

Laps: Cases to Models to Complete Expert Systems

Joseph S. di Piazza and Frederick A. Helsabeck

In the short history of expert systems, a variety of approaches have been used to tackle the difficult problem of knowledge acquisition, among which are the following common types: consulting a library of models; using automatic induction from cases; and using triadic differentiation, which is repeated contrasting of two of the expected

output of an expert system with a third. To be topical, all this knowledge-acquisition research has been done in the name of constructing expert systems in an easier, faster, and more maintainable manner because there is a growing consensus that expert systems are stuck on a productivity plateau in light of first-generation tools still being used without an effective knowledge-acquisition and knowledge-structuring front end.

Contrary to many prevailing approaches to knowledge acquisition, Laps, our expert-interviewing software, begins by soliciting cases from the expert, but it does not end there. Its uniqueness lies in the fact that it interweaves knowledge gathering, organizing, and testing. Laps begins with a case in the form of a sample solution path elicited from the domain expert. This sample solution path is refined by a process called dechunking, which facilitates finding a model of the expert's reasoning process. The model guides the determination of the structure of alternatives tables at an effective level of abstraction. Once these tables have been set up, the expert is able to produce row after row on his own until a complete rule base is built. A rule generator currently produces rules in Clips or M.1 syntax.

Triadic differentiation is used in the ground-breaking ETS and in the follow-on Aquinas (Boose, Bradshaw, and Shema 1988; Shaw 1988) and in the two commercial knowledge-acquisition tools, Nextra (Gaines 1988) and Auto-Intelligence (Parsaye 1987). Induction from cases has been used in at least two commercial tools: VP

Expert and Auto-Intelligence. Creation of a library of models is championed in Europe by Breuker et al. (1987) and in the United States by Chandrasekaran (1983) and McDermott (1988).

Related to the knowledge-acquisition literature is the work on the verification and validation of expert systems, especially the Eva testing software (Stochowitz et al. 1988). Broadly speaking, this software aims to deter-



mine that the right kind of knowledge is expressed in the right form. However, unlike Eva, Laps (logic aids for problem solving), our software for knowledge acquisition, does not separate knowledge acquisition from verification and validation.

Laps was given its name because it contains logic tools to aid the expert's reasoning to be complete, and so on, while he is decompiling his problem-solving knowledge. Moreover, it is the unique thesis on which the project that produced Laps is based that the following four operations should be carried out as parts interwoven into one process: knowledge gathering or acquisition, knowledge structuring or modeling, knowledge testing or validation and verification, and knowledge encoding (di Piazza 1990).

Where do we start knowledge acquisition? In our experience, the extraction of cases seems to be the most effective way to begin knowledge acquisition for any expert system because case description is the easiest way for the expert to express him(her)self. To our knowledge, Becker (1988) is currently one of the few other researchers who make case elicitation the origin of knowledge acquisition and also make case analysis the follow-up stage. Solving cases or problems is the experts' daily fare, so they can pontificate on them endlessly, although they are notorious for being laconic in their case descriptions. With the exception of the iterative inductive approach to knowledge acquisition, the previously stated approaches do not begin with cases, which is where Laps begins. Although automatic induction also begins with cases, this approach, besides being tedious, carries other inadequacies with it.

As we note, Laps assists the expert in sequentially expressing his case knowledge. Also, it builds on this knowledge in addressing what to do with these cases. However, unlike the pure inductive, or case-based, approach, Laps aims to help the often too terse expert and the knowledge engineer extract a model from the cases and then flesh out this model into a complete expert system. Toward the end of this article, more of an appraisal of the other approaches, including the non-case-based, is offered.

A Brief Description of Laps: Its Three Sessions

Laps takes a case-based, expertise-decompilation approach, seeking to interleave knowledge acquisition, organization, and validation and verification. The description to follow is a

sketch of the current three sessions of Laps. The first, or case, session extracts one or more solved cases, or sample solution paths. This *sample solution path* consists of starting facts or statements usually followed by test results and at least some reasoning steps or intermediate conclusions that the expert takes en route to his final conclusion. Associated with each statement is a list of reasons that, as we see, is in the form of numbers for statements already occurring on the list. The expert is free to run as many paths as he desires, although he usually describes only a few cases—some are routine and others difficult.

The second session *dechunks*, or extracts steps that the user-expert might have omitted in the original solution path, thus helping to unearth hidden knowledge. This knowledge could be an explanation, heuristics, or a recurring pattern of reasoning. This additional knowledge could become the source of a *model*, or abstract representation, of the essentials of the domain.

The third session of Laps manages the development of a set of *tables*, typically, one table for each conclusion-row that is extracted by either of the previous two sessions. A familiar device to all users, tables are used in Laps as a means to organize an expert's knowledge in a complete manner. When the table is first displayed, it contains a single row corresponding to one of the conclusions in one of the sample solutions. By the end of the session, Laps has helped the user to produce a table that, as we see, has undergone a number of completeness and consistency checks by direct questioning and system-generated guidance in a modified depth-first order of completion.

After any session, it is possible to invoke Laps's rule maker and, thereby, make rules coded into M.1 or Clips syntax. The expert or end user can then run the rules using the corresponding inference engine. The user of Laps has all the advantages of consultation sessions to throw adventitiously dreamed up cases at the rules in an attempt to find gaps in its logic. However, it is the purpose of Laps to provide aids for preventing gaps or errors and, thereby, a heavy reliance on the chance-laden use of repeated consultation sessions.

Laps: From Cases to a Model

This section deals with the transformation of a carefully selected case into a model through knowledge decompilation. A specific solved problem is used as a running illustration throughout this article.

. . . contains logic tools to aid the expert's reasoning to be complete . . . while he is decompiling his problem-solving knowledge.

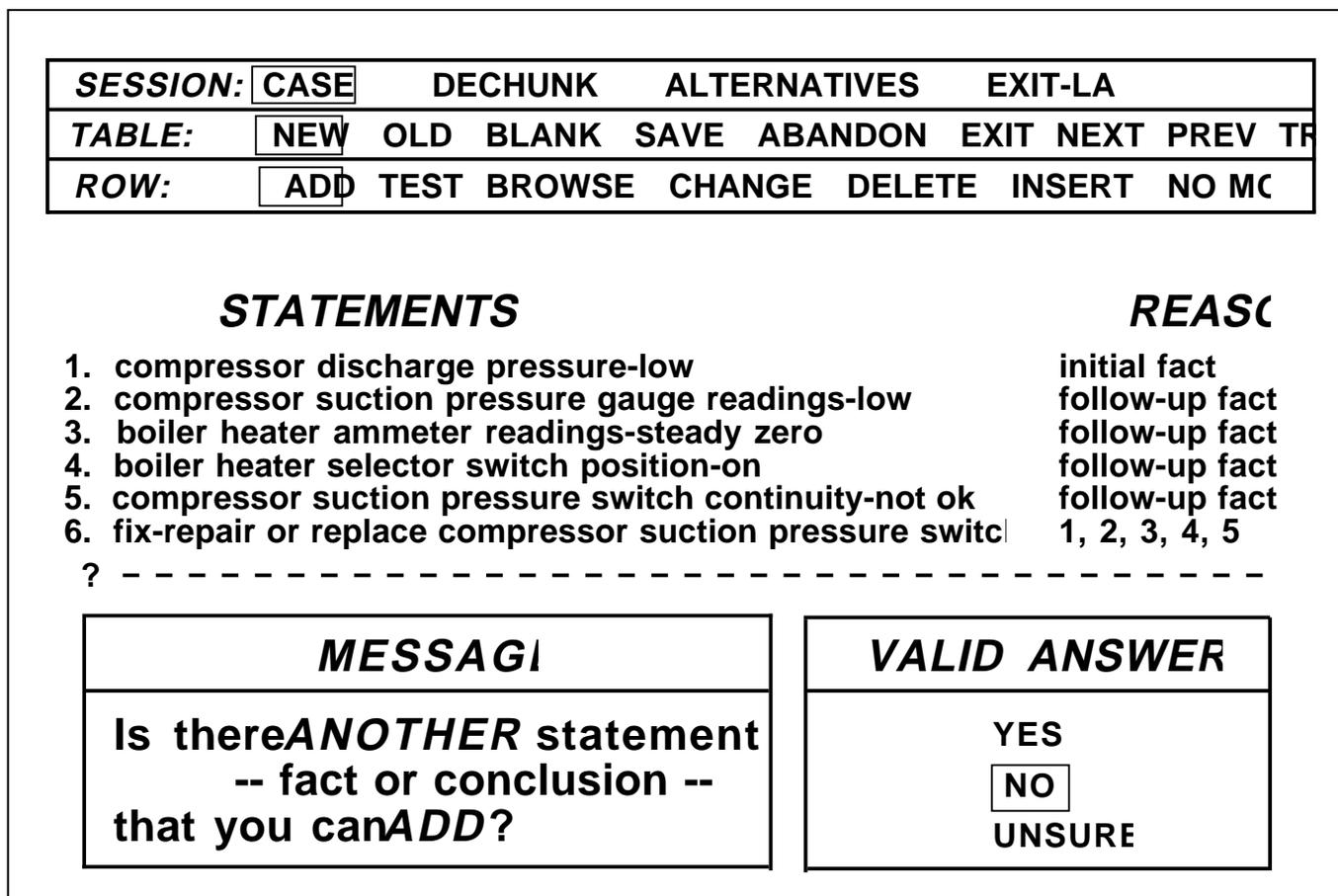


Figure 1. A Case, or Sample Solution, Composed by the Expert.

Observations are listed in the order considered in reaching the final conclusion or recommendation.

The problem to be solved was the diagnosis of a distillation system that converts seawater into fresh water on board a submarine; hence, the expert system was dubbed Still. Ralph Slater, the expert, is an engineering graduate of the United States Coast Guard Academy, with many years of experience troubleshooting various submarine-related devices during his tenure at the Electric Boat Division, the submarine division of General Dynamics.

The Sample Solution Session: Acquire Easily Given Knowledge

This session was used as a case-elicitation session. The expert had no difficulty listing a set of hypothetical observations and then recommending a fix (figure 1). The menus at the top allow the user to access and do operations on any one of the three Laps sessions, a particular table therein, and a given row within this table. In figure 1, the user has

accessed the case session, is on a new table, and is in the process of adding rows. A message box is present to guide the user in what to do next, usually by presenting questions, sometimes augmented by text. The valid-answers box is a list of the acceptable answers. If the the user enters unsure, he is given illustrations at first, then textual tutorial information. Here, the user has finished the sample solution path and has answered no to the question of whether to add another row to the table.

An examination of the list of statements in this figure makes clear the sequential nature of the reasoning process. In some cases, an expert can draw a conclusion from a list of initial facts, facts that are part of the problem description. In this case, only one such fact was given. The others, called *follow-up facts*, emerged in the course of solving the problem.

At this earliest point, the rule generator in Laps would only operate on the last statement in the sample solution path—the only

statement showing a dependency on previous statements in the reasons column—producing the following disturbingly long M.1 rule:

```

if compressor discharge pressure-low
and compressor suction pressure-low
and boiler heater ammeter-steady zero
and boiler heater selector switch position-
on
and compressor suction pressure switch
continuity-not ok
then repair or replace compressor suction
pressure switch.

```

Matters would be far worse if this expert had given us a fully idiosyncratic solution; then the first two lines would read

```

if compressor discharge pressure 55-units-
low
and compressor suction pressure 25-units-
low.

```

An expert system based on rules of this type would be hopeless to build and would produce consultation sessions with questions that would be endless and random. Additional refinement of this sample solution path is desperately needed.

This facts-to-fix sample solution is similar to a path through a flowchart, having no intermediate blocks. Indeed, in our current expert system domain, which centers on reviewing documents, the experts readily give us solved cases in terms of flowcharts stripped of intermediate conclusions. However, in our initial test-bed domain, which we focused on a few years ago, experienced submarine diving officers gave sample solutions with at least some intermediate conclusions.

Essentially, we take the given expert's initial dumps of the solved cases in whatever forms are most convenient for his expression; it is these solved cases, stripped down or not, from which the knowledge engineer needs to extract a model during the next session of Laps. The simplest form of the first session is the following: The expert fills out the statements column first and later fills in the reasons column, using the knowledge engineer's help when necessary. In this way, the expert is not distracted by starting with entering statement one, then switching to giving its justification or reason, then back to statement two, next a reason, and so on. Thus, the expert is able to express himself freely from the start.

The Dechunking Session in Laps: Find Intermediate Conclusions and a Model

Chunking (Laird, Rosenbloom, and Newell 1987) appears to be the activity of economiz-

ing on a reasoning process by directly or indirectly eliminating intermediate conclusions, as in skipping theorems used to prove other theorems. The sample solutions, chunked or compressed as they are habitually by the expert, cry out for some dechunking. Dechunking is our name for the antithesis of chunking.

This second, or knowledge decompilation, session was a joint effort between the knowledge engineer and the expert. This session dechunks to reach steps that the user-expert might have omitted in the original solution path, thus unearthing hidden knowledge. This knowledge could be an explanation, heuristics, or a recurring pattern of reasoning. This additional knowledge could become the source of a high-level model.

Mechanically, a case is dechunked by simply inserting lines into the original list. The question is how to guide these insertions. Let us take another look at the original sample solution path.

The follow-up facts have an ambiguous role because each follow-up fact is both dependent on, and independent of, the previous statement. For example, consider the first follow-up fact, "low suction pressure." The fact that the suction pressure is low comes from direct observation and not from any prior statement. However, the fact that the suction pressure was observed in the first place does depend on the previous statement. We made this dependency explicit by proposing that the expert insert the phrase "check suction pressure." He agreed. This statement follows unequivocally from statement 1; so, 1 can be entered into the reason column. Similar insertions down the list would produce the following rule chain:

```

if compressor discharge pressure-low
then check compressor suction pressure.

if check compressor suction pressure
and compressor suction pressure-low
then check boiler heater ammeter.
Etc.

```

The insertion of still another statement further clarifies the reasoning process, answering the question, Why did the expert choose to check the suction pressure? The answer is that he suspected the cause to be upward in the process stream in the boiler section and that his suspicion could be confirmed by checking the suction pressure. It was confirmed, and this confirmation led him to look still further upstream. Part of this reasoning can be captured by inserting the phrase "suspect upstream boiler section," producing the following chain:

SESSION: CASE	<input type="checkbox"/> DECHUNK	<input type="checkbox"/> ALTERNATIVES	<input type="checkbox"/> EXIT-LA
TABLE:	<input type="checkbox"/> NEW	<input type="checkbox"/> OLD	<input type="checkbox"/> BLANK
ROW:	<input type="checkbox"/> ADD	<input type="checkbox"/> TEST	<input type="checkbox"/> BROWSE
	<input type="checkbox"/> CHANGE	<input type="checkbox"/> DELETE	<input type="checkbox"/> INSERT
	<input type="checkbox"/> NO	<input type="checkbox"/> MC	

STATEMENTS	REASC
1. compressor discharge pressure-low	initial fact
1.a. suspect upstream boiler section [subsystem]	1
1.b. check compressor suction pressure gauge readings	1.a
2. compressor suction pressure gauge readings-low	follow-up fact
2.a. suspect upstream boiler heat control [subsubsystem]	1.b, 2
2.b. check boiler heater ammeter readings	2.a
3. boiler heater ammeter readings-steady zero	follow-up fact
3.a. suspect boiler heater selector switches [component]	2.b, 3
3.b. check boiler heater selector switch position	3.a
4. boiler heater selector switch position-on	follow-up fact
4.a. suspect compressor suction pressure switch [component]	3.b, 4
4.b. check compressor suction switch continuity	4.a
5. compressor suction pressure switch continuity-not ok	follow-up fact
5.a. fault-compressor suction pressure switch failure	4.b, 5
6. fix-repair or replace compressor pressure switch	5.a

MESSAGE
<p>Is there <i>ANOTHER</i> statement -- fact or conclusion -- that you can <i>INSERT</i> anywhere? <i>HINT : Consider one or more of the statements ABOVE a specific insertion point -- as possible reasons for an intermediate conclusion.</i></p>

VALID ANSWER
<p>YES <input type="checkbox"/> NO UNSURE</p>

Figure 2. Dechunked Case Completed by Joint Effort of the Expert and the Knowledge Engineer.

Statement numbers with appended letters indicate inserted statements. The result is a recurring pattern proceeding from a suspected cause to a test, the result of which leads to another suspect narrower in scope.

if compressor discharge pressure-low
 then suspect upstream boiler section.

if suspect upstream boiler section
 then check compressor suction pressure gauge reading.

if check compressor suction pressure gauge reading
 and compressor suction pressure gauge reading-low
 then check boiler heater ammeter reading.
 Etc.

These and similar insertions produced the table in figure 2. The repeating pattern shown

in this table is pictured as the model in figure 3.

The most important pattern that can be found is the model of the overall reasoning process. The table shows an observation (already entered as part of the original sample solution), then, on the basis of this observation, a suspected cause. This suspected cause is then tested, and the result of this test is the next observation (already obtained in the prior session). This process continues, or *iterates*, until the bad component and its fix is determined.

Let *model* refer to an abstract representation or pattern pertaining to problem solving in a domain. The pattern described here is what

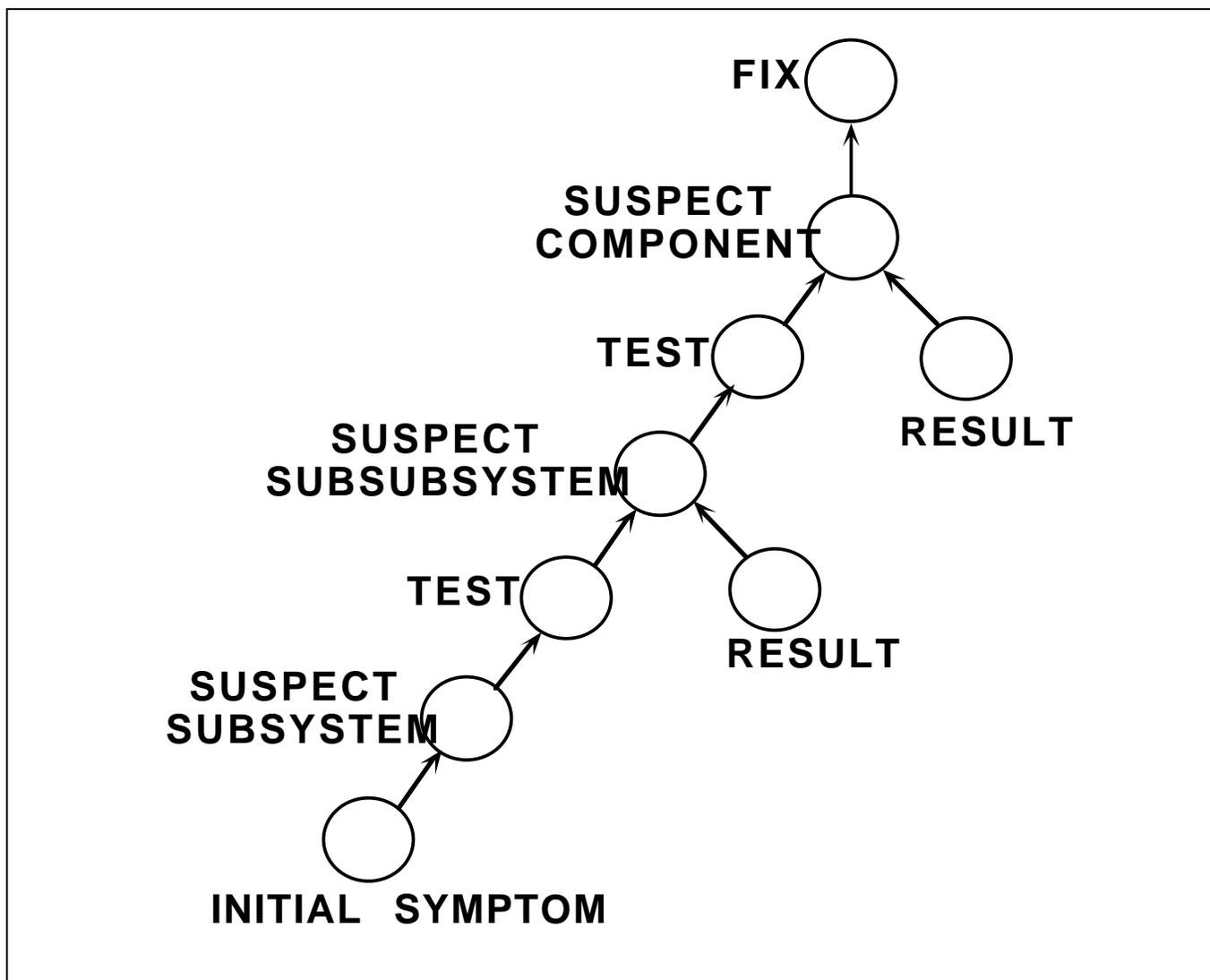


Figure 3. A Domain-Type Model: An Inference Tree of Attributes.

This transdomain tree, extracted from the dechunked or amplified case, offers the expert and the knowledge engineer a bird's-eye view of this domain, along with related domains, providing a manageable basis for this skeleton to be completely fleshed out in the upcoming alternatives tables. (This model is called domain type because it widely applies to diagnostic-type problems.)

we call a *domain-type model*. It is distinguished from a *domain or domain-specific model* in that the latter is specific to a domain. In our example, a domain model would consist of an inference network of such terms as discharge pressure or ammeter reading and would apply only to a particular piece of equipment. However, the previous domain-type model would apply to a wide variety of diagnostic problems. Still more general would be a *metadomain model* or just *metamodel*. A metamodel might contain such terms as node or branch or object or attribute and apply to any domain and, therefore, to more than one domain type (diagnosis, design, and so on).

In statements 1.a., 2.a, 3.a, and 4.a of figure 2, we superimposed in brackets what was later developed (in finding the domain-type headers for the all-important alternatives tables, which are the focal point of the next session of Laps). The insertion shows the reader that within the dechunked solution path are the seeds for reaching a generic hierarchical model (by way of guided induction, as we see later). Thus, no magic act is occurring here because the insertions are the fruits of knowledge decompilation: They are intermediate conclusions drawn from already entered facts or some earlier inserted conclusions (see the hint in the message box in figure 2).

The overarching strategy is to abstract from the model the generic headers of the alternative tables . . .

The Tree Representation: A Picture of the Whole Domain

Figure 3 illustrates a domain-type model. The tree should be read in this manner: An initial symptom implies or suggests a suspect subsystem, a subsystem where the faulty unit might lie. This subsystem implies an appropriate test to perform (a meter reading, a component to be inspected, and so on) to refine one's knowledge. Next, the test, conjoined with its result, implies the suspect sub-subsystem. The cycle continues down to the component or fault level. It should be understood that iteration over the suspect-test-result cycle is necessary at the same level in the hierarchy when a test result fails to confirm a suspected cause, and another must be tried.

Additional Observations

There are three major follow-up observations. The first is regarding hierarchy. As previously observed, the expert, prodded by the knowledge engineer, revealed that he reasoned hierarchically, from a faulty subsystem to a faulty sub-subsystem to a faulty component. The hierarchy not only describes the expert's pattern of thought but also something of the structure of the equipment to be diagnosed—the beginnings of a process model. The use of a hierarchy suggests a convenient way to partition a large domain, allowing systematic piece-by-piece development.

The second observation is regarding the process. Also revealing of the expert's reasoning is the use of the word upstream in statements 1.a and 2.a of figure 2. After some prodding from the knowledge engineer, the expert revealed that he makes use of his awareness or knowledge of the distillation system as a process stream. He then looks in one of three directions: upstream of the immediate source of the symptom; downstream; or at the source itself, in this case, the compressor. He looks upstream first because this search only requires a glance at another gauge. From a practical standpoint, recording this information helped the expert to keep his bearings; so, this level of reasoning was at least implicitly encoded as part of the rule base.

The third observation is regarding dechunking. How much dechunking is enough? When should the dechunking end? The answer is when a sufficient degree of dechunking has been done to effectively partition the domain. The signs of effective partitioning efforts are (1) that in the judgment of the knowledge engineer, it enables the proposed expert system to be built in a thorough and modular manner and that it enables the expert system

to be maintained and (2) that it enables the end user consulting the expert system to find its reasoning or explanation trace to be enlightening. In other words, the answer is when a model has been built that is sufficient for the purposes of constructing an expert system that is maintainable, expandable, and intelligible to the builders, maintainers, and end users of the expert system. One extreme to be avoided would be a largely undechunked rule base, with few intermediate conclusions (spaghetti logic); another would be dechunking to the point of reaching the austere first principles of physics, in the case of the distillation of seawater to fresh water.

Laps: From a Model to a Complete Expert System

The model having been uncovered, the challenge now is to *flesh out*, or expand, the model into a complete expert system. The alternatives table, or third, session of Laps helps the expert to produce a complete set of alternatives to each inferential step in any sample solution that he has already completed. The *inferential*, or conclusion, steps are the statements that follow from previous statements on the list, that is, statements for which statement numbers occur in the reasons column of the sample solution path.

Most significantly, it was in filling out these tables that the expert was most productive, working on his own most of the time by following an easy-to-use completion strategy. Hence, these tables provide the biggest payoff in using Laps and its knowledge extraction techniques.

The overarching strategy is to abstract from the model the generic headers of the alternatives tables, then find some effective method to aid the expert to fill out the rows of the alternatives tables with all acceptable combinations of values so that a complete expert system is the end product. By the end of the session, Laps has helped the user to produce tables that have undergone a number of completeness and consistency checks so that verification and validation are concurrent and preventative, not just an afterthought for curing problems.

Of course, it almost goes without saying that tables are perhaps the most familiar form of representation whose structure is immediately recognizable by any user. As far as possible we, like Twine (1988), exploit the recognition factor of tabular representation. However, where trees, rules, frames, lists, and so on, are useful, we employ or will employ

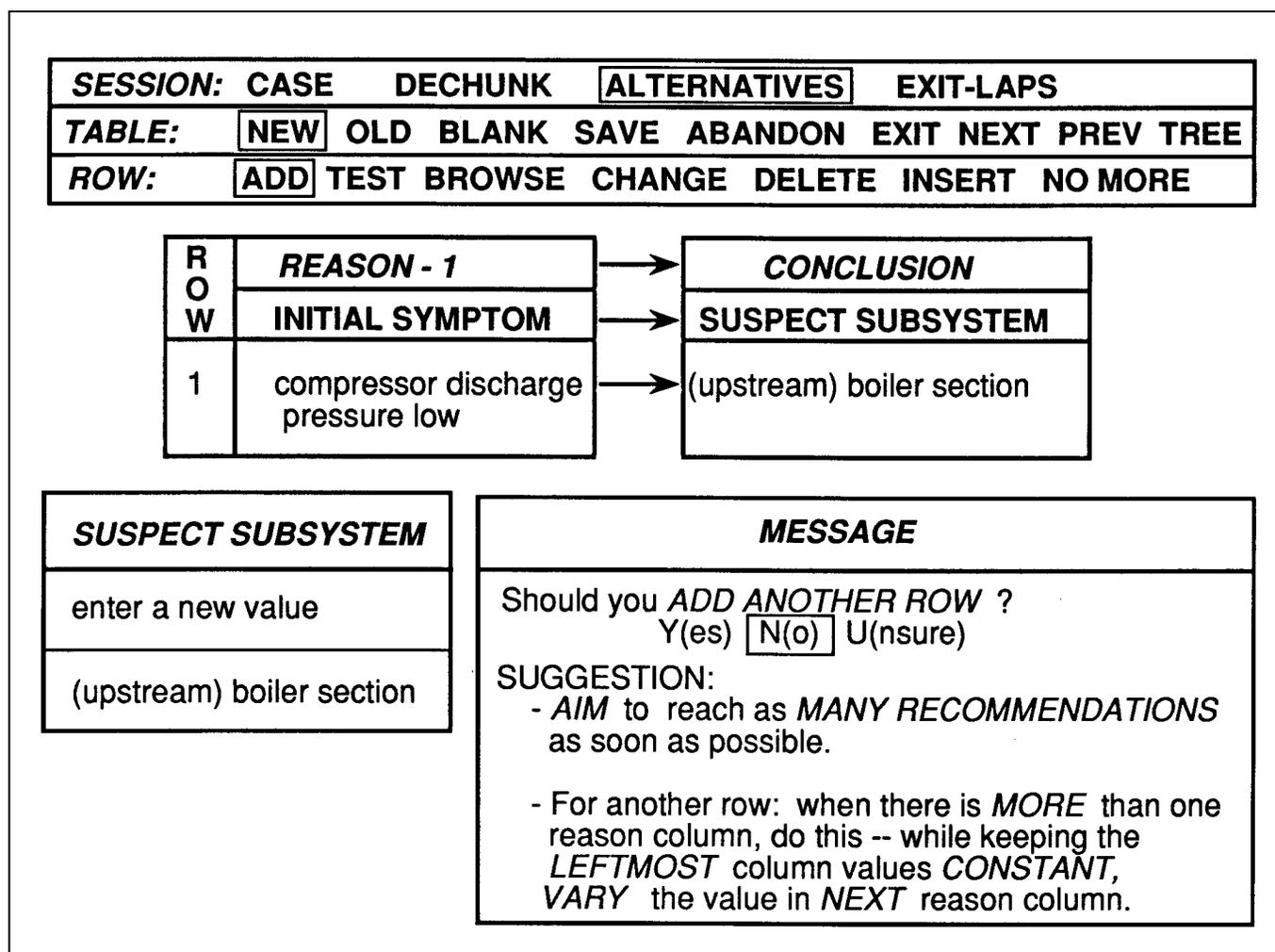


Figure 4. Initial Alternatives Table: Initial Symptom-to-Suspect Subsystem.

This table shows all the possible ways in which reasoning about the distillation system could start. Only one entry is made at this time. Possibly, the next row is not entered until all reasoning from the current row has been mapped out.

these forms of representation. As far as higher forms of representation (Chandrasekaran 1983), we use cases and models (as was already shown).

As we see, the alternatives tables form a chain. In the following subsections, we discuss the alternatives tables within Still.

The First of the Chain of Alternatives Tables: Initial Symptoms as Starters

The headers for the alternatives tables can be plucked from the nodes of the inference tree of attributes (figure 3). The first table is based on the first step of this figure, the step going from initial symptom to suspect subsystem. These entries are found in the table in figure 4.

The screen consists of the alternatives table containing a single row with data for a single

rule, a valid-values box, and a message box. The valid-values box is shown in the lower left part of the screen. Each time a new header is created, a valid-values table or menu is created. At first, the only entry is "enter a new value." The user selects this option and types a phrase into the alternatives table. At this point, the user has just typed "(upstream) boiler section" in the second column. The same phrase is automatically entered in the list of valid values. In future work on this table—better, on any table having this column header—this item can be selected at any time from the valid-values box for entry in the table. This feature is important not only for convenience but also to preserve the consistency of the entries and ensure chaining in the rules to be produced.

The message box prompts the user with

Laps is pushing the user in the direction of the right abstractions by confronting him with the correct domain-independent queries.

some logic or reasoning aids. Using non-AI terminology as much as possible, Laps makes an effort to prompt the expert to concentrate on depth-first development. This development strategy is in accord with his own habit of pushing forward to a solution as soon as possible. Accordingly, Laps urges the expert to reach as many final recommendations as soon as possible. The menu system facilitates this procedure; the user next clicks on NEW next to TABLE at the top of the screen.

The second hint does not apply in this instance because there is only one reason column in this table. Its purpose is to suggest a systematic way to exhaust the combinations when there are at least two reason columns. The dechunking process is expected to reduce the number of reason columns ideally needed to no more than two. Of course, another ounce of automation could be added, such that this advice is smartly given only when there is a minimum of two reason columns.

Setting Up the Tables: An Alternative Strategy to Obtain the Generic Headers

Dechunking efforts during session two could have yielded the headers in the initial alternatives table: initial symptom leading to suspect subsystem. If, however, the dechunking efforts did not yield a model of sufficient abstractive scope, Laps provides the user with a second chance. For this purpose, Laps uses a technique called *guided induction*, that is, induction guided by prompts. As we see later, Laps is pushing the user in the direction of the right abstractions by confronting him with the correct domain-independent queries. According to this scenario, at the outset of session three, the user has entered no headers. However, in this event, row 1 is automatically filled with the first inferred statement of figure 2 together with its reason, producing the following:

Compressor discharge pressure-low Suspect upstream boiler section.

As a suggestion for constructing the headers, the Laps user is presented with the instruction, "Think of many very different values that will appear in a given column. To what common feature (attribute or category) do they pertain? Then make this feature into the header of the column. Try to enter a header that will pertain to only other closely related domains as well as the current domain."

To illustrate, the kind of different values that might have been summoned to mind are "compressor discharge pressure, high or low," inspiring the header, "compressor discharge pressure." With this header, the following modification of the table results:

Headers: Compressor Discharge Pressure ?
Cell Values: low Suspect upstream boiler section.

This header is too restrictive. Among other problems, a plethora of short tables will result that are difficult to track and do not yield the same level of clarity as the more generic headers. Moreover, the generic headers, especially when sketched in the form of the inference tree of attributes (figure 3), immediately allow for dialog among the expert, the knowledge engineer, managers, and those controlling the purse strings. However, domain-specific headers—the components of a domain model—are appealing to specialists whose customary language abounds in jargon such as "boiler heater ammeter reading." Such domain-centered headers tend to deter transdomain dialog. However, in the absence of transdomain headers—the components of a domain-type model—domain-centered ones must be used and can be seen as stepping-stones to the more generic headers.

However, because more widely ranging values came to mind, say, "compressor discharge pressure high," "suction pressure low," and "low production of distillate (very different)," the more encompassing header "initial symptom" was, in fact, abstracted, resulting in the table in figure 4.

Of course, if the user answers unsure to these guided-induction questions, Laps provides a breakdown of the sometimes highly compressed lead-off query into smaller, bite-sized questions. These questions are ones whose answers play the role of subgoals. Illustrations are also available. Finally, all the queries posed by Laps are, in essence, guided-induction types; they do not just indicate requirements to the knowledge engineer and the expert (indicate an object and its attributes) but offer steps to the user to fulfill

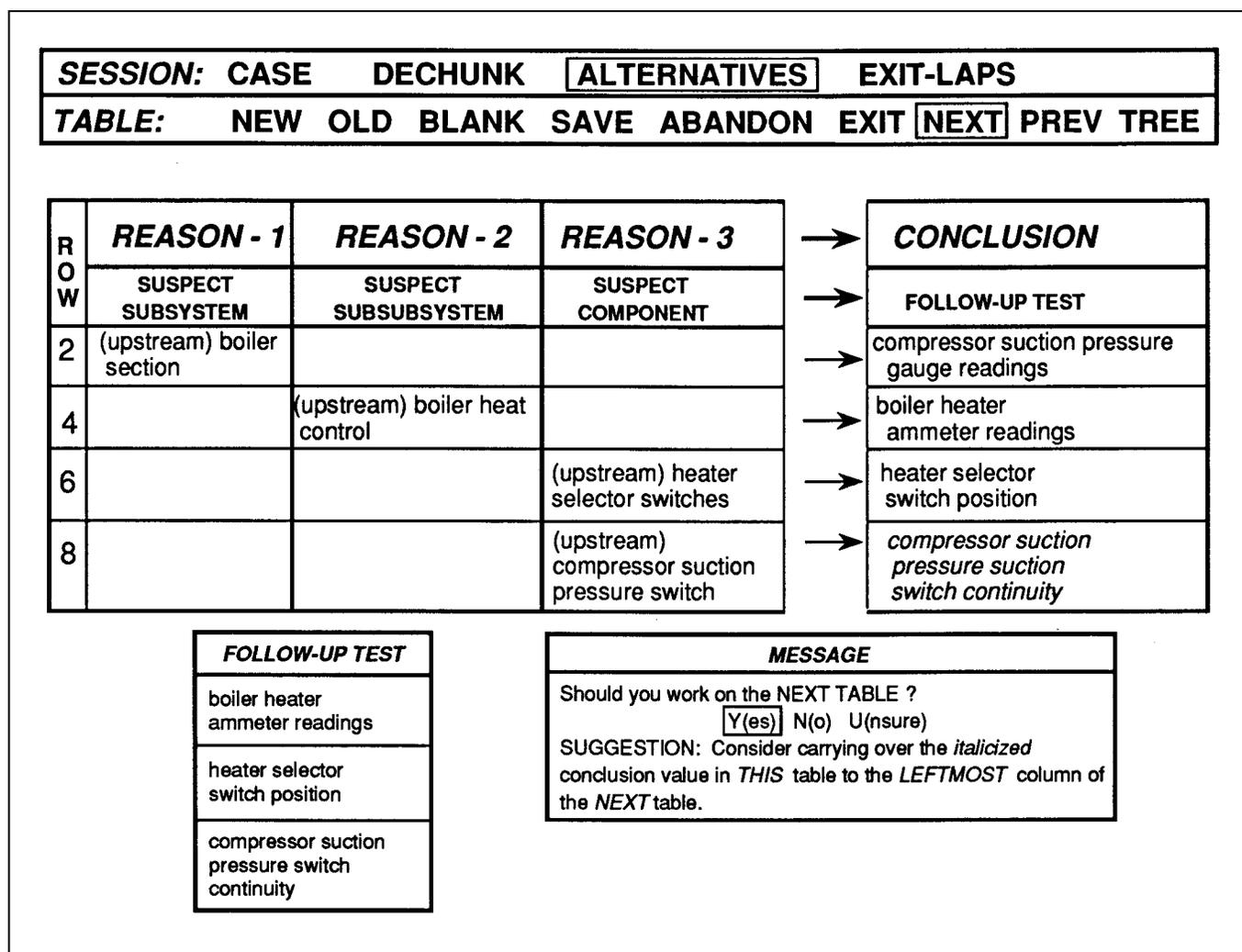


Figure 5. The Second Alternatives Table after Its Completion: Suspect-to-Test Table.

A suspect unit section is brought over from the first or third table by selecting it from the appropriate valid-values box. For each suspect unit, the confirming or follow-up test is cited. At the later point pictured above, the user has just entered the italicized value and is being instructed to carry it over to the next table.

these requirements, especially those of knowledge decompilation and, as we see, knowledge expansion.

The Second of the Chained Tables: From Suspect Section to Test

Following the previous instructions to reach as many (final) recommendations as soon as possible, the expert left the first table; called up a second, blank table; and entered its headers and row 2 (figure 5). It is important to note at this point that the row numbers in column one of any alternatives table are given in order of entry throughout the three tables. Thus, the next rows entered are in table 3, shown in figure 6 as rows 3a, 3b, and 3c.

The table in figure 6 contains the reasoning from a suspect unit (be it a subsystem, sub-subsystem, or component) to the test whose performance will refine one's knowledge of where the fault lies. The headers for this table can be found in the model in figure 3.

The Third in the Chain of Tables: Tests to Lower-Level Suspects

After entering just row 2, the expert then switched to the third or final table (see figure 6). Based on a refinement strategy, this table describes the reasoning from the already performed test and its result to the next lower-level suspect unit of the distillation device. It should be stressed that in this table, the

TABLE: NEW OLD BLANK SAVE ABANDON EXIT NEXT PREV TREE						
ROW	REASON - 1	REASON - 2	CONCLUSION / REASON			CONCLUSION
	FOLLOW-UP TEST	RESULT	SUSPECT SUBSYSTEM	SUSPECT SUBSUBSYSTEM	SUSPECT COMPONENT	FIX
3a	compressor suction pressure gauge readings	low		(upstream) boiler heat control		
3b	compressor suction pressure gauge readings	high	(process) vapor section			
3c	compressor suction pressure gauge readings	ok	(process) vapor section			
5a	boiler heater ammeter readings	steady zero			(upstream) heater selector switches	
5b	boiler heater ammeter readings	steady max			(upstream) compressor suction pressure switch	
5c	boiler heater ammeter readings	ok	(process) vapor section			
7a	heater selector switch position	off			(upstream) heater selector switches	set switch to on
7b	heater selector switch position	on			(upstream) compressor suction pressure switch	
9a	compressor suction pressure switch continuity	not-ok			(upstream) compressor suction pressure switch	repair or replace compressor suction pressure switch
9b	compressor suction pressure switch continuity	ok			(upstream) compressor suction pressure switch isolation valve	

SUSPECT COMPONENT
(upstream) heater selector switches
(upstream) compressor suction pressure switch
(upstream) compressor suction pressure switch isolation valve

MESSAGE
Should you work on the NEXT TABLE ? <input checked="" type="checkbox"/> Y(es) <input type="checkbox"/> N(o) <input type="checkbox"/> U(nsure) SUGGESTION: Consider carrying over the <i>italicized</i> conclusion value in THIS table to the LEFTMOST column of the NEXT table.

Figure 6. The Third Alternatives Table: Tests to Suspects or Faults and Fixes.

A follow-up test is brought over from the second table. Each of the possible results of this test is entered with the corresponding suspect unit. If a suspect is confirmed as the fault at the required level of refinement, the appropriate fix is entered.

expert entered all three of the possible results of the test, completing a rule set of three rows (3a, 3b, and 3c) before returning to the second table to continue down the solution path. The headers of this table can also be found in the model in figure 3.

The exception to the expert's refinement strategy occurs when his effort to refine fails; in this event, he backs up to the higher level he is trying to refine, say, a subsystem, and tries another option at this higher level. In rows 3b, 3c, and 5c, he switches gears, turning from the boiler section or subsystem to the vapor section or subsystem, once the search for a sub-subsystem does not produce positive test results.

Thus, there is no question that Still falls into McDermott's (1988) expert system mode of a cover-and-differentiate system because the expert ultimately develops a complete list of faults. However, Still, as we see from the expert's switching gears, is also a propose-and-revise system, to cite McDermott's other top-level problem-solving mode because the expert switches gears when his proposal of the boiler section or subsystem fails at refinement and then revises his proposal, seeking to explore the vapor section or subsystem. Moreover, it is as if the expert were on his own using depth-first search, along with backtracking, to do refinement-style thinking during his problem-solving efforts.

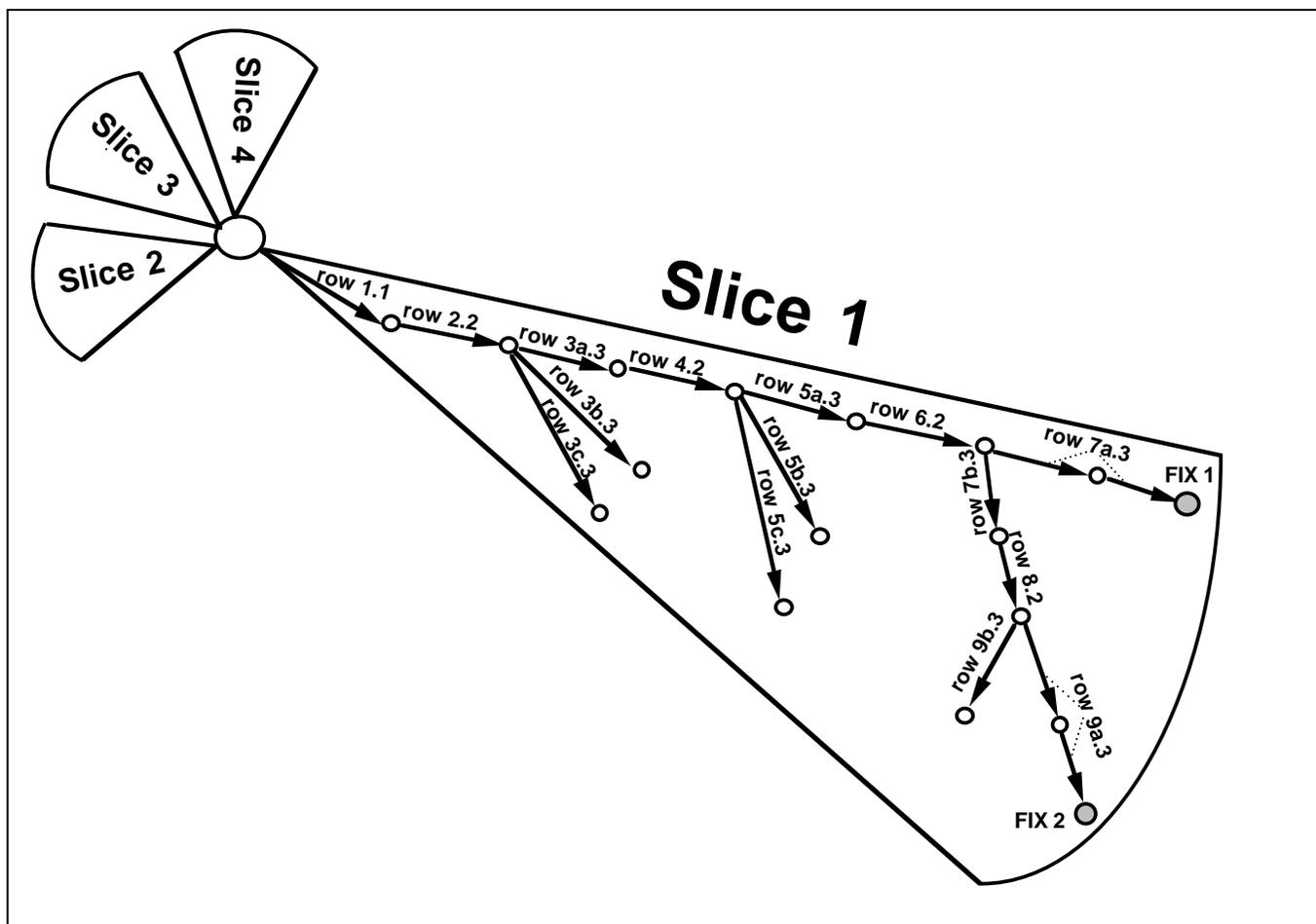


Figure 7. Completion Strategy: A Graphic Review.

The left side of each step or arc label indicates the step's order of entry into the alternatives tables. The right side indicates which of the three tables is used. For example, row 4.2 refers to the fourth row to be entered into any of the tables, but the entry happened to be into table 2. As a second example, row 3a.3 refers to the third row to be entered in any table, being the a or first row in the successively input triple of 3a, 3b, 3c and entered in table 3.

Completion Strategies for a Domain Sector and the Entire Domain

We begin by describing the modified depth-first strategy used to develop the previous domain, then discuss other completion strategies that can be used with Laps.

Completing a Pie Slice: Using the Modified Depth-First Approach and Extending Stubs.

In discussing the completion strategy used by the expert, it is helpful to graphically represent the sequential development shown in the three alternatives tables (figure 7). The sequential development of the three alternatives tables can be characterized as modified depth first, leading to a single pie slice, or *sector*, belonging to the whole pie, or *rule base*. Refer to figure 7. A single pie slice is

one that begins with one initial datum—here, the initial symptom, “compressor discharge pressure low”—and fans out through all the intermediate stops along the way (that is, tests along with their results) to reach every termination point (to a generic line of reasoning) or fix that can occur in the wake of this initial datum.

In figure 7, the first solution path to terminate does so with fix 1, cited in row 7a.3 in figure 6. Here, the dechunked sample solution path (figure 2) is shown as the path leading to fix 2. When the alternatives tables are filled out, the basically depth-first order of entry undergoes some modification. The entries in table 3 include all branches at the decision point, a breadth-wise move making the path stubby or bushy. In addition, when one of the stubs (7a.3) can be completed with only one

*Stubbing
... provides
hooks for
gradual
extension of
the expert
system ...*

more step, it is taken before continuing down the path to fix 2. Again, this order of development is called *modified depth first*.

After the bushy path is completed, the user continues development by completing stub 9b.3. This expansion might involve additional branching points, with associated stubbing, but ultimately, the possibilities are exhausted, and the subtree at 9b is completed. Then the user backs up to 5b and continues in a similar manner. This process is continued until 3b is developed, and a pie slice of the whole decision tree is complete. What results is a complete chain of rule sets, starting with a single initial symptom and ending with all possible faults and fixes that could flow from this symptom. Another research group (the AI group at Honeywell, Inc., in Minnesota) has also found the depth-first development strategy to be more in accord with the expert's habitual flow of reasoning (Cochran 1988). However, the group does not appear to have modified this approach to include breadth-wise thrusts, bringing, as they do, the dividend of stubs to be fleshed out.

Other Uses of Stubbing. *Stubbing*, which provides hooks for gradual extension of the expert system, was used earlier while building the pie slice, but it can also be used in other ways. For example, mathematical computations, or checklists or print routines (pertaining to painting a screen for the end user or presenting the user with messages or warnings during the consultation sessions), can be transformed to fit the kind of inferencing assumed by Laps by using backward chaining from a stub or clause in some rule. However, these non-decision-intensive forms of reasoning are just as well stubbed or left out of the rule base early in the development process. They can be added later after Laps has helped to produce the decision-intensive part of the knowledge base, which is the most important part and the one most difficult to do.

In traditional terms, stubs are like function calls whose body of code can be written later. In closing, the thinking behind Laps is that the decision-intensive core to any expert system should be kept as syntactically simple as possible, containing only propositions or imperative statements readable by lay persons to AI (see the sample M.1 rules previously displayed—with no nested parentheses). However, from some of these statement stubs—"determine such and such" or "print this warning"—can later be hung procedures, and so on.

Completing the Pie: Make Pie Slice after Pie Slice. A pie slice of a domain can be said to have a tip or initializing datum—here, "compressor discharge pressure-low"—just as a tree can be said to have a root. A second pie slice will, of course, be launched with another tip or initializing datum—say, "compressor discharge pressure-high." After one slice from one tip or root is made, other pie slices can be made in the order indicated by the root numbers (figure 7). As one can readily see, the accumulation of pie slices can eventually lead to building the whole pie, no matter how large. The expert is confident that this pie slice strategy can exhaust the domain of Still.

It should be pointed out that this modified depth-first strategy is not an absolute requirement of the Laps tool. Users are free to use any method to complete the tables as long as it works for them. However, what is so attractive about this method is that it enables the expert to track his work and stay aware of all the acceptable combinations and stubs. How would our industrious expert assure himself that he had covered all the initial symptoms and, hence, all the subsequent pie slices? The answer is by following the flow of causal connections in his process model of how the distillation device works. Certainly, our expert would never try to directly build the list of initial symptoms; he would find them by creating more and more pie slices guided by his understanding of the distillation system as a process.

Using and Lifting Simplifying Assumptions. The second strategy for completing a rule base, or the whole pie, is to have the expert narrow the scope of the rule base by making simplifying assumptions. After the rule base is built using these assumptions, the constraints can be lifted one at a time, producing an increasingly comprehensive rule base.

We did some exploratory investigation of this approach while working with submarine diving officers (di Piazza 1988). As the first step in an early sample solution path, two assumptions were frozen, or added, as givens: The ship's speed was constant, and the ship was holding depth. Later, each of these constraints was lifted (speed might not be constant, and the ship might not be holding depth). As a result, the steps in the sample solution path increased although much of the pattern of reasoning used to solve simpler problems, according to our design, was incorporated into the solutions to larger problems.

Furthermore, in 1989, this second development strategy was successfully used in another

er test-bed application of the Laps methodology. This expert system application, which was surnamed Strucon, was somewhat like Sacon (Teknowledge 1986); Strucon was also for selecting the cost-effective and fitting code or utility for doing structural analysis but was more detailed than Sacon and was for submarine hulls. Unlike Sacon, it sought to advise the less experienced code user on how to appropriately use the correctly selected code.

In this domain, the expert, Carl Dyka, could pour out his knowledge only after he froze several assumptions or initial facts, such as the problem pertaining to a static situation describable in nonlinear terms, at our urging. In the wake of these assumptions, he readily delivered a multitude of subtrees that were then fed into Laps alternatives tables. He later returned to lift one assumption at a time, first changing the situation from static to dynamic.

Cell and Row Testing: Necessity and Sufficiency Testing.

In Laps, testing can reach down to the level of the cell and the row. *Necessity testing* determines the appropriateness of the cell entries, and *sufficiency testing* helps determine the appropriateness of the number of columns (attribute-value pairs) for the rows falling under them (di Piazza 1986).

Necessity Testing: Find the Not-Too-Specific Cell Values. To do necessity testing, the question posed in the message box is the following: "In this row, can you replace the cell value under this reason header by another specific value and still reach the same cell value in the conclusion column? If so, then replace the current value under the reason header with one that covers all other specific values that it should cover and no others."

On his own, the diagnostic expert used cell values that were generic in nature and not replaceable by more specific values, such as "compressor discharge pressure-high" (not "compressor discharge pressure-9 lbs/square inch") and "compressor discharge pressure-low" (not "compressor discharge pressure-5 lbs/square inch"). Hence, his cell entries always passed the necessity test. In another domain, however—that of the diving officers—the experts would regularly use values specific to the problem, such as "2 degrees up," instead of "up," which is a value generic to their domain.

Hence, it can be noted that although the diving officers' sample solutions did not pass the necessity test before their specific values were generalized, they were not referring to any explicit model. (Certainly, however, an

implicit model had to be made explicit by using necessity and sufficiency testing as well as dechunking.) However, the diagnostic expert's sample solutions did pass the necessity test (and, as we see, the sufficiency test), and an explicit model was already underlying his expressions. To offer a rather rigorous definition, a model is an abstract representation of the problem-solving process in a domain, prescinding from any cases, that passes necessity and sufficiency testing and is characterized by intermediate conclusions for the sake of intelligibility, as with figure 3 showing the fruits of dechunking. In one way, models can be classified in terms of levels of abstraction, with their attendant advantages and disadvantages, as in the case of domain (or domain-specific) models, domain-type models, and metadomain models (see the discussion of the object-oriented model and the object-to-object tables in *A Structured Table: Its Advantages*).

Sufficiency Testing: Find All the Reason Factors.

Eshelman and McDermott (1986) approximate sufficiency testing for massaging rules but not in a tabular setting or as part of a knowledge-gathering, knowledge-structuring, and knowledge-testing triptych.

The prompt in the message box here, which is another example of a Laps aid for guided or dialectical induction, reads, "In this row can the same reasons lead to a conclusion that can replace the current one? If the answer is yes, then add another column such that regarding the updated row, the answer to the above question is no."

For a simple example of sufficiency testing, we can examine the following assertion: "If the engine's spark is irregular, then replace the points."

Here, prodded by a question about trying to replace the conclusion for the same premises, the expert realizes that in the case where the engine's spark is irregular, the points could need replacement or, alternately, they could just need cleaning. What makes the difference? He advises that if the points are also burned, then they need replacement, whereas if they are just dirty, they only need cleaning.

Again, the diagnostic expert on the distillation device regularly entered materials that passed the test for sufficiency of reasons. Perhaps the reason for his higher degree of rigor on the first try was the use of transdomain, rather than domain-specific, headers. The lesson seems to be that if a genuinely rigorous model is developed before enough necessity and sufficiency testing in the alternatives

. . . the expert was best at supplying data and the knowledge engineer at abstracting the model.

tables, then less need appears to exist for the testing. However, the model, as previously defined, must have already been explicitly or implicitly tested for necessity and sufficiency.

Both a domain model and a domain-type model, as kinds of models, do, of course, meet one of the minimum standards for a model: They pass necessity and sufficiency testing or approach such success, with confidence factors used as a last resort.

The Laps Code Maker: Automatic Encoding of Decision-Intensive Code

In session three, for example (figure 4, row 1), the code generator in Laps would generate the following rule in M.1 syntax:

```
if    initial symptom = compressor discharge
    pressure-low
then suspect subsystem = (upstream) boiler section.
```

The following is an illustration of a chain of rules that the rule maker would create if it is directed to use Clips (figure 6, row 9.a) on the previous tables:

```
(defrule rule-9.a.1
  (follow-up test compressor suction
   pressure switch continuity)
  (result not-ok)
=>
  (assert (suspect component
  (upstream) compressor suction pressure
  switch)))
```

```
(defrule rule-9.a.2
  (suspect component (upstream) compressor
  suction pressure switch)
=>
  (assert (fix repair or replace compressor
  suction pressure switch))) .
```

Certainly, messages or questions to the consultation session user can be affixed to the rules by adding germane columns to the tables.

Of course, Laps code makers can be created for Nexpert or potentially any expert system shell language as far as any of the decision-intensive code or part of the expert system is concerned. Indeed, manual coding for this purpose should largely become a thing of the past. However, the coding for the input-output, user interface, and graphics part of the expert system is still done manually, aided perhaps by object-oriented software utilities such as HyperCard™.

It is important to stress in this context that, as previously shown, rules can capture a model; they need not be looked on as only able to capture heuristics because rules, as a generic syntactic form, can be wrapped

around any semantic material. However, in terms of the much ballyhooed debate over the representational forms of rules and frames, rules seem better fitted as the framework for decompiling cases of problem solving and, hence, as the way to launch an expert system construction effort. Objects and frames seem to be of more assistance as a frame of reference for creating a new user interface and similar items where a history of solved cases is yet to be amassed.

The Roles of the Expert and the Knowledge Engineer: Laps Aids, Not Eliminates, the Knowledge Engineer

The expert could easily generate cases. The knowledge engineer, however, had to take the lead during dechunking and in abstracting column headers for the alternatives tables. After this work was done, the expert became increasingly autonomous in filling out the rows or alternatives. Because the generation of alternatives takes the largest proportion of the time in constructing expert systems, the existence of a tool allowing him to work alone during this stage was helpful. In short, the expert was best at supplying data and the knowledge engineer at abstracting the model. Indeed, to the degree that an expert supplies the domain-type or metadomain model, he is not acting then as a domain expert but as the knowledge engineer. Thus, to express the relationship between the domain expert and the knowledge engineer, the expert indicates the what, and the knowledge engineer points out where in the knowledge map it is to be put. Finally, to the degree that the knowledge engineer relies on the prompts or help messages in Laps, this tool aids him, not replaces him.

The Expert's Preferences: High-Level Attributes, Fewer Tables

Working with the expert creating Still gave us a chance to note certain of his reactions to the various techniques developed on the Laps project. First, the expert preferred the economy of high-level or domain-type attributes such as the subsystem or follow-up test over domain-specific attributes such as compressor suction pressure or ammeter readings. Thus, he favored working with a domain-type model applicable to several domains (figure 3) as opposed to a domain-specific model.

To a degree, the expert must be trained to proceed in this modified depth-first manner.

However, the training clearly went with his problem-solving grain, not against it. After being helped to complete a few rows, he went on to finish a given pie slice on his own. It is clear that this method gave him the important feeling that he was in control and that he was not omitting rows or branches.

Further, he preferred to take the additional step needed to complete the branch to the fix. He also preferred chained tables. Both preferences were accommodated by appending the fix column to the third table rather than creating a separate fault-to-fix table. In rows 7a and 9a in figure 6, the entries in the component column serve a dual purpose: They are conclusions to the statement to their left, and they are reasons for the statements on their right. The advantage of this combined table is that in a glance, the expert clearly senses that he is building a chain of tables and that he can avoid the extra work of moving to another screen to make an entry in the fix table. Moreover, the expert preferred the fusing of tables by combining all three levels, from subsystem to sub-subsystem to component, into a single table.

A Structured Table: Its Advantages

Recently, in line with the expert's penchant for working on as few tables as possible, thought has been given to producing a single composite table, one containing all the reason columns of the previous three tables on the left side of the inference arrow and all the chained conclusion columns on the right. In this setup, he need not shift from table to table to fill in a new row. In its distinctive, continual efforts to provide user-friendly cognitive, as well as logical, aids to reasoning, Laps needs this single-table alternative; at times, users have reported that they have forgotten their line of reasoning right in the middle because it was spread over several tables. This metadomain kind of structured table—its headers can apply to any domain yet can introduce structure or articulation into a domain—has all the advantages of the old or domain-type structured tables previously described as well as a set of new advantages.

Inherited Advantages

This new table inherits the advantages of the previous three alternatives tables: the use of modified depth-first development, hierarchical decomposition of the troubleshooting search techniques, the domain-type model (however, subsystem, sub-subsystem, and so

on, are referred to in the cells under the object header), preventative testing (for necessity and sufficiency and other matters, both online or offline), row insertion and other tabular editing techniques, and so on. Again, it is noteworthy that this metadomain table includes the domain-type table; nothing is lost. Some users might still prefer to segregate the tables into three, although occasional blank rows in the one universal table might serve the same purpose. See figure 8.

New Advantages

The new benefits seem to fall into two categories: representational advantages and advantages for intellectual or knowledge base bookkeeping.

Advantages for Representation. The object-to-object table raises the level of abstraction from the domain-type to the metadomain level, making possible the mechanical and cognitive conveniences of one table. In line with its object-oriented structure, this kind of table also possesses the advantages of object-oriented construction: message passing and a semantic network.

In effect, the object-to-object table does the equivalent of message passing using a table of rows or rules because the object-attribute-value triple on the right side of the row or rule is the triple to be invoked when a fact matches the triple on the left-hand side of the same row. Row or rule four illustrates both the message passing and the semantic network structure; it essentially says, "If the follow-up test on the boiler section is the compressor suction pressure gauge readings, then what is the result of this set of readings?"

Note that in semantic network fashion, what was the left-hand-side value, "compressor suction pressure gauge readings," becomes the object on the right-hand side. This transformation is done to find out the value of a feature of this new object, the feature being the result of this test or set of readings.

A warning might be in order here. Looking at the world of problem solving through the glasses of object-attribute-value triple, an expert does have a handy template for giving explicit organization to his thinking. However, to keep from being a foreign, too-high-level construct, this triple, as with other metadomain constructs such as "if . . . then," should not be used at the outset of the knowledge-acquisition process. They should be used only after obtaining sample solution paths and dechunking them to a suitable point for the end user of the expert system.

ROW	OBJECT	ATTRIBUTE	VALUE	OBJECT	ATTRIBUTE	VALUE	PROCEDURE	COMP
1.			INITIAL FACT	INITIAL SYMPTOM	IDENTITY		QUESTION: WHAT IS INITIAL SYMPTOM?	<input checked="" type="checkbox"/>
2.	INITIAL SYMPTOM	IDENTITY	COMPRESSOR DISCHARGE PRESSURE LOW	SUSPECT SUBSYSTEM	IDENTITY	(UPSTREAM) BOILER SECTION		<input checked="" type="checkbox"/>
3.	SUSPECT SUBSYSTEM	IDENTITY	(UPSTREAM) BOILER SECTION	(UPSTREAM) BOILER SECTION	FOLLOW-UP TEST	COMPRESSOR SUCTION PRESSURE GAUGE READINGS		<input checked="" type="checkbox"/>
4.	(UPSTREAM) BOILER SECTION	FOLLOW-UP TEST	COMPRESSOR SUCTION PRESSURE GAUGE READINGS	COMPRESSOR SUCTION PRESSURE GAUGE READINGS	RESULT		QUESTION: WHAT ARE COMPRESSOR SUCTION PRESSURE GAUGE READINGS?	<input checked="" type="checkbox"/>
5A.	COMPRESSOR SUCTION PRESSURE GAUGE READINGS	RESULT	LOW	SUSPECT SUB-SUBSYSTEM	IDENTITY	(UPSTREAM) BOILER HEATER CONTROL		<input checked="" type="checkbox"/>
5B.	COMPRESSOR SUCTION PRESSURE GAUGE READINGS	RESULT	HIGH	SUSPECT SUBSYSTEM	IDENTITY	(PROCESS) VAPOR SECTION		<input type="checkbox"/>
5C.	COMPRESSOR SUCTION PRESSURE GAUGE READINGS	RESULT	OK	SUSPECT SUBSYSTEM	IDENTITY	(PROCESS) VAPOR SECTION		<input type="checkbox"/>
6.	SUSPECT SUB-SUBSYSTEM	IDENTITY	(UPSTREAM) BOILER HEATER CONTROL	(UPSTREAM) BOILER HEATER CONTROL	FOLLOW-UP TEST	BOILER HEATER AMMETER READINGS		<input type="checkbox"/>

Figure 8. The Object-to-Object Table, Using a Metadomain Model.

This one table, in particular, rows 2 through 6, covers rows 1 through 4 on the previous three alternatives tables. (The earlier tables were designed for the M.1 code maker, and the current one has a Clips rule maker in mind.) It combines the advantages of the series of domain-type alternatives tables with others: one-table accessibility; organization in terms of objects; knowledge base bookkeeping techniques, such as completion check-off boxes; and the procedure column with its calls to many kinds of routines.

Knowledge Base Bookkeeping Advantages and the Procedure Column. Two features fall into the category of intellectual bookkeeping, or *knowledge base bookkeeping*, which is a set of mechanical techniques that help the knowledge base builders to be accurate and complete. These features are single-table accessibility and completion check-off boxes. The multipurpose procedure column is similar in intent.

As far as gaining all the advantages of using just a single table, it is the object-oriented structure of the table that makes this convenience possible with so few columns. It is possible—but probably not desirable—to make a composite out of the three alterna-

tives or attributes tables. However, the resulting table would have 12 columns: 6 reason columns and 6 conclusion, with only “initial symptom” appearing as only a reason column and only “fix” appearing as only a conclusion column. It sounds nightmarish for a user to follow, and left-to-right scrolling would help but would put many columns out of sight. This option will probably be ruled out a priori chiefly because the object-to-object table is a composite with only 7 columns.

To illustrate how the completion box works, the act of clicking on the COMP(letion) box to the right of row 4 places an x in the box, meaning that all its children,

or chained-to rows (5A, 5B, and 5C), have at least been started. A single click can be used to copy this triple: “compressor suction gauge readings,” “result,” and the blank contents of the cell under the VALUE header. These three cell values are then copied to the left-hand side of row 5A. Two more clicks on the same x-filled box inaugurates rows 5B and 5C with the same triple. Unfilled completion boxes cry out to be filled; so, the expert system is less likely to have gaps. Gaps or incompleteness of any kind can shake end user confidence in the expert system as an embodiment of expertise.

Again, in an effort to provide aids to achieve reasoning and its completeness (even more bookkeeping aids, unsavory as the word bookkeeping might be to AI thinkers), we are leaning toward the following strategy: having the same triple printed in the next two rows for each click on the completion box. At the sight of the extra row having a completed left side, the user might be prompted to think of a value for the attribute that he had somehow overlooked. Moreover, there is a need to consciously delete the extra row if it holds no value. This slightly irritating option, however, is better than having the user overlook a row.

To turn to the new column, the procedure column could give various commands to routines to print selection lists, questions, messages, or drawings. All these elements can be expressed in some preprogrammed hierarchy of stock screen types, which we have already sketched. In addition, commands could thus be sent to iteration or calculation routines, and so on. The rationale here is that as far as possible decision-intensive coding should be done declaratively in frame-laden rules or rows that are transparently clear and English-like. Relegate the non-decision-intensive thoughts to attached procedures, which should nonetheless be as close to English as possible, as in the scripts in HyperCard.TM

Implementation: Three Versions of Laps, Current Challenges, and a New Table

In the following subsections, we discuss the three versions of Laps, some current challenges, and the recent unstructured question-to-question table.

The Three Versions of Laps: PC, Mac, and Apollo

The first Laps appeared in the clothing of M.1. The logic of Laps was thereby made clear

The object-to-object table raises the level of abstraction from the domain-type to the metadomain level . . .

in this declarative rule-based language, but the screen management was woefully slow. To gain the speed advantages of C relative to processing input-output and all else, Laps was next implemented in Microsoft C on a personal computer (PC). As a parallel effort for the sake of improvements in the C version, a HyperCard implementation on the MacintoshTM was begun that has proven to be excellent. HyperCard impresses us as the most rapidly programmable experimental tool available to date.

The following is noteworthy: The PC version is altogether system directed or prompt laden, stepping the user through all the hoops, as the prompt or message boxes in figures 4 through 8 indicate. The Mac and Apollo versions, however, are more user directed than system directed, letting the user fend for himself, thus presupposing users who are experienced enough to use the Laps techniques without prompts.¹

Some Current Challenges: Porting and a New Domain

The next move on the migration path of implementations is placing more of Laps on the Apollo for the sake of another application domain. In conjunction with this porting effort, the Laps team is currently engaged in an application project involving the smart review of documents. This current challenge is of the propose-and-revise type; the reviewer, like a teacher or editor correcting essays, is determining if a whole battery of constraints—a smart checklist or, better, tree of matters to check—has been met by the authors of the document.

This project, with hundreds of document types to be covered by the expert system, is large and complex in its own scope, apart from comparison with other domains. It is also large relative to the earlier test-bed domains that we dealt with as researchers, including the diagnostic domain used as our running illustration. This observation is true

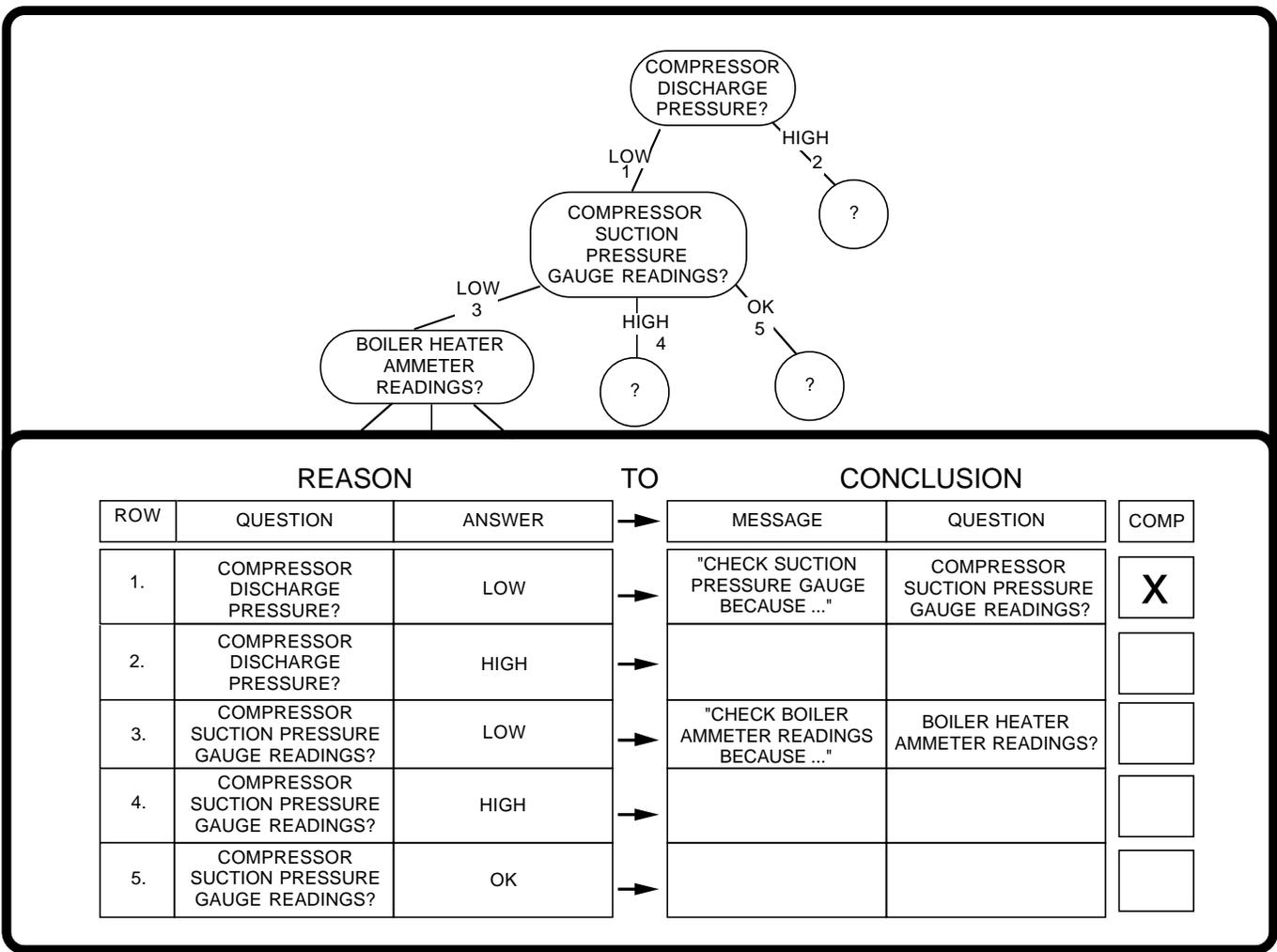


Figure 9. A Decision Tree Converted into a Question-to-Question Table.

This tables illustrates a decision tree of questions that is developed in the modified depth-first manner: The numbers on the arcs indicate the order of entry. Each row of the corresponding question-to-question table has an opening number matching the arc value reproduced in this row.

despite the fact that there are dozens of initial symptoms, each of which can launch a pie slice. Thus far, the ability to stub and freeze assumptions is serving us well in this document-review effort.

The Unstructured Question-to-Question Table: Contains a Tree of Sample Solutions as a Prototype but Cannot Support a Large Expert System

This year, a new kind of Laps table was designed and programmed on the Apollo

system (using Domain C and the utility Dialog) to capture sample solutions—in terms of a series of questions, not statements—and reflect a tree or network. For the sake of continuity, let us use the diagnostic domain to illustrate the question-to-question decision table, although it was designed to satisfy the needs of the experts on the document-review project. These experts have readily produced flowcharts, which, reflecting the character of their domain, happen to be decision intensive; their flowcharts are more like decision trees.

This kind of table, which is metadomain in

character, is purposely unstructured, designed for the easy capture of knowledge. Indeed, we have found this newest table to be the easiest to use of all those we have created and tried thus far. The expert is to transfer a decision tree to a decision table, which can be completed in the modified depth-first manner previously described. The value “compressor suction gauge reading” is reproduced three times by clicking on the completion box. Rows 3, 4, and 5 correspond to arcs 3, 4, and 5. Next, the completion box to the right of row 3 can be clicked on so that row 6 will begin with “boiler heater ammeter reading?” To make a connection, the answers to the questions, if expressed as statements such as “boiler heater ammeter readings-steady zero,” are the statements in the fully chunked kind of sample solution, nonbranching table, or list displayed in figure 1. By having the Laps user pick only one answer to each question for chaining forward and leave the other responses as stubs to be extended later, he is, in effect, listing or making a sample solution path of the kind displayed in figure 1.

The motivation here was to position the expert to give us as big a dump of his knowledge as possible at the outset. The hope is that the expert gets so used to this simple table that he bypasses scribbling down trees or flowcharts on paper. On the issue of graphics and trees versus tables, we have tentatively found that screen-based trees, once they reach any considerable size, can wrap around themselves, acting like a spiral of pie slices covering slices beneath. Zooming, flashy as it is, does not help much if the text is too hard to decipher. Breaking on-screen trees into subtrees, which is a necessity to stop the wrapping, leaves one with the problem of forgetting where the subtree came from or goes to. Too little information—too much white—is shown in trees, whereas one can accuse tables of being just the opposite—too crowded. However, in the long run, whatever works is our choice.

Some amount of dechunking can be done using the message or comment column, thereby keeping the same alternatives table format for all sessions of Laps. The first row message might in full be “check the compressor suction pump pressure because the suspect subsystem is the upstream boiler section, and . . .” As with the formal dechunking session (figure 2), the comments can allude to underlying hierarchical or control reasoning or to whatever can be concluded after each question is assigned its specific answer on the left-hand side.

Some warnings, however, are in order

because of the tempting simplicity of the question-to-question table.

Caveat 1: The dechunking effort is severely limited because no intermediate conclusions are encodable into rules, and no effort is made to find and insert intermediate conclusions between facts and already inserted intermediate conclusions (figure 2, row 1.b).

Caveat 2: Popular among the experts because it is simple to use, this unstructured table is only capable of rapid prototyping, as with the sample solution table. This decision-tree-into-a-table approach will collapse under the weight of a wide-ranging domain of hundreds of types (of documents or whatever) in any domain largely without explicit organization. Organization begets economy of focus. Psychologists tell us that people are limited to focusing on, or remembering about, six to seven numbers at a time. Similarly, people might be limited to focusing on just five to seven divisions of some topic or class before they need to be given two or more intermediate topics or classes—if they are to find relating a multitude of ideas manageable and not too complex.

To be sure, a model or structure of some type is necessary for the sake of constructing the expert system and educating users of the expert system. In such a large domain without the model, the builders will not be able to see where they are going or know where they came from or when they arrive. Consequently, they will be unable to build a plan for constructing the expert system along modified depth-first and pie-slice-after-pie-slice or other lines. Without a model, the consultation session traces appear to the advice seeker as a case of touching or tagging black box after black box: Not much enlightenment for the advice seeker can occur. Moreover, the trace is silent in answer to the question, What is the connection between one domain question and the next one? Further, mere trees lack even the minimal over-viewing advantages of block diagrams, not to mention the superior structuring benefits accruing from object-attribute-value representations.

The Still domain often becomes involved in network thinking; if you will, the pie slices overlap. To show another inadequacy of the question-to-question table, it is helpful to compare it with the object-to-object table regarding the handling of networks.

Caveat 3: The question-to-question table can handle network thinking but in a manner that does not build on meaning or knowledge, whereas the object-to-object approach can handle network thinking in a manner conformable with a knowledge-based system.

Let us turn to an overused but still useful example, and consider the resulting four rules to note the numeric or nonsymbolic way this seductively simple approach traverses to a node in a network from its two parents:

If the animal eats meat, then ask question 1: Does it have spots?

If the animal eats no meat, then ask question 2: Does it have spots?

If the answer to question 1 is yes, then the animal is a leopard.

If the answer to question 2 is yes, then the animal is a giraffe.

If we omitted the numbers for designating the questions, then a logic problem would arise: The same condition leads to two mutually exclusive conclusions because the animal is both a leopard and a giraffe.

Now consider object-laden rules and how they avoid this logic problem—not by the kludge of numbering but by the use of meaningful symbols or knowledge. If the differentiating feature of a carnivore (one value for the current object) is spots, then the identity of the carnivore is a leopard. However, if the differentiating feature of a herbivore (another value for the object) is spots, then the identity of the herbivore is a giraffe. This pair of object-to-object rules shows that in an object-laden table, the different paths to the same node (or attribute)—here, differentiating feature having the same value, spots—can be done without a logic problem and without a kludge if different objects are introduced, taking us beyond the intrinsic limitations of an objectless decision tree or network.

Unavoidable Omissions: The Scope Needed Narrowing

There are three or four omissions, as egregious as they were unavoidable in our work. In particular, they were necessary to narrow the scope of the task to manageable proportions for the expert.

First, a grave but unavoidable omission in Still is that of experience-based control rules. For instance, the reason for the expert's conclusion or determination to focus on the boiler section before the vapor section is not made explicit. In this case, the expert, it was later learned, based his decision on two factors: The boiler section was both more likely and easier to confirm than the vapor section. A quick glance at the suction pressure gauge was all that was necessary.

Why did we not include a rule set on this admittedly critical area of smart control? Simply put, we did not have the time. How-

ever, as it turns out, by setting up the expert to first use his facts and test results to reach faults and fixes, we simplified the task for him. Asking him about his control logic is better done as an afterthought, for example, after a sample solution. We could then ask, Why did you seek this fact before that? If we wait until the dechunked version is on the table, with its explicit model, this sequence query can be refined; why did you seek this test result before that? If we wait until a pie slice is completed, at least more transdomain queries become possible; in a network, why did you inquire about this subsystem (the boiler section) before that (the vapor section)?

Second, for the same kind of compelling reasons, there should have been a separate fault-to-repair-or-replace table. The expert himself recognized that he sometimes replaces a component or piece-part but at other times he repairs it—a perfect instance of insufficient conditions, raising the need for a differentiating factor. However, to save time and prevent distraction while he was learning to use the modified depth-first expansion of his dechunked sample solution path, he was not asked to try to concentrate on this loose end as well. At times, it is necessary to tolerate some chunking and put off dechunking until conditions are ripe.

Third, it should be noted that at times, the expert on Still pushed his search for a fault beyond the component level (the molecule) to the piece-part level (the atom, or indivisible unit for repair or replacement purposes). Sometimes, he did not. When he went for the piece-part, the added expense and time must have been worth the effort. When he settled for pointing the finger at a component as the sufficient fault, further expense and time were not worth the effort. However, again to simplify his task and this presentation, we omitted the task of building these admittedly critical control matters and others (multiple faults, causal or other axiom-based models, and so on) into rule sets. Still, much of an expert's years of sweat-based wisdom is rolled into his control knowledge and process model, as was the case with the expert behind Still.

An Evaluation of Other Knowledge-Acquisition Approaches: Pluses and Minuses

As was pointed out at the outset of this article, a variety of approaches have been used to tackle the difficult problem of knowledge

acquisition. Next is a brief look at three other kinds of approaches and some of their advantages and shortcomings along with indications of how Laps attempts to overcome these shortcomings. Often, the basic shortcoming of these approaches is that some fruitful technique is taken out of context. In other words, the technique is not presented as only a part of an appropriate beginning-to-end sequence or ontogeny; for example, no background to the use of the tool or no follow-up to its use is provided.

Use of Preexisting Models

In this subsection, the term models in phrases such as preexisting models refers to a domain-type model because only a domain-type model can serve as preparation for prying out revelations about the domain-specific model from the expert; if the domain model is at hand at the outset, obviously not much prying is left to be done.

Here, we consider how preexisting models can be taken out of context, isolated from what should come before the nominating of the model. We also consider what should follow the selection or construction of the model: a system for filling out the model.

Preexisting Models: Isolated from the Upstream and Downstream Requirements.

If a model exists that precisely fits the problem solution pattern, then direct use of this model, as Marcus (1987) does, appears to be the best answer. Marcus's article is a thorough illustration, relative to Salt, of what John McDermott calls the propose-and-revise mode of reasoning. After presenting the pertinent model to the expert, the knowledge engineer can urge him expert to assign values to the attributes of the model. Also significant is Musen's (1989) book, which is concerned with automatically generating a series of model-based knowledge-acquisition tools. Musen contributed to the ontogeny of knowledge expression with novel software for handling two matters: assisting knowledge engineers in building a model and then assisting domain specialists in fleshing out the model. Even here, however, there is still the need for additional aids before and after the assignment of the model.

Primarily, before a model can be used, it must be correctly selected. The knowledge engineer has to size up the domain in question to determine which model to use. Asking the expert for solved cases, as is done in the first session of Laps, can aid in this process. Herein, the expert can readily work on his own, as he

seldom can do with the model-based tools. What's more, after a model is selected, it must be fleshed out systematically and completely. The third session of Laps, filling out the alternatives tables, can aid in this process.

To our knowledge, the book by Breuker et al. (1988) contains the most extensive library of models for a wide range of problem-solving tasks. McDermott's (1988) project at Digital Equipment Corporation of creating a taxonomy of models, like Wielinga's, is important. Although there is nothing like a complete catalog of models to choose from, the work of model-oriented knowledge-acquisition researchers can certainly make the process of using Laps easier. What is specifically made easier are the dechunking session and the setting up of column headers at the beginning of the alternatives table session, the stage requiring the greatest involvement of the knowledge engineer. The knowledge engineer can consult any existing catalog of models. If he specifically finds the one not needing any tailoring, he is in luck. If he finds a somewhat similar model, he can think analogically. In short, preexisting models are a valuable supplement to the use of Laps, where at least an analogue exists. Where not even an analogue exists, which is still too often the case, then Laps or some other model-building aid is obviously necessary. Moreover, at the same time, the use of Laps can contribute new members to the library of preexisting models.

Laps: A Domain-Independent Knowledge-Acquisition Tool Throughout.

In this context, it is important to note that Laps is consistently a domain-independent tool but not at the overly abstract level of asking the expert from the opening bell on to produce another rule or describe another object (as in Kee). It clearly is such during the case and dechunking sessions because no explicit domain-relevant model exists, whether domain specific or more abstract, prior to the dechunking or model-unpacking session. Even after the model is obtained and enshrined in the headers of the alternatives tables, the modified depth-first strategy or other techniques is the supradomain tool. Of course, once the headers are displayed for the expert and the world to see, the expert is then working in his own element or domain but seeing through the domain-independent glasses supplied by the Laps queries, urging the modified depth-first and other strategies.

The model-based approach to knowledge acquisition rightly criticizes the overly abstract approach, which dominated the first generation of expert system shells (Chan-

. . . after a model is selected, it must be fleshed out systematically and completely.

drasekaran 1983) from Kee and Art to Nextpert. Urging neophyte knowledge engineers and even domain experts to build rules, frames, trees, and semantic networks from the outset of the interviewing process is a case-oblivious, model-unaware attitude because alone, these representations are too emaciated to offer much guidance and sustain the weight of a growing domain. (Chandrasekaran's article is one of the earliest on the need for model-driven knowledge acquisition.) However, it seems that the modelers are possibly not aware that another domain-independent approach having Laps-like tools (such as case elicitation, dechunking, the modified depth-first method for fleshing out a model) and, possibly, triadic differentiation for finding hierarchies is necessary or helpful to employ both before and after the model is available for use.

Finally, as we have seen, Laps is a domain-independent tool that helps expert and knowledge engineer alike build domain-type models, which are captured in the headers of the alternatives tables, whether they are the early domain-type headers or the later object-attribute-value headers—all reached ontogenetically by massaging cases. Thus, it is then as mistaken as it is commonplace to believe there are domain-independent approaches and domain-dependent approaches to knowledge acquisition and never the twain shall meet.

Automatic-Induction Approaches: No Guided Induction and No Use of Models

Automatic-induction approaches get off to a good start through cases but then unfortunately jump to the creation of the knowledge base with no intervening attempt to create a model. True, such induction from numerous cases does not require a model and can in some cases be a quick way to a knowledge base with small domains. It also allows the expert to work alone for long periods of time. However, such an effort can be tedious (the refrain is "give another case"). This approach, moreover, is notorious for failing to automate *feature selection*, the process, to use Laps terminology, of locating the headers of the alternatives tables. The advocates of automatic induction seem to deny even the need for such features or attributes, not to mention the effort to use some process such as guided induction to develop such headers. As a consequence, the approach almost certainly pro-

duces long, flat, unchained rules that result in colossal maintenance difficulties with large domains. Critically, no model means no bird's-eye view of the domain, as illustrated by the inference tree of attributes (figure 3). Such an overview is needed to enable the builders of the rule base to readily and completely see the skeleton of the domain and, then, to tame the complexity of the domain. In contrast, Laps requires only a small number of well-selected cases at the beginning of the knowledge-acquisition process and encourages the creation of a highly structured model-inspired rule base.

Another induction candidate is on the horizon, in the sense that it starts with concrete cases or situations. In brief, *case-based reasoning*, or *analogical reasoning* over cases, is an induction approach that does metadomain-inspired analogical reasoning over possibly only a few key cases; it does not look for just mere similarities over an army of cases. As its adaptation rules show (Kolodner and Riesbeck 1989), its model is metadomain, although at times a domain-type or domain-specific model can also be used, as in reasoning involving legal precedents. However, the case-based-reasoning approach is not prompt driven like the system-directed version of Laps. This approach might well be trying to accomplish, through its adaptation rules, some of what Laps is aiming to achieve through its prompts for guided induction.

Ironically, although case-based reasoning relies heavily on these metadomain rules, its representatives are often critical of rule-based systems, as if they cannot be frame laden and model inspired and case based, as are Laps-made rules. Still, prospects for various syntheses exist. Why could not a rule-based expert system be updated in part through the massaging of cases by way of adaptation rules? Why could not case-based reasoning and Laps techniques, such as dechunking and other guided-induction prompts, be conjoined? Why not interactively ask the human expert to massage cases—the ones that support the rules and are exceptions thereto (di Piazza 1990), and so on—as well as aim to automate this case analysis work using adaptation rules?

Triadic Differentiation: Good for the Organization of Class Trees but Not Knowledge Decomposition

Triadic differentiation has so little in common with the knowledge disassembling, model-enfleshing approach used by Laps that

it almost defies comparison. Here, a list is constructed of the expected output from the expert system to be built. A subset of three elements from this list is selected, and the expert is asked to pick one and give a factor that will differentiate it from the other two. This process continues with other combinations of items until a complete set of factors is uncovered. Subsequent guided-induction aids for finding features or attributes are happily used.

A rating or repertory grid (Shaw 1988) is invoked to try to use quantitative methods, perhaps in a somewhat artificial or even Procrustean manner. (In this article, Shaw, the originator of repertory grids, illustrates KSS0, the forerunner of Nextra.) Logically, it is more sound to carry sufficiency and necessity testing, along with dechunking and model-building, as far as is needed before using any quantifying in terms of grids, certainty, or other statistical factors. In short, although attractive, measurements cannot be reliable if a less than sufficient qualitative analysis has not already taken place. Rating grids can hide truths that only qualitative analysis can reveal.

Triadic differentiation had its origins in aiding students of the psychologist G. Kelly to differentiate among personal values and then build some generic values hierarchy. Thus, it appears to be more of a tool for a non-expert who does not have a mastery of a domain and could not solve problems in the domain for which he can cite the details; this tool aids the novice in making some distinctions to clarify his confused state of mind.

Not surprisingly, in our experience and that of others, triadic differentiation is not a good icebreaker for the expert's initial expression of his knowledge. We have found that experts are comfortable giving descriptions of the steps taken in solving sample problems of the kind they do day in and day out. In contrast, triadic differentiation does not appear to be a decompilation tool for problem-solving knowledge. Instead, triadic differentiation requires experts to think in a manner that goes against the grain of their problem-solving style of thinking. In trying to use this method as an initial way to express themselves, they feel as if they are playing a strange game.

Let us build on a distinction of McDermott's to show another limitation of the triadic differentiation approach to knowledge acquisition. All the triadic differentiation or repertory-grid knowledge-acquisition tool makers readily admit that their software can handle only the cover-and-differentiate kind of problem, or what Clancey (1985) called *heuristic classification*. The reason is that these

tools require a goal category having a range of values to be nominated at the outset, something like the *L. L. Bean Catalog* from which a buyer can solve the problem of which winter coat to buy.

As discussed earlier, the diagnostic domain described in this article can be considered to fit both the cover-and-differentiate and propose-and-revise modes; just reconsider the sample solution session, where among the earlier steps could be some proposals ("suspect the boiler section") and among the later some results that violate some constraints or tests. Then the fixes or revisions ("the new suspect is the vapor section") are proposed and tested until a solution is found that meets all the constraints. In addition, in the large document-review expert system project on which we are now working, the appraisal process ends with the reviewer indicating his judgment: approved or disapproved. Thus, in lieu of a catalog of predetermined final items driving the search, there is a tree of constraints. In conclusion, Laps and other case-based tools, unlike triadic differentiation tools, are not limited to catalog-selection problems.

However, triadic differentiation could be a powerful aid in selection and classification domains that are less developed or systematized. We are currently working on a task in which the experts for particular problem areas have not set up a rigorous catalog of all the problem types they face. In this situation, where ironically even the experts are novices to some degree, triadic differentiation could be useful in exposing concepts that will aid in classification and selection. In conclusion, it would seem that this technique should be considered as a specialized tool to be used in addition to the techniques described in this article.

Other Validation and Verification Approaches: The Need to Ensure Completeness in Expert Systems

Let us look at the need to find an effective way to ensure completeness in expert systems and at the differences between the cure-only approach and a prevent-then-cure-if-necessary approach. In addition to the question of the best way to build on cases, there is the critical question of how to produce a complete expert system. This problem must be faced when a model is created or even when it exists from the beginning. Much of the inefficiency in building expert systems might be rooted in the unnecessary isolation or serialization of the components of the knowledge

transfer process: the gathering, structuring, encoding, and testing of knowledge.

The need to avoid unnecessary isolation, which considers factors out of context, is especially important with large systems such as R1, Digital's pioneering expert system for configuring their computers. Hindsight is always 20/20, but it is still true that the structural overhaul required in R1 (Soloway 1987) might have been prevented by a systematic development method in which testing for various types of completeness was done during the knowledge-acquisition process. Indeed, the Soloway article is a latter-day recognition that rapid rule-based prototyping, leading to a large and unmaintainable expert system, needs to use techniques of structured or hierarchical programming. These techniques, we add, should not be used after or in lieu of prototyping but should be integrated with rigorous prototyping.

In a related vein, it can be observed that Eva (Stochowitz 1988) is a thorough verification and validation tool but, as implied at the outset of this article, it unfortunately is intended to be used to fix errors only after the knowledge base has largely been gathered. Eva would seem best used as a supplement to a particular knowledge-acquisition approach. This approach should use knowledge decompilation methods and attempt to prevent logic and structure errors in the first place, leaving few errors for a verification tool to catch. For example, Laps protects its expert system builders from the logic error of circular reasoning by building on sample solution paths, which by their nature cannot be circular. Because sample solutions are the basis for later structuring efforts, it is clear that these efforts profit from the same protection. As another example, let us ask, Why not pose simple queries to the expert system builders, doing sufficient testing before each rule is input, as Laps does? Why not follow this pattern in addition to what Eva does—wait for another rule to be input that has the same conditions and a different conclusion?

Summary: Cases to Models to Complete Expert Systems

Two kinds of summaries can be offered. One pertains to Laps in general as it can be applied to any application. The other pertains to Laps as it is used in the Still expert system. Finally, there are some specialized final observations, such as where the productivity gains lie in using Laps and how to ascend beyond the productivity plateau on which knowledge engineering is stuck.

Summary: Laps in General

Laps has been used to elicit cases from an expert in equipment diagnosis (and in other domains). The case-centered session was followed by a second in which a case was dechunked, producing a refined sequence of steps leading to the final conclusion. A model of the expert's reasoning was abstracted from the dechunked solution path. The model, in turn, was used to guide the generation of alternative paths until all paths issuing from a single initial symptom had been completed. This process resulted in a pie slice of the complete decision tree and suggests a general strategy for building large knowledge bases by constructing them pie slice by pie slice. Other procedures for taming a large domain are lifting assumptions and stubbing to the user.

After any session, the Laps rule maker can produce rules in M.1, Clips, and potentially other languages as background for a consultation session. However, these sessions are far less frequent because the knowledge entered has already been structured and tested.

In 1990, two new tables were created—one structured, the other unstructured. The object-to-object kind of structured table makes one-table accessibility and other book-keeping benefits possible, and it retains the advantages of the series of domain-type tables. The question-to-question kind of unstructured table assists the expert—more than any other premodel format—in dumping a complete pie slice on his own. However, for the sake of the enlightenment of advice seekers or the taming of a large, complex domain, it is clearly necessary to use tools to extract and en flesh a model of the domain in a complete but trackable, hence, tractable, manner.

Summary: Laps and Still

In the domain of diagnosing a submarine distillation system, cases were elicited using the first session of Laps. The expert had no difficulty listing a set of hypothetical observations and then recommending a fix. A typical case was selected, and the second session was used to abstract a model of the expert's reasoning from this case. The challenge was to build on this case by supplying intermediate conclusions, citing the reasons, and supplying certain linking statements, all necessary steps in producing rule chaining. This task required a joint effort between the knowledge engineer and the expert.

The result was a dechunked solution path showing a model, or pattern of reasoning, that proceeded from an observation to a sus-

pected cause to a test of the cause to another observation. The expert's reasoning proceeded down through a hierarchy from subsystem to component. Guiding the expert's selection of suspected causes was the consideration of the system as a process stream. Specifically, he looked in one of three directions: upstream, downstream, or at the source of an observation, with the order of consideration influenced by factors such as likelihood of the suspect or ease of testing.

As this article argued, after the model was abstracted, the task was to systematically extract all possible value assignments of this model. In the third session, the model was used to determine column headers for three alternatives tables. The first table showed the initial symptom leading to a suspected subsystem. The second table showed suspected subsystems, sub-subsystems, or components leading to appropriate tests. The third table showed tests and observations leading to other suspects or determined faults along with their fixes. This table listed rows for other possible observations for a given test; so, the contents of the three tables could be said to contain a bushy slice through the expert's decision tree. The stubs of the bushy slice were completed, and a pie slice of the complete decision tree resulted.

Final Reflections: The Next Generation of Tools for Building Expert Systems

With its modified depth-first approach to developing the tables, its necessity and sufficiency testing of the rows or rules, its guided induction and dechunking techniques to arrive at a model of attributes, and its aids to logical thinking and knowledge base book-keeping, Laps is much more than a knowledge-acquisition spreadsheet. Moreover, at every turn in Laps, questions help the user to test the rule base for consistency and completeness, and there are other knowledge-prodding guided-induction questions. Accordingly, the presence of these queries supports the thesis underlying Laps that the following knowledge-massaging operations should be interwoven into one fabric: the gathering, structuring, testing, and encoding of expert knowledge.

The research behind Laps has helped to clarify the roles of the domain expert and knowledge engineer. The domain expert could easily generate cases, but the knowledge engineer—on his own or after consulting a library of models—took the lead in abstracting the model and setting up the alternatives tables for the expert to complete.

Laps attempts to prevent errors while the knowledge base is being built rather than catch . . . errors . . . after they have occurred.

Then the expert could work alone for long periods of time generating the alternatives.

Significantly, the great increase in productivity through using Laps comes in the expert's filling out the alternatives tables, which make up the knowledge base and, of course, can have rows or rules running into the hundreds or more. However, without the modified depth-first strategy, there would have been no easy-to-track method for completing a pie slice of the domain. In addition, without the model, there would have been no skeleton to flesh out or, in terms of the alternatives tables, no headers to guide the entries into the rows. Moreover, without the cases or sample solutions, there would not have been the bases for doing the subsequent knowledge decompilation and the fleshing out of the model.

In addition, this strategy has implications for verification and validation in that Laps attempts to prevent errors while the knowledge base is being built rather than catch formal or consistency errors, as Eva does, only after they have occurred. The other approaches to validation and verification and knowledge acquisition point to a piece of the solution to the difficult knowledge-acquisition problem, say, the organizational value of a model. However, they treat this piece in isolation, not dealing with, say, the need for a case-based background to find the appropriate model or the need for a user-manageable system for fleshing out the model using, for example, the modified depth-first approach. Thus, without exception, the techniques of other approaches have substantial contributions to make, but unfortunately, they are taken out of a fruitful ontogenetic context.

Let one of our last points be about software engineering. Brooks (1987) considers the essence of software engineering to be aids for

unraveling, in its fullness, the logic behind large software programs as used in industry. For him, graphic aids and all other nonreasoning aids are peripherals, or incidentals, for the purposes of software engineering. He calls for the creation of techniques for doing the logical analysis needed for structured prototyping, these techniques, in his eyes, being essential to the task of improving software engineering. With its many logic aids, Laps is a response to the need that Brooks perceives.

To offer a retrospective look at the history of thought, at midcentury, philosophers Ludwig Wittgenstein, J. L. Austin, and Edmund Husserl were doing conceptual analysis, not of concepts such as diagnosis and design but of background concepts such as knowledge, belief, and science. To plunge further into the philosophical past, Alfred North Whitehead opined that all of philosophy is but footnotes to Plato. To paraphrase him, much of AI's disputes over models, levels of abstraction, and such are but footnotes to Plato's myth of the divided line in the *Republic*, where he discusses and relates observation, useful belief, and first principles of what we would term domain-type and metadomain models.

In conclusion, expert systems must climb beyond their current productivity plateau, where almost all commercial knowledge engineering tools are still first generation: They are bereft of a knowledge-acquisition front end and automatic encoding of any kind. Thus, in a real sense, these tools are empty shells. We hope Laps and other knowledge-acquisition software will be of help. Perhaps, some grand synthesis, which Steels (1990) inaugurates, will be the final, long-range answer but a synthesis that also does justice to the ontogeny of knowledge acquisition and the integration of case elicitation, knowledge structuring, and testing. Such a synthesis should also put the expert in a position to do a great deal of methodical inputting of his knowledge for the sake of productivity gains. However, improvement, if not a full solution, must come soon, if scenario 1 pictured by Weitz (1990), where expert systems do not have a widespread impact during the next 10 years, is not to come to pass. Finally, we believe that progress beyond the current plateau, including attempts at synthesis, will evolve from the frequent interaction between tool making and industrial-commercial expert system praxis or application. To paraphrase Immanuel Kant, tools without continual praxis are empty; praxis without improved tools is blind.

Acknowledgments

We are grateful to Richard Miller for his often insightful observations on the Laps project over the years and to Richard Abate for his years of help with funding and encouragement. Special thanks to Dr. Lundy Lewis for being a voluntary reader and gadfly and to Drs. Carol Davis and Tom Skrmetti for being the friendly model-based opposition. Many thanks to Dorothy Minior and Joseph Leone for precision work and suggestions. Thanks, too, to Stuart Brown and Lee Anderson for their lively input. Finally, thanks to other colleagues at General Dynamics who have directly or indirectly helped the Laps project over the years.

Notes

1. Because the various programmed versions of Laps have various functions for various purposes, the figures in this article represent an amalgam of these functions along with some readability and resolution improvements, some offline prompts, and additional features still in the design stage. For instance, help messages, which a click on any rectangle on the screen can generate, are currently only available on the Apollo version. These instructions assist the user who requests them, replacing the continually on-screen message box in the personal computer (PC) version. Moreover, the Apollo (the only version in color) and the Mac versions have menu systems and completion boxes, unlike the PC version. Only the PC version currently has valid-values boxes. Only the latest Apollo version has the question-to-question table, but the object-to-object version is in the advanced design stage. Finally, the oldest or PC version alone has the sample solution session along with the dechunking session.

REFERENCES

- Becker, B. 1988. Towards a Case-Oriented Concept of Knowledge Acquisition. In Proceedings of the Second European Knowledge-Acquisition Workshop, 9:1-12. Bonn: German Institute for Mathematics and Data Processing.
- Boose, J.; Bradshaw, J. M.; and Shema, D. B. 1988. Recent Progress in Aquinas: A Knowledge-Acquisition Tool. In Proceedings of the Second European Knowledge-Acquisition Workshop, 2:1-15. Bonn: German Institute for Mathematics and Data Processing.
- Breuker, J.; Wielinga, B.; et al. 1987. Model-Driven Knowledge Acquisition: Interpretation Models, Memorandum, 87, KADS Project, Dept. of Computer Science, Univ. of Amsterdam..
- Brooks, F. P. 1987. No Silver Bullet: Essence and Accidents of Software Engineering. *Computer* 5(4): 10-18.
- Chandrasekaran, B. 1983. Towards a Taxonomy of Problem-Solving Types. *AI Magazine* 4(1): 9-17.

Clancey, W. 1985. Heuristic Classification. *Artificial Intelligence* 27(3): 289–350.

Cochran, E. L. 1988. KLAMshell: A Domain-Specific Knowledge-Acquisition Shell. Honeywell Corp., Golden Valley, Minn.

di Piazza, J. S. 1990. Interweaving Knowledge Extracting, Organizing, and Evaluating: A Concrete Design for Preventing Logic and Structure Bugs While Interviewing Experts. *Journal of Automated Reasoning*. Forthcoming.

di Piazza, J. S. 1988. Laps: An Assistant for Debriefing Experts. In Proceedings of the Second European Knowledge-Acquisition Workshop, 14:1–17. Bonn: German Institute for Mathematics and Data Processing.

di Piazza, J. S. 1986. An Interactive Knowledge-Acquisition and Verification System: A Study in Logic Requirements. Presented at the Workshop for High-Level Tools for Knowledge-Based Systems, Ohio State University.

Eshelman, L., and McDermott, J. 1986. MOLE: A Knowledge-Acquisition Tool That Uses Its Head. In Proceedings of the Fifth National Conference on Artificial Intelligence, 440–445. Menlo Park, Calif.: American Association for Artificial Intelligence.

Gaines, B. 1988. Second-Generation Knowledge-Acquisition Systems. In Proceedings of the Second European Knowledge-Acquisition Workshop, 17:1–14. Bonn: German Institute for Mathematics and Data Processing.

Kolodner, J., and Riesbeck, C. 1989. Case-Based Reasoning. Presented as a tutorial at the Eleventh International Joint Conference on Artificial Intelligence.

Laird, J.; Rosenbloom, P.; and Newell, A. 1986. *Universal Subgoaling and Chunking*. Boston: Kluwer Academic.

McDermott, J. 1988. Preliminary Steps toward a Taxonomy of Problem-Solving Methods. In *Automating Knowledge Acquisition for Expert Systems*, ed. S. Marcus, 225–256. Boston: Kluwer Academic.

Marcus, S., et al. 1987. VT: An Elevator Configurer That Uses Knowledge-Based Backtracking. *AI Magazine* (9)4 : 39–56.

Musen, M. 1989. *Automatic Generation of Model-Based Knowledge-Acquisition Tools*. London: Pitman.

Parsaye, A. 1987. *Auto-Intelligence User's Manual*. Los Angeles: IntelligenceWare.

Shaw, M. 1988. Problems of Validation in Knowledge Acquisition Using Multiple Experts. In Proceedings of the Second European Knowledge-Acquisition Workshop, 5:1–15. Bonn: German Institute for Mathematics and Data Processing.

Soloway, E., et al. 1987. Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY [sic] Large Rule Base. In Proceedings of the Sixth National Conference on Artificial Intelligence, 422–426. Menlo Park, Calif.: American Association for Artificial Intelligence.

Steels, L. 1990. Components of Expertise. *AI Magazine* 11(2): 28–49.

Stochowitz, R., et al. 1988. The Verification and

Validation of Expert Systems. Presented as a tutorial at the Eighth National Conference on Artificial Intelligence.

Teknowledge. 1986. Mini-SACON. In M.1: Sample Knowledge Systems. Tutorial.

Twine, S. 1988. From Information Analysis to Knowledge Acquisition. In Proceedings of the Second European Knowledge-Acquisition Workshop, 6:1–15. Bonn: German Institute for Mathematics and Data Processing.

Weitz, R. 1990. Technology, Work, and the Organization: The Impact of Expert Systems. *AI Magazine* 11(2): 50–60.



Joseph S. di Piazza has been a principal engineer at the Electric Boat Division of the General Dynamics Corporation in Groton, Connecticut, since 1984. He has been the principal investigator on the Laps knowledge-acquisition project since its inception in 1986.

From 1969 to 1984, he was a professor at Southern Connecticut State University in New Haven, where he taught philosophy and computer science. He has a Ph.D. in philosophy, specializing in epistemology and the philosophy of science, from the University of Toronto. He also has an M.S. in computer science from the Polytechnic Institute of New York. His doctoral dissertation was on the Gestalt-Kantian method used by Kurt Goldstein, M.D., to diagnose brain-damaged persons. Reflecting his two backgrounds, his interests and publications are in the areas of applied epistemology and the development of expert systems. He is a member of the American Association for Artificial Intelligence and the Computer Society of the Institute of Electrical and Electronic Engineers.



Frederick A. Helsabeck has been a computer systems engineer in the Data Systems Division at General Dynamics Corporation since 1982, where he has worked with knowledge-acquisition techniques, expert systems, and intelligent tutoring systems. He was a teacher for 20 years prior to his employment at General Dynamics. He has a B.S. (1959) in chemistry from Lynchburg College and a Ph.D. in psychology from Michigan State University. His Ph.D. dissertation was an analysis of difficulties in syllogistic reasoning. He is a member of the American Association for Artificial Intelligence and the Computer Society of the Institute of Electrical and Electronic Engineers.

at General Dynamics. He has a B.S. (1959) in chemistry from Lynchburg College and a Ph.D. in psychology from Michigan State University. His Ph.D. dissertation was an analysis of difficulties in syllogistic reasoning. He is a member of the American Association for Artificial Intelligence and the Computer Society of the Institute of Electrical and Electronic Engineers.