-DATA $FINDING)
  (NONSPECIFIC $FINDING)
  (REDFLAG $FINDING)
(HYPOTHESIS $PARM)
  (STATE-CATEGORY $HYP)
  (TAXONOMIC $HYP)
    (PARENTOF $TAXPARM $PARENT)
    (COMPLEX $TAXPARM)
```

(CAUSES $HYP1 $HYP2)
(SUBSUMES $FINDING1 $FINDING2)
(PROCESSQ $FINDING1 $FINDING2)
(CLARIFYQ $FINDING1 $FINDING2)
(SOURCE $FINDING1 $FINDING2)
(SCREENS $FINDING1 $FINDING2)
(PROCESS-FEATURES $HYP $SLOT $VAL $FINDING)

(ALWAYS-SPECIFY $FINDING)
(ASKFIRST $FINDING)
(PROMPT $FINDING $VAL)
(BOOLEAN $PARM)
(MULTIVALUED $PARM)
(TABLE $BLOCKPARM $FINDING)

(ENABLINGQ $HYP $FINDING)
(SUGGESTS $PARM $HYP)
(TRIGGERS $PARM $HYP)

Relations classifying domain rules

(ANTECEDENT-IN $FINDING $RULE)
(APPLICABLE? $RULE $CNTXT $FLG)
(EVIDENCEFOR? $PARM $HYP $RULE $CF)
(COMMONCASERULES $HYP $RULE)
(UNUSUALCASERULES $HYP $RULE)

(PREMISE $RULE $VAL)
(ACTION $RULE $VAL)
(ANTECEDENT $RULE)
  (TRIGGER $RULE)
(SCREEN $RULE)

Belief relations

(BELIEF $HYP $CF)
(CUMCF-VALUE $HYP $CF)
(MAX-CONSIDERED-HYP-CUMCF $CF)

(PREVIEW $CNTXT $RULE)

(DEFINITE $CNTXT $PARM)
(DEFIS $CNTXT $PARM $VALUE)
(DEFNOT $CNTXT $PARM $VALUE)
(NOTKNOWN $CNTXT $PARM)
(SAME $CNTXT $PARM $VALUE $CF)
(SAMEP $CNTXT $PARM)

Dynamic search or focus relations

(CONSIDERED $HYP)
(DESCENDENTS-EXPLORED $TAXPARM)
(PARENTS-EXPLORED $TAXPARM)

(APPLIEDTOP $RULE $CNTXT)
(DONTASKP $CNTXT $PARM)
(TRACEDP $CNTXT $PARM)

(SPECIFICS-REQUESTED $FINDING)
(SUBSUMPTION-CONCLUDED $FINDING)
(USERSUPPLIED $FINDING)

(TASK-COMPLETED $TASK)
(TASK-COMPLETED-FOCUS $TASK $FOCUS)

Dynamic relations describing the SSM

(CURFOCUS $HYP)
(DIFFERENTIAL $HYP)
(NEW.DIFFERENTIAL)
(WIDER.DIFFERENTIAL)
(DIFFERENTIAL.COMPACT)

(NEXT-HARD-DATAQ $FINDING)
(NEW.DATA $FINDING)
(PARTPROC.DATA $FINDING)

Computational relations

(ABS $ARG $RESULT)
(CFCOMBINE $CF1 $CF2 $RESULT)
(EQ $ARG1 $ARG2)
(GREATERP $ARG1 $ARG2)
(LESSP $ARG1 $ARG2)
(MINUS $ARG1 $ARG2 $RESULT)
(MINUS $ARG)
(NULL $ARG)
(TIMES $ARG1 $ARG2 $RESULT)

(FIRST-ONE $LIST $RESULT)
(LENGTH $LIST $RESULT)
(MEMBER $MEN $SET)
(SINGLETON? $LIST)

(PREDICATE $REL)

(IMPLEMENTATION $REL $VAL)
(MULTIPLEMATCH $REL)

(UNIFY $PATTERN $FACT)

Relations defined by rules (composites)

(ACTIVE.HYP? $HYP)
(ALWAYS-SPECIFY? $FINDING)
(ANTECEDENT.RULES? $PARM $RULE)
(ANY.ANCESTOR? $HYP1 $HYP2)
(BESTCOMPETITOR $CURRENTHYP $BETTERHYP $BHCF)
(BESTHYP $HYP)
(CHILDOF $HYP $CHILD)
(CLARIFY.QUESTIONS $FINDING $PROCPARM)
(DIFF.EXPLAINED $FINDING)
(DIFF.NOTPARENTS-EXPLORED?)
(DIFF.NOTPURSUED?)
(ELIGIBLECHILD)
(ENABLING.QUESTIONS $HYP $RULE)
(EXPLAINEDBY $FINDING $HYP)
(EXPLORE.CHILD? $HYP $H)
(EXPLORE.HYP? $HYP)
(EXPLORE.SIBLING? $OLDFOCUS $HYP)
(NEXTGENERALQ? $FOCUSQ)
(PARTPROC.NOTELABORATED? $FINDING)
(PARTPROC.SUGGESTRULES? $PARM $RULE)
(POP-FINDING $NEWDATA $FINDING)
(POP-HYPOTHESIS $NEWDATA $HYPOTHESIS)
(POP-REDFLAG-FINDING $NEWDATA $FINDING)
(PROCESS-QUESTIONS? $PARM $PROCTYPEPARM)
(REFINABLE? $HYP)
(REFINABLENODE? $OLDFOCUS $FOCUSCHILD)
(REMAINING.QUESTIONS $HYP $RULE)
(SINGLE.TOPCAUSE $FOCUS)
(SOURCEOF $PARM $SOURCE)
(STRONG-COMPETITOR? $CURRENTHYP $BESTCOMP)
(SUBSUMPTION.SUBTRACED $CNTXT $PARM)
(SUBSUMPTION.SUPERFO $CNTXT $PARM)
(SUBSUMPTION.SUPERTRACED $CNTXT $PARM)
(SUBSUMPTION. SUPERUNK $CNTXT $PARM)
(SUGGESTRULES? $PARM $RULE)
(SUPERS.NOTRACED $PARM $SUPERPARM)

```
(TAXANCESTOR $HYP1 $HYP2)
(TAXREFINE? $HYP)
(TOP.UNCONFIRMED? $ANCESTOR)
(TOPUNCON $ANCESTOR $HYP)
(TRIGGERQ $HYP $RULE)
(TRIGGERS? $FINDING $RULE)
(UNAPPLIED? $RULE)
(UNCLARIFIED-FINDING $NEW.DATA $FINDING)
(UNEXPLOREDDIFF.COMPACT? $HYP)
(UPDATE.DIFF.RULES? $FINDING $RULE)
(WAITINGEVIDRULES? $HYP $RULE)
(WEAK.EVIDENCE.ONLY? $HYP)
```

Acknowledgement and historical notes

NEOMYCIN was first designed and implemented in November 1980 with the assistance of Reed Letsinger and the late Timothy Beckett, MD. Conrad Bock is responsible for the recoding into MRS in 1982 that inspired and provided the data for this paper. Many of the programs mentioned here were co-designed and implemented by Stanford students: David Wilkins (ODYSSEUS, 1986), Arif Merchant and Diane Hasling (explanation), Bob London (IMAGE, 1982), Mark Richer (GUIDON-WATCH, 1985), Tim Thompson (CASTER, 1986), Naomi Rodolitz (GUIDON-MANAGE, 1987), John Sotos (explanation); they were assisted by system programmers Steven Barnhouse, David Leserman, Steve Oliphant, and Monique Barbanson. I am indebted to Bruce Buchanan for providing the research environment at the Stanford Knowledge Systems Laboratory that made this work possible. NEOMYCIN, its tutoring and knowledge acquisition adjuncts, and the development of the qualitative process modeling metaphor exemplifies "The MYCIN Experiments" [12] that Bruce directed and promoted from 1972 to 1987. John McDermott, Georg Klinker, Mark Musen, and reviewers made many thoughtful suggestions for revising this paper.

The programs described here were implemented in InterLisp-D on Xerox 1100 Series machines. Computational resources were provided by the SUMEX-AIM facility (NIH Grant RR00785), managed by Tom Rindfleisch. This research was supported in part by ONR and ARI Contract N00014-79C-0302 under the supervision of Marshall Farr and Susan Chipman and by a gift from the Josiah Macy Jr Foundation, Award B852005 in a funding program for medical cognitive science championed by John Bruer. Funding is currently provided by Digital Equipment in a program supervised by John McDermott, and IRL, via gifts from the Philips-Netherlands and Xerox corporations.

References

[1] J.S. Aikins, Prototypical knowledge for expert systems, *Artif. Intell.* **20** (2) (1980) 163–210.

[2] J.H. Alexander, M.J. Freiling, S.J. Shulman, J.L. Staley, S. Rehfuss and M. Messick, Knowledge level engineering: ontological analysis, in: *Proceedings AAAI-86*, Philadelphia, PA (1986) 963–968.

[3] J.R. Anderson, C.F. Boyle and G. Yost, The geometry tutor, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985) 1–7.

[4] G. Bateson, *Steps to an Ecology of Mind* (Ballentine Books, New York, 1972).

[5] J.S. Bennet, ROGET: a knowledge-based consultant for acquiring the conceptual structure of an expert system, Report No. HPP-80-7, Computer Science Department, Stanford University, Stanford, CA (1980).

[6] R.J. Brachman, On the epistemological status of semantic networks, in: N.V. Findler, ed., *Associative Networks: Representation and Use of Knowledge by Computers* (Academic Press, New York, 1979) 3–50.

[7] R.J. Brachman, What's in a concept: structural foundations for semantic networks, *Int. J. Man-Mach. Stud.* **9**, 127–152.

[8] M.L. Brodie, J. Mylopoulos and J.W. Schmidt, *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages* (Springer, New York, 1984).

[9] J.S. Brown, R.B. Burton and J. de Kleer, Pedagogical, natural language, and knowledge engineering techniques in Sophie I, II, and III, in: S.D. and J.S. Brown, eds., *Intelligent Tutoring Systems* (Academic Press, Orlando, FL, 1982).

[10] J.S. Brown, R.B. Burton and F. Zydbel, A model-driven question-answering system for mixed-initiative computer-assisted instruction, *IEEE Trans. Syst. Man Cybern.* **3** (3) (1973) 248–257.

[11] J.S. Brown, A. Collins and G. Harris, Artificial intelligence and learning strategies, in: H. O'Neill, ed., *Learning Strategies* (Academic Press, New York, 1977).

[12] B.G. Buchanan and E.H. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Heuristic Programming Project* (Addison-Wesley, Reading, MA, 1984).

[13] B.G. Buchanan, G. Sutherland and E.A. Feigenbaum, Heuristic Dendral: a program for generating explanatory hypotheses in organic chemistry, in: B. Meltzer and D. Michie, eds., *Machine Intelligence* (Edinburgh University Press, Edinburgh, Scotland, 1969) 209–254.

[14] R.R. Burton, 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.

[15] B. Chandrasekaran, Expert systems: matching techniques to tasks, in: W. Reitman, ed., *AI Applications for Business* (Ablex, Norwood, NJ, 1984) 116–132.

[16] B. Chandrasekaran, Generic tasks in knowledge-based reasoning: characterizing systems at the "right" level of complexity, in: *Proceedings IEEE Second Conference on Artificial Intelligence Applications*, Miami, FL (1985).

[17] B. Chandrasekaran, Towards a taxonomy of problem solving, in: R. Engelmore, ed., *Readings from the AI Magazine* **1–5** (AAAI, Menlo Park, CA, 1988) 534–544.

[18] W.J. Clancey, An antibiotic therapy selector which provides for explanations, 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).

[19] W.J. Clancey, Methodology for building an intelligent tutoring system, in: W. Kintsch, J.R. Miller and P.G. Polson, eds., *Method and Tactics in Cognitive Science* (Erlbaum, Hillsdale, NJ, 1981).

[20] W.J. Clancey, The advantages of abstract control knowledge in expert system design, in: *Proceedings AAAI-83*, Washington, DC (1983) 74–78.

[21] W.J. Clancey, The epistemology of a rule-based expert system—a framework for explanation, *Artif. Intell.* **20** (1983) 215–251.

[22] W.J. Clancey, Acquiring, representing, and evaluating a competence model of diagnosis, in: M. Chi, R. Glaser and M. Farr, eds., *The Nature of Expertise* (Lawrence Erlbaum, Hillsdale, NJ, 1988) 343–418.

[23] W.J. Clancey, Details of the revised therapy algorithm, in: B.G. Buchanan and E.H. Shortliffe, eds., *Rule-Based Expert Systems: The MYCIN Experiments of the Heuristic Programming Project* (Addison-Wesley, Reading, MA, 1984).

[24] W.J. Clancey, Heuristic classification, *Artif. Intell.* **27** (1985) 289–350.
[25] W.J. Clancey, Qualitative student models, in: J.F. Traub, ed., *Annual Review of Computer Science* (Annual Review Inc., Palo Alto, CA, 1986) 381–450.
[26] W.J. Clancey, From Guidon to Neomycin and Heracles in twenty short lessons, in: A. van Lamsweerde, ed., *Current Issues in Expert Systems* (Academic Press, London, 1987) 79–123.
[27] W.J. Clancey, *Knowledge-Based Tutoring: The GUIDON Program* (MIT Press, Cambridge, MA, 1987).
[28] W.J. Clancey, The knowledge engineer as student: metacognitive bases for asking good questions, in: H. Mandl and A. Lesgold, eds., *Learning Issues in Intelligent Tutoring Systems* (Springer, Berlin, 1988).
[29] W.J. Clancey, *The Knowledge Level Reinterpreted: Modeling How Systems Interact* (Kluwer Academic Publishers, Boston, MA, 1989) 287–293.
[30] W.J. Clancey, The frame of reference problem in the design of intelligent machines, in: K. vanLehn and A. Newell, eds., *Architectures for Intelligence: The Twenty Second Carnegie Symposium on Cognition* (Erlbaum, Hillsdale, NJ, 1990).
[31] W.J. Clancey and C. Bock, Representing control knowledge as abstract tasks and metarules, in: L. Bolc and M.J. Coombs, eds., *Computer Expert Systems* (Springer, Heidelberg, 1988) 1–77.
[32] W.J. Clancey and R. Letsinger, 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, 1981).
[33] G.F. Cooper, NESTOR: a computer-based medical diagnostic aid that integrates causal and probabilistic knowledge, Report No. 84-48, Department of Computer Science, Stanford University, Stanford, CA (1984).
[34] R. Davis and D. Lenat, *Knowledge-Based Systems in Artificial Intelligence* (McGraw-Hill, New York, 1982).
[35] B.G. Deutsch, The structure of task-oriented dialogs, in: *Proceedings IEEE Symposium for Speech Recognition* (1974) 250–253.
[36] P. Friedland, Knowledge-based experiment design in molecular genetics, in: *Proceedings IJCAI-79*, Tokyo (1979) 285–287.
[37] M.R. Genesereth, The use of design descriptions in automated diagnosis, *Artif. Intell.* **24** (1984) 411–436.
[38] F. Gomez and B. Chandrasekaran, Knowledge organization and distribution for medical diagnosis, in: W.J. Clancey and E.H. Shortliffe, eds., *Readings in Medical Artificial Intelligence* (Addison-Wesley, Reading, MA, 1984) 320–338.
[39] A. Gorry, Computer-assisted clinical decision making, in: W.J. Clancey and E.H. Shortliffe, eds., *Readings in Medical Artificial Intelligence: The First Decade* (Addison-Wesley, Reading, MA, 1984) 18–34.
[40] J.G. Greeno, Cognitive objectives of instruction: theory of knowledge for solving problems and answering questions, in: D. Klahr, ed., *Cognition and Instruction* (Erlbaum, Hillsdale, NJ, 1976).
[41] T. Gruber, A method for acquiring strategic knowledge, *Knowledge Acquisition* **1** (1989) 255–277.
[42] P.E. Hart, Directions for AI in the eighties, *SIGART Newslett.* **79** (1982).
[43] R.T. Hartley, Representation of precedural knowledge for expert systems, in: *Proceedings Second IEEE Conference on Artificial Intelligence Applications*, Miami, FL (1985).
[44] D. Hasling, W.J. Clancey and G. Rennels, Strategic explanations in consultation, *Int. J. Man-Mach. Stud.* **20** (1) (1983) 3–19.
[45] B. Hayes-Roth, B. Buchanan, O. Lichtarge, M. Hewett, R. Altman, J. Brinkley, C. Cornelius, B. Duncan and O. Jardetzky, Protean: deriving protein structure from constraints, in: *Proceedings AAAI-86*, Philadelphia, PA (1986).
[46] B. Hayes-Roth and F. Hayes-Roth, A cognitive model of planning, *Cogn. Sci.* **3** (1979) 275–310.
[47] B. Hayes-Roth, M. Hewett, M. Vaughan Johnson and A. Garvey, ACCORD; a framework for a class of design tasks, Report No. 88-19, Knowledge Systems Laboratory, Stanford University, Stanford, CA (1988).

[48] K. Ishii, Knowledge-based design of complex mechanical systems, Civil Engineering Department, Stanford University, Stanford, CA (1987).
[49] R.Y. Kain, *Automata Theory: Machines and Languages* (McGraw-Hill, New York, 1972).
[50] P.D. Karp and D.C. Wilkins, An analysis of the distinction between deep and shallow expert systems, *Int. J. Expert Syst.* **2** (1) (1989) 1–32.
[51] R.M. Keller, Deciding what to learn, Tech. Report No. ML-TR-6, Rutgers University, New Brunswick, NJ (1986).
[52] G. Klinker, KNACK: sample-driven knowledge acquisition for reporting systems, in: S. Marcus, ed., *Automating Knowledge Acquisition for Expert Systems* (Kluwer Academic Publishers, Boston, MA, 1988) 125–174.
[53] G. Klinker, C. Boyd, S. Genetet and J. McDermott, A KNACK for knowledge acquisition, in: *Proceedings AAAI-87*, Seattle, WA (1987).
[54] B.J. Kuipers, The limits of qualitative simulation, in: A. Joshi, ed., *Proceedings IJCAI-85*, Los Angeles, CA (1985) 128–136.
[55] C.A. Kulikowski, Artificial intelligence methods and systems for medical consultation, in: W.J. Clancey and E.H. Shortliffe, eds., *Readings in Medical Artificial Intelligence* (Addison-Wesley, Reading, MA, 1984) 72–97.
[56] C.P. Langlotz and E.H. Shortliffe, Adapting a consultation system to critique user plans, *Int. J. Man-Mach. Stud.* **19** (1983) 479–496.
[57] D. Lenat, M. Prakash and M. Shepherd, Cyc: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks, *Artif. Intell. Mag.* **6** (4) (1986) 65–85.
[58] V.R. Lesser, D.D. Corkill, R.C. Whitehair and J.A. Hernandez, Focus of control through goal relationships, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 497–503.
[59] B. London and W.J. Clancey, Plan recognition strategies in student modeling: prediction and description, in: *Proceedings AAAI-82*, Pittsburgh, PA (1982) 335–338.
[60] W.J. Long, N. Shapur, M.G. Criscitiello and R. Jayes, Development and use of a causal model for reasoning about heart failure, in: P. Miller, ed., *Selected Topics in Medical Artificial Intelligence* (Springer, Berlin, 1988).
[61] W.C. Mann and S.A. Thompson, Rhetorical structure theory: a theory of text organization, Report No. ISI/RS-87-190, Information Sciences Institute, University of Southern California, Marina del Rey, CA (1987).
[62] S. Marcus, ed., *Automating Knowledge Acquisition for Expert Systems* (Kluwer Academic Publishers, Boston, MA, 1988).
[63] J. McDermott, Preliminary steps toward a taxonomy of problem-solving methods, in: S. Marcus, ed., *Automating Knowledge Acquisition for Expert Systems* (Kluwer Academic Publishers, Boston, MA, 1988) 225–256.
[64] P. Miller, Strategy selection in medical diagnosis, Report No. AI-TR-153, Artificial Intelligence Laboratory, MIT, Cambridge, MA (1975).
[65] P. Miller and H. Black, Medical plan-analysis by computer: critiquing the pharmacologic management of essential hypertension, *Comput. Biomed. Res.* **17** (1982) 38–54.
[66] T.M. Mitchell, S. Mahadevan and L.I. Steinberg, LEAP: a learning apprentice for VLSI design, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985) 573–580.
[67] J.D. Moore and W.R. Swartout, A reactive approach to explanation, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 1504–1510.
[68] D.J. Mostow, Machine transformation of advice into a heuristic search procedure, in: R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds., *Machine Learning, An Artificial Intelligence Approach* (Tioga, Palo Alto, CA, 1983) 367–404.
[69] W.R. Murray, Control for intelligent tutoring systems: a comparison of blackboard architectures and discourse management networks, Report No. R-6267, Central Engineering Lab, FMC.
[70] M.A. Musen, Automated support for building and extending expert models, *Mach. Learning* **4** (1989) 347–375.
[71] R. Neches, W.R. Swartout and J. Moore, Explainable (and maintainable) expert systems, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985) 382–389.
[72] A. Newell and H.A. Simon, *Human Problem Solving* (Prentice-Hall, Englewood Cliffs, NJ, 1972).

[73] H.P. Nii, Blackboard systems, *Artif. Intell. Mag.* **7** (3–4) (1986) 40–107.
[74] R.S. Patil, P. Szolovits and W.B. Schwartz, Casual understanding of patient illness in medical diagnosis, in: W.J. Clancey and E.H. Shortliffe, eds., *Readings in Medical Artificial Intelligence* (Addison-Wesley, Reading, MA, 1984) 339–360.
[75] H.E.J. Pople, The development of clinical expertise in the computer, in: P. Szolovits, ed., *Artificial Intelligence in Medicine* (Westview Press, Boulder, CO, 1982) 79–117.
[76] M. Richer and W.J. Clancey, GUIDON-WATCH: a graphic interface for viewing a knowledge-based system, *IEEE Comput. Graph. Appl.* **5** (11) (1985) 51–64.
[77] N.S. Rodolitz and W.J. Clancey, GUIDON-MANAGE: teaching the process of medical diagnosis, in: D. Evans and V. Patel, eds., *Medical Cognitive Science* (Bradford Books, Cambridge, MA, 1989) 313–348.
[78] A.D. Rubin, Hypothesis formation and evaluation in medical diagnosis, Report No. AI-TR-316, Artificial Intelligence Laboratory, MIT, Cambridge, MA (1975).
[79] E.D. Sacerdoti, Planning in a hierarchy of abstraction spaces, *Artif. Intell.* **5** (2) (1974) 115–135.
[80] R.C. Schank, Failure-driven memory, *Cogn. Brain Theory* **4** (1) (1981), 41–60.
[81] A.H. Schoenfeld, Episodes and executive decisions in mathematical problem solving, Tech. Report, Mathematics Department, Hamilton College, Clinton, NY (1981).
[82] H.A. Simon, *The Sciences of the Artificial* (MIT Press, Cambridge, MA, 1969).
[83] R.G. Smith, P.H. Winston, T.M. Mitchell and B.G. Buchanan, Representation and use of explicit justifications for knowledge base refinement, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985) 673–680.
[84] J. Sowa, *Conceptual structures* (Addison-Wesley, Reading, MA, 1984).
[85] L. Steels, Cooperation between distributed agents through self-organisation, in: Y. Demazeau and J.-P. Müller, eds., *Decentralized AI* (North-Holland, Amsterdam, 1990).
[86] M. Stefik, Planning with constraints, Report No. STAN-CS-80-784, Computer Science Department, Stanford University, Stanford, CA (1980).
[87] W.R. Swartout, Explaining and justifying in expert consulting programs, in: *Proceedings IJCAI-81*, Vancouver, BC (1981) 815–823.
[88] P. Szolovits and S.G. Pauker, Categorical and probabilistic reasoning in medical diagnosis, in: W.J. Clancey and E.H. Shortliffe, eds., *Readings in Medical Artificial Intelligence* (Addison-Wesley, Reading, MA, 1984) 210–240.
[89] T. Thompson and W.J. Clancey, A qualitative modeling shell for process diagnosis, *IEEE Softw.* **3** (2) (1985) 6–15.
[90] A. van Lamsweerde, B. Delcourt, E. Delor, C. Schayes and R. Champagne, Generic lifecycle support in the ALMA environment, *IEEE Trans. Softw. Eng.* **14** (6) (1988) 720–741.
[91] W. van Melle, A domain-independent system that aids in constructing knowledge-based consultation programs, Ph.D. Dissertation, Computer Science Department, Stanford University, Stanford, CA (1980).
[92] L. von Bertalanffy, *General System Theory: Foundations, Development Applications* (George Braziller, New York, 1968).
[93] D.E. Wilkins, W.J. Clancey and B.G. Buchanan, An overview of the ODYSSEUS learning apprentice, in: T.M. Mitchell, J.G. Carbonell and R.S. Michalski, eds., *Machine Learning: A Guide to Current Research* (Academic Press, New York, 1986) 369–373.
[94] D.E. Wilkins, W.J. Clancey and B.G. Buchanan, On using and evaluating differential modeling in intelligent tutoring and apprentice learning systems, in: J. Psotka, D. Massey and S. Mutter, eds., *Intelligent Tutoring Systems: Lessons Learned* (Erlbaum, Hillsdale, NJ, 1988).
[95] W.A. Woods, 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.