Expert Systems Tools and Techniques: Past, Present and Future

Peter Szolovits

My task in this article is to try to bring a little clarity to the discussion of tools for expert system construction. This is difficult mainly because the topic is still very much in flux, with new ideas and new products coming onto the stage constantly, and because our basis of practical experience is still small enough that it is not easy to separate the lasting ideas from the overblown publicity surrounding the commercialization of artificial intelligence technology. My approach here will be to give a brief overview of the current state of expert systems and expert system construction tools, and then to discuss the directions of research in applied artificial intelligence, trying to indicate the likely future developments of both expert systems and the tools built to support their development.

This commercialization of AI, mainly in the form of expert systems, is becoming a large market, with current revenues in the $100 million range, projected to reach around $800 million by 1990.[1] Despite the large dollar figures, the overall impression given by the field is that only a very small number of projects has actually reached the stage of routine commercial application. The forecast quoted above goes on to suggest that the market

[1]These figures, as well as the ones below on tool product sales, are from a presentation by Alex Jacobson, President of Inference Corp., at the 1985 IJCAI meeting in Los Angeles. As an officer of one of the companies whose future depends on the commercial success of this field, I would guess the numbers are more likely to be optimistic than pessimistic estimates.

for expert systems tools is now roughly a third the size of the total expert systems market, projected to be about half its size (i.e., $400 million) by 1990. This is a peculiar situation. In the commercial data processing world, it would correspond to the almost unimaginable scenario in which nearly as many people would be building COBOL compilers as writing data processing applications. This disproportionate emphasis on tools may have several explanations; only time will tell which, if any, are correct:

- Many commercial systems destined for actual use are in various stages of development, but because the big push to build such systems began only in the past couple of years, it is too early to see the finished products in widespread use. Perhaps more systems are even in routine use but being kept secret for possible competitive advantage.
- The overwhelming interest of inexperienced potential expert systems builders to "get on the bandwagon" both demands relatively simple-to-use tools to help them get started and provides large commercial incentives to more experienced tool-builders to cash in on the wave of interest.
- Many researchers who have built at least experimentally successful expert systems believe that their underlying approach is universal, so they are motivated to generalize and export that technology for use by others.
- Progress in the general field of artificial intelligence (especially in knowledge representation, inference, meta-level reasoning, knowledge acquisition and learning) and in software engineering (especially the use of graphical interfaces, powerful single-user workstations, and interactive editing/debugging tools) have really made it possible to build much improved computer language/environment combinations to support the kind of programming involved in creating expert systems.

Whatever the real situation is, the disproportionate effort going into tool construction should serve as a caveat; much is still poorly understood.

As a result of the huge volume of activity in expert systems tool construction, it is impossible for me to survey here even the most well-known of the large system-building tools, much less to try my hand at bringing some order to the plethora of small PC-based tools and languages being offered for general use. Typical of the many attempts to survey this field are articles such as "An Evaluation of Five Commercial Expert Systems Tools:

Advisor	Hulk II	Parys
Apes	Hypnotist	Personal Consultant
Cambridge Lisp 68000	Inference Manager	Prolog2
ESP Advisor	Insight2	Reveal
ExperLisp	KES	Rulemaster
ExperOps5	Lightyear	Savoir
Expert4	LPA Micro-Prolog	Superfile ACLS
Expert Ease	M1	Tess
Ex-Tran 7	Micro Expert	Timm
Franz Lisp	Micro-Synthese	Xi
Golden Common Lisp	Natpack	Zxpert

Figure 1. A large (but incomplete) list of expert system development tool and languages for microcomputers, from *Expert Systems User*, Vol. 1, No. 4 July/August 1985.

KEE, SRL$^+$, S.1, ART, DUCK,"[2] which investigates five of the best-know large systems, and "Overview of Small Expert Systems Building Tools,' which lists the set of programs and tools shown in Figure 1.

Instead of a comprehensive *Consumer Reports* style survey, therefore I would like to focus more on the underlying concepts and technology of ex pert systems and tools. I want to review the basic ideas of what an exper *system* is and to relate the notion of tools to the more classical computer sci ence topic of language design. I will then review two of the simplest "pure' strategies for building expert systems—the use of rule chaining and of fram matching—which provide the bases for most of today's tools. Next, I wil turn to a discussion of a set of criteria according to which various exper system tools can be evaluated. Then I will describe current efforts to fin underlying commonalities among different tasks and problem-solving meth ods and suggest directions for expert system tools if these efforts succeed Finally, I conclude with some experimental ideas for including more sophis ticated reasoning about causality, constraints, etc. into expert systems an speculate on the effect such ideas will have on future tool developments.

Expert Systems

A conceptual revolution during the late 1960's suggested that the wa to make "artificial intelligence" programs smart was not simply to giv

[2]This appeared in the August 1985 issue of *The Artificial Intelligence Report*, Artificia Intelligence Publications, 3600 West Bayshore Road, Palo Alto, CA 94303.

them Sherlock Holmes' powers of deduction but instead to pack them full of a very large amount of knowledge specific to problem-solving in their intended domain of application. Based on this insight, the first AI programs of real use to people other than their developers were built for doing symbolic mathematics [Mathlab 83] and determining chemical structures [Feigenbaum, et al. 71]. In both these cases, the systems could do as well at some tasks as human experts. In addition, both systems were organized around a few relatively simple but powerful central ideas (several representations for algebraic expressions, a non-redundant constrained generator of possible chemical structures) augmented by vast numbers of special bits of knowledge most useful in simple combinations (e.g., how to do integration by parts, what features of a mass spectrum indicate the presence of CH_3 groups).

The terms "knowledge-based systems" and "expert systems" were invented to describe programs such as these and their intellectual successors, to emphasize their reliance on much knowledge embedded in the program and on their utilization of expert human methods in attempting to achieve expert-level performance.[3] The boundaries between expert systems and others are, of course, not sharply drawn. After all, a very conventional payroll program may also encode vast numbers of intricate minutiae of the tax code, and a statistical classification program may perform as well as human experts in diagnosing abdominal pain [de Dombal, et al. 72], yet their authors might not choose to characterize these programs as expert systems. Conversely, with the current near-magical attractiveness of expert systems, almost any complex program might get labeled as such. Nevertheless, there are some characteristics that typically identify an expert system, based on the type of problem it is solving, the form in which its knowledge is encoded, etc.

Many expert systems have been implemented to solve problems that are some form of diagnostic reasoning, ranging from medical diagnosis to troubleshooting generators, locomotives, and telephone networks. Many expert systems use simple if-then rules to express their bits of knowledge, with a uniform rule interpreter to link together such rules into more complex, longer chains of inference. Much of the knowledge in expert systems

[3]My own preference is for the term "knowledge-based systems" because it focuses on the critical role of knowledge. Whether the program actually works in ways comparable to those used by human experts or, indeed, whether its competence is that of an expert or only that of a thorough journeyman seems less significant. Nevertheless, I will use the more widely-used term "expert systems" here.

is heuristic in nature—it consists of rules of thumb that are often, but not always, true. Many systems reason by making assumptions that are then verified or rejected in later analysis. None of these characteristics is necessary in an expert system, however, and none is criterial; yet most expert systems have characteristics similar to these.

A more abstract, and more useful, view of expert systems is as a programming methodology that emphasizes the separation of what is true about the world from how to use knowledge to solve problems. In this view, an expert system consists of two components:

DOMAIN KNOWLEDGE	PROBLEM-SOLVING METHODS

In simple cases, the domain knowledge will be a collection of facts and the problem-solving methods will be more general-purpose reasoning mechanisms.

For a concrete example of how one can separate domain knowledge from problem-solving methods, consider the following (oversimplified) example from medical diagnosis. We may begin with a general notion such as "If Mary has a fever, then Mary has an infection," which is a rather specific problem-solving method for diagnosing the cause of Mary's fevers. Using the traditional programming notion of variables, we can generalize to the following picture:

FACTS	**METHOD**
Mary is a patient.	If the *patient* has a fever, then the *patient* has an infection.

where the left column represents domain facts and the right is the slightly more general method.

A more general rule, which begins to deserve to be called a true problem-solving method, is achieved by further abstracting this case:

FACTS	**METHOD**
Mary is a patient. Fever is a symptom. Infection is a disease. Fever is a symptom of infection.	If the *patient* has, the *symptom* of a *disease* then the *patient* has the *disease*.

A final generalization would bring us to something like

FACTS	**METHOD**
Mary is a patient.	If the *object* exhibits
Fever is a symptom.	a *feature* of a *class*,
Infection is a disease.	then the *object* belongs to the *class*.
Fever is a symptom of infection.	
Patient is an object.	
A symptom is a feature.	
A disease is a class.	

This latest method is in fact close a general expression of what has been called *abduction*, the process of working backward from observable manifestations to their probable explanatory cause.

In this view, then, expert systems provide a programming methodology that separates factual statements from the methods that determine how those facts will be used. Within this methodology, there remains the flexibility to have both very specific and very general facts and methods. In fact, if a domain lends itself to reasoning by encoding specific ways to deal with many specific problems, then we might expect many rather special-purpose methods for each of them. An example might be testing the many components of a computer system, where each component is likely to require highly-specialized testing methods. If possible, however, the methodology encourages the development of more general problem-solving methods, as illustrated in the example above.

Classical Models of Expert System Construction

The best-known technical approach to building expert systems has employed simple if-then rules to express virtually all of a system's knowledge and a straightforward rule interpreter to link such rules together into longer chains of inference. The assumption here is that all expertise consists of small, modular chunks of knowledge, and that overall intelligent behavior emerges from the simple goal-directed combination of these many chunks. Each rule has the form

If ⟨list of premises⟩ then ⟨conclusion⟩.

A backward-chaining rule-based system, such as Mycin [Shortliffe 76], begins with a goal it is trying to achieve: for example, to determine the identity of any infectious organisms causing significant disease in the patient. It then operates by interleaving two recursive procedures (Figures 2 and 3)

1. If the validity of the goal can be determined by facts already known, then simply return with that answer; otherwise,
2. If the goal is an observable objective fact that the user can supply, then simply ask for it; otherwise (or if the user is unwilling to supply it),
3. Collect the list of rules whose conclusion asserts facts that are potentially relevant to the goal under consideration.
4. Apply each rule (if any) in turn.
5. If the rules that have applied have succeeded in asserting new facts from which the validity of the goal can now be ascertained, then return that result; otherwise,
6. If the user has not already been asked to determine this fact, then ask and return that result.

Figure 2. Mycin's `findout` procedure to determine if a goal fact is true or false.

1. For each item in the rule's list of premises in turn, call the `findout` procedure to determine if it is true or false.
2. As soon as any item is false, the rule fails and its conclusion is not asserted.
3. If all items are determined to be true, then assert the rule's conclusion, with a degree of certainty computed from the certainties of the rule itself and the premises.

Figure 3. Mycin's `monitor` procedure to apply a rule.

that work backward from that goal to further subgoals and eventually to data that can simply be acquired from the user.

The other major rule-based technology employs forward-chaining rules, in which a new fact asserted to the system causes any rule whose premises match that fact to fire. Then, if all the other premises are also matched among facts that have been asserted, the rule's conclusion is in turn asserted as a new fact. OPS is a system operating in just this way, having been carefully tuned to eliminate redundant matching of facts to rules. Systems built using forward-chaining of rules are typically most appropriate for tasks that respond to a stream of input data, such as monitoring problems. It is also possible, however, to simulate the behavior of a backward-chaining system using forward-chaining rules, by explicitly asserting goals as data and re-writing rules to depend on such goal statements. For example, the effect of the backward-chaining rule

If A and B then C.

can be achieved in a forward-chaining system by translating it to the following three rules.

If (Goal C) then (Goal A).
If (Goal C) and A then (Goal B).
If A and B then C.

The first assures that if (Goal C) is ever asserted, then the system will seek the goal of A, in effect chaining backward from the goal of C to the A clause of the premises of the original rule. The second guarantees that if we are still seeking C and A has been demonstrated, then B becomes a goal, and the third is the simple statement that if both A and B have been shown then C should be asserted.

Despite the fact that simulation of backward chaining is possible by forward chaining, some systems provide specific user-visible support for both methods, to hide the details of the translation process exemplified above. A typical example of how both can be useful in the same system is in a diagnostic task, where forward-chaining rules may be used to generate diagnostic hypotheses from facts about a case, and then backward-chaining rules can be used to verify those hypotheses by seeking more data directly relevant to the subgoals of that verification [Fieschi, et al. 85].

Situation:	**Frozen Gas Line**	**Wet Plugs**
Typical Case:	Car won't run	Car won't run
	Weather very cold	Weather humid or rainy
	Started but stalled	When running, rough
Therapy:	Warm up car	Dry out plugs
Predictions:	Once warm, will run	Will stall in rain

Figure 4. Two sample frames from a system to diagnose and fix simple faults in automobile operation.

Although rule-based systems appear to be by far the most common means of implementing expert systems, there are other equally-effective approaches that have been successfully explored as well. The most important such method matches characteristics of a situation to stored profiles of common situations (often called *frames*) in order to identify which one best matches the case at hand. In medical diagnosis, for example, situations correspond to diseases and characteristics to the signs, symptoms and laboratory data commonly associated with each disease. General matching methods, possibly based on additional measures of likelihood or seriousness, can be used to compute the degree of match, and general information-acquisition strategies can determine what additional tests should be performed in order to best differentiate between competing hypotheses. Programs such as

the Present Illness Program [Pauker, et al. 76] and Internist-I[Miller, et al. 83] employed such methods, which also have much in common with the more traditional methods of statistical pattern recognition. Figure 4 shows two frames from a simple auto-club style good Samaritan's knowledge base. Such systems are typically able not only to diagnose what is wrong by matching the preponderance of symptoms, but also to suggest the most useful questions to differentiate among alternatives (e.g., ask about the weather in our example), to suggest appropriate interventions, and to make predictions about what should be expected after the intervention. Naturally, the failure of such an expectation can be a very effective clue to recovering from an incorrect initial decision.

Another form of expert system, for which many tools are also becoming available, actually rejects the need for human beings to formalize the knowledge of a domain. It relies, instead, on general-purpose pattern-recognition or learning methods to generate its own decision-making methods from a sample of previously-decided cases. Most such systems are considerably simpler in their internal structure than other types of expert systems because they base their classification decisions on some combination of independent features of a case. Despite some significant successes with such methods [Quinlan 86, Quinlan, et al. 86], they appear unlikely to capture much of the subtlety or complexity of human decision-making, and will probably be limited to use in data-rich and theory-poor domains.

What is an Expert System Tool?

When early expert systems were constructed, their implementors typically began with a general-purpose programming language (usually LISP) and built a number of utility data structures and programs to represent facts, rules, and strategies, and to perform the basic operations of the system such as rule-invocation, matching, computation of certainties, etc. Initially, it was often very difficult to determine how much of that machinery was highly specific to the domain of the system being built and how much of it would generalize to other fields, because both the specific knowledge of the experimental system and the more general mechanisms it used would be modified as new challenges pointed out defects in the system.

Once a few experimental applications had been built using a particular approach, however, researchers began the difficult task of abstracting out of those programs those aspects that appeared universal. The fruits of such labors were the earliest expert system tools, such as EMYCIN [van Melle 81]. The intellectual and practical claim of such a system is that, based on past experience, it provides a useful organization of knowledge and an appropriate problem-solving method. Thus, to build a new application it is only necessary to express the domain knowledge of the new task in that system's formalism and then the built-in problem-solving techniques will apply to generate intelligent behavior for the new task.

I believe the best way to view such tools is as new programming languages. Unlike conventional programming languages that provide primitive data objects such as integers, strings and arrays, these new languages provide objects such as facts and rules. Unlike languages, whose control structure consists of sequential execution, conditionals and looping constructs, these tools provide rule-chaining and pattern matching as their fundamental control structures.

Evaluating Expert System Tools

The principal objective of evaluating tool systems is to determine their "fit" to the problem under consideration. As in the case of evaluating languages, there is probably no "best" language overall, only a variety of well and poorly-designed tools that make different assumptions about the sorts of problems and domains to which they will be applied and that make different engineering trade-offs that effect the cost and performance of the tools. In this section, I propose a number of characteristics of tool systems and describe some of the questions that should be raised in evaluating specific tools with respect to these characteristics. It is critical to keep in mind, however, that the most important aspect of the evaluation is still how well the tool supports the needs of the system it is to help implement. The lack of an elegant but irrelevant feature should, therefore, lead to no condemnation, and the inclusion of large numbers of other features that are of little use should not count too much in favor of an individual tool. This obvious idea must be kept in mind especially because of the fact that different tool systems vary in price by up to two or three orders of magnitude (not counting the ones that are free, of course) and that the uses they may be put to range from the development of a simple experimental prototype intended primarily to teach its developers about expert system building to serious, large-scale systems intended for production use.

I will address my comments under the following categories: epistemology, inference, assumptions, uncertainty, meaning, user interface, performance and practical issues.

Epistemology

What can be said about the world in the formalisms of the tool? Most tool systems permit the representation of entities, which have attributes and relations to other entities. Tools whose entities are called *concepts* typically also support a taxonomic organization of concepts, where certain pre-defined relationships have built-in meaning. Most common examples of such are:

- is-an-instance-of – "Clyde is an elephant."
- is-a-kind-of – "Elephant is a kind of animal."
- part-of – "The balleen is part of a sperm whale."

The value of such built-in relations is that common inferences that are entailed by these relationships should then be automatically made by the system, typically in a fashion that is more efficient than by its normal inference mechanisms. Typical inferences of this sort might be:

- Clyde has a trunk because he is an elephant and elephants have trunks.
- Elephants must ingest food, because all animals do. In turn, Clyde must eat, because he is an elephant.
- If I haul a sperm whale on board the Pequod, I must have lifted its balleen as well, because it is part of the whale.

Tools based on the notion of *frames* ordinarily also provide constraints on what values and other frames can fill the attributes and satisfy the relationships of an object. (These attributes and relations are often called *slots* in frame systems. For example, we might wish to indicate as part of the definition of the concept of a birthday party that one of the participants must be someone whose birthday is being celebrated. We may also include constraints among slots: for example, that if the guests at the party are children, the beverage to be served is non-alcoholic. Frame systems usually also provide defaults. For example, if I imagine a child's birthday party, I can be fairly sure that dessert will be ice cream and cake if nothing has been said about dessert.

Tools can, in addition, come primed with a large store of actual knowledge about the world, though they ordinarily do not. For example, the tool might come pre-loaded with knowledge of physical objects, ideas, kinds, actions, beliefs, etc. It might thus innately know that a book is not physically divisible for simultaneous use by several people, that the water in a river is, though resulting in less water per person, and that an idea is also, and that it remains whole for each user. Tools that are pre-charged with what might be called "common sense" theories of some aspects of the world have the advantage that they make the task of the system builder simpler. Rather than having to invent the mapping from the real world into the structures provided by the tool *de novo*, the user has a much more structured framework in which to describe the application domain and at the same time automatically gets the tool's built-in knowledge. Despite this promise, existing common-sense theories are rather weak and fragile, and tools ordinarily provide only a few built-in taxonomic relations such as those listed above.

Inference

How can things be figured out? Tools typically provide the capability of making inferences by providing built-in mechanisms to do so automatically and by allowing the tool user to build rules or procedures that will, under the control of the tool, run at appropriate times to deduce some facts from others.

In rule-based systems, one can ask whether the system supports the use of rules in forward or backward chaining (or both), and what other mechanisms might be provided to improve efficiency or to give a finer degree of control over the operations of the system. For example, the *rete* algorithm of OPS reduces the amount of effort the system must expend on matching rules to facts to the bare minimum. Explicit mechanisms such as agendas, which indicate which of a possibly large set of rules are ready to run, and meta-rules, which allow control over the selection or ordering of the agenda, are useful in complex problems where carefully directing the efforts of the problem-solver is necessary to achieve adequate speed. Facilities for chunking of rules to form rule sets, which may be made activate or inactive in groups, can also reduce computing effort significantly if the problem decomposes into recognizably-distinct components that can be addressed in parts.

Matching, between facts in a data base and rules or frames, is also typically provided by a tool. Important considerations are whether variables are supported in either the data or rules, and how efficiently relevant

matching operations can be expressed in the pattern language. For an example of possible inefficiency, consider a system to analyze electrocardiogram recordings, which may have records of hundreds of heart beats in its data base. If I want a rule that checks for a certain condition between two successive heartbeats and express it as

```
If $b_1$ is a beat and $b_2$ is a beat and time($b_1$) < time($b_2$) and there is
    no $b_3$ such that $b_3$ is a beat and time($b_1$) < time($b_3$) < time($b_2$) and
        ... then ...
```

then it will be applied n^3 times where n is the number of beats, in order to match each b_i to each beat. In reality, of course, there are only $n-1$ successive pairs of beats, but the pattern language of the system may not make it possible to avoid the redundant matches that then take more time to filter out.[4] Indeed, typical tutorials on programming with rule-based systems (e.g., [Brownston et al. 85]) include important hints on how to avoid matching inefficiencies in using the tool, often by reformulating the description of the problem being solved or altering the problem-solving strategies. The most general matching capabilities are provided by *unification*, used in PROLOG. Naturally, this generality may, without careful thought, also lead to the most severe inefficiencies.

In frame systems, computations are often *attached* to particular slots of frames, serving as *if-needed* or *if-added daemons* that are run when some other part of a system either needs or asserts the value of a slot. The computation may be expressed either as a rule or, more commonly, as a simple procedure in a programming language such as LISP. These daemons can be used for propagating information in a system, for creating new entities when needed, and for any other computational task. An example of propagation might be an if-added rule on the *celebrant* slot of a birthday party frame that fills in either Kool-Aid or Champagne punch in the *refreshments* slot when the celebrant is asserted, depending on his or her age. New entities may be created by daemons, for example, when someone accepts an invitation to a birthday party (is added to the list of attendees and puts it on his schedule) and creates a new entity for the present he plans to bring.

The calculations of the built-in implications of predefined relations such as is-a-kind-of are also part of the responsibility of the inference system. Thus, inheritance of attributes via a kind hierarchy is an important capability given by many tools. In addition, certain classification tasks can be automatically performed by the tool because of those relations. For

[4] This example is an adaptation of a personal communication from William J. Long, describing an ECG analysis system under development by Warren Muldrow.

example, it might be able to conclude automatically that a horse's tail is an animal's tail because it knows that a horse is-a-kind-of animal [Hawkinson 75]. The system might also be able to compute the implications of definitions. Thus, if X is a quadrilateral with equal opposite sides, then X is a parallelogram, by definition. In addition, the tool may provide primitive means to express the idea of covering, mutual exclusion and partition [Brachman and Schmolze 85]. For example, if a shape taxonomy distinguishes between straight-line objects and curved ones, then these two categories may be said to be both mutually exclusive and a covering of the set of shapes. Therefore, any object known to be curved cannot match a straight-line object and, in addition, any shape must be either curved or straight-line. These capabilities support reasoning by exclusion. They can also greatly improve the efficiency of problem-solving by narrowing the search space of objects that might be useful in some task.

In some domains, especially those involving design, *constraint propagation* is another useful inference mechanism. The following is an example from an experimental system intended to design recombinant DNA procedures [Stefik 81a, Stefik 81b]. If bacteria successfully inoculated with a viral fragment are to be separated from those whose inoculation failed by killing the failing bacteria with an antibiotic, then the viral fragment must confer immunity to the bacterium against that antibiotic. Note that this constraint leaves open the choice of which particular bacterium is to be used, what virus will be involved, or what antibiotic will be selected; indeed, these choices can be made in any order by other parts of the reasoner. However, no matter how the choices are actually made, the constraint cannot be violated without invalidating the designed procedure. Making any of these choices can narrow the possibilities allowed for any of the others and in fact may determine that choice uniquely.

Assumptions

In many systems, it may be useful to make assumptions during the course of reasoning. This might be done because a problem-solver is stuck given only what is known to be true and must make some further assumptions (perhaps to be verified later) to proceed, or it may arise because it is useful to compare different hypothetical alternatives. Because assumptions will sometimes be wrong, it is important that the system provide some way to retract them, and furthermore to cause the retraction of an assumption also

to retract the consequences of that assumption. This chaining of retraction is critical because new information inferred from an assumption will often turn out to be wrong if the assumption itself is wrong. This issue has been a difficult one in many historical systems, some of which could fail to retract no-longer-valid consequences and most of which would retract even those that were still valid. The techniques of *truth maintenance* or, as some of its creators would prefer, *reason maintenance*, now solve this problem correctly by keeping with each fact in the data base an indication of what rule or computation asserted it and what other facts [Doyle 79] (and ultimately, assumptions [de Kleer 86]) it depends on.

One particularly useful type of assumption is a *default*, meaning

In the absence of any knowledge to the contrary, assume ...

This type of assumption is quite troublesome, however, because the "absence of knowledge to the contrary" is not something true or false in the world, but only in the inference machinery of the problem solver. Thus, it may change as further computation is performed, and default assumptions may need to be retracted as additional knowledge is gained. The fact that the system can take back assertions that it has previously made makes such systems *non-monotonic* because their list of assertions does not grow monotonically. The greatest difficulty arises when a default assumption causes the system to discover something "to the contrary," in which case care must be taken not to leave the system in an inconsistent state. Some tools prohibit the automatic use of default assumptions in order to escape this difficulty, and require the user or some other program to make assumptions explicitly.

Another common type of assumption that is receiving much study in the research laboratories but that is not widely supported in available systems is the *closed-world assumption*, which roughly states that what is not known to be true is false, or the only objects that satisfy some description are the ones known to satisfy it. Such an assumption is often very useful because it codifies the notion that one can ignore what one doesn't know about. It also formalizes reasoning by exclusion: if I can show that all but one possible hypothesis is false, then the remaining one must be true, if there can be no others.

Terms such as *contexts*, *viewpoints*, *environments* and *hypotheticals* all relate to the idea that separate sets of assumptions can be maintained simultaneously within a system. This is important for tasks where hypothetical alternatives must be compared, for example, in doing sensitivity analyses. Without a built-in ability provided by the tool system, such comparisons can be inordinately expensive. They might require the assertion of one set of assumptions, inferring their consequences, than retracting those assumptions, making the alternative set, and then inferring its consequences. The cycle of retraction and re-assertion must then be performed each time the problem solver wishes to switch back and forth between two sets of assumptions to be compared. Systems such as the ATMS [de Kleer 86] provide more efficient built-in support.

Uncertainty

Many, if not most, application domains innately deal with uncertain knowledge. A tool used to build expert systems for such an application would be most useful if it provides innate support for uncertainty. However, there are many competing methods for dealing with uncertain reasoning.

Classical probability theory, Bayes' rule, statistics and subjective probabilistic techniques all rely on numerical assessments of likelihood, obeying well-known and time-tested laws. The principal problems of these methods revolve around the need to make simplifying assumptions about the real world in order to apply them practically. Recent advances show promise of providing in this classical framework the kind of flexibility that has motivated the development of other methods [Pearl 86].

Certainty factors, theory of confirmation, and other non-probabilistic variants concentrate on *probability updates*—the degree of change in a system's belief in response to new evidence—rather than on absolute probabilities. Although intuitively such methods seem to differ from probability theory, recent analyses indicate that they may simply represent a particular set of independence assumptions in a probabilistic framework [Heckerman 86].

Fuzzy set theory, fuzzy logic and its relatives focus on an imprecise *plausibility* rather than true probabilities, reflecting psychologically motivated observations that in many real-world cases descriptions of an object or situation are only plausible to a degree [Zadeh 78], and that sets describing characteristics of groups (e.g., tall) have imprecise boundaries.

The Dempster-Shafer theory of evidence [Shafer 76] uses a least-commitment approach to probabilistic reasoning, allowing evidence to assign belief not only to individual hypotheses but to sets among whose members the evidence fails to distinguish. Because this theory is expressed in terms of

the powerset of the set of hypotheses, the straightforward application of this theory is impractical except for very limited domains. Considerable research is now pursuing its practical application under realistic simplifying conditions [Gordon and Shortliffe 85].

Structural encoding of uncertainty uses a taxonomic organization of descriptions and represents uncertainty by picking that node in the taxonomy that subsumes all possible descriptions. This is a standard AI method, but is effective only if the taxonomies used in a system correspond to the sets among which uncertainty may lie. Furthermore, such methods give no fine-grained numeric means of expressing the degree of uncertainty.

Several experimental programs rely on some form of dialectical argumentation, where each piece of evidence for and against each hypothesis is explicitly remembered, and where further arguments may apply not only to the hypotheses but to other arguments. This method has been proposed in a system for deliberative reasoning [Doyle 80] and, in a slightly different form, in the theory of endorsements [Cohen 85].

Various *ad hoc* numerical methods that approximate some form of probabilistic reasoning but do not have a precise-enough definition to be analyzed formally.

Many expert systems are in fact built with *ad hoc* methods for dealing with uncertainty. Tools, if they support any uncertain reasoning at all, mostly support some form of probabilistic reasoning or certainty factors or fuzzy set theory. If the tool does not provide any innate support for uncertainty, the developer of a specific expert system may still incorporate his or her own uncertainty-handling theory by making explicit assertions of *degree of belief* on other assertions, if the tool's epistemology allows assertions to be made about other assertions.

Reasoning with uncertainty is a "hot topic" in AI research [Kanal and Lemmer 86], and one may expect the fruits of many current efforts to be reflected in future tool systems.

Meaning

It seems so obvious as to make almost ludicrous the suggestion that an important criterion in judging expert system tools is that their methods of representation and their inference procedures should have some well-defined meaning or, technically, *semantics*. In fact, however, most tools support what is kindly called *operational semantics*, i.e., that what the system does is what it means. This is not to suggest that the operation of the system is not fully defined—only that there may be no simpler way to determine what the system ought to do in some situation than to set up that situation and run the system. It becomes difficult, therefore, to prove any global properties of the system or to characterize fully under what circumstances it will behave as intended.

Although the abuses of knowledge representation schemes without well-defined semantics have been well documented [Woods 75], there is still considerable controversy about how to define a system that is both flexible and expressive enough to state conveniently all the sorts of knowledge needed in expert systems and formal enough to permit an unambiguous interpretation of what any statement means. Systems that are simultaneously (a) formally defined, (b) expressive and (c) complete in their built-in inference procedures appear doomed to being innately very inefficient [Brachman and Levesque 84]. In practice, it appears irresistible for expert system builders to take liberties with the semantics provided by formal systems, to trade some "correctness" for expressiveness or efficiency. A typical example of this is when some association that is almost always true is instead taken to be true by definition, to avoid the system's exploration of the contingency of its being false.

Most current tools pay little attention to this matter, but simply provide a set of capabilities that can be manipulated at will by the expert system builder, who then must shoulder the responsibility for determining that the tool's facilities are being combined in a meaningful manner. For applications in which unanticipated failure can lead to disaster—control of nuclear power plants comes to mind as a frightful example—the fact that the language in which the domain knowledge is expressed has no formal interpretation may be a serious deficiency.

User Interface

If a picture is worth a thousand words, a program with an appropriate graphical interface may be worth dozens that are restricted to text-only interaction. Expert system tools are not unique in wishing to offer sophisticated graphics interfaces, but an important criterion in judging a tool system is what facilities it provides, or, at least, what independently-provided facilities does it allow the user to incorporate?

One important new idea in interfaces that has emerged in expert system tool construction and that is now incorporated in some of the high-end tools is a direct two-way interconnection between the internal state variables of the program and the graphical images depicting those variables. This means that the tool user/expert system builder can easily design an interface that clearly shows relevant aspects of the internal state of the expert system, and, further, that the ultimate expert system user can thereby directly manipulate the internal state of the program simply by manipulating the graphical presentation.

For example, in building a control system for a steam plant, the programmer can declare that one standard gauge should always display the current boiler pressure, another one should display the desired pressure, and a third the anticipated pressure one minute later, in the absence of interventions. The ultimate application user can then directly manipulate these gauges to control or interrogate the system. Changing the indicated value on the "desired pressure" gauge (e.g., by moving the graphical needle with a mouse) might actually change the internal value, and changing the "anticipated" gauge might request a sensitivity analysis to suggest under what conditions the future value could be the one indicated by the user. Of course it has always been possible for an application builder to create such a sophisticated interface. What is new here is a set of built-in facilities to make the two-way mapping state and appearance easy to build.

Performance and Practicality

All the usual issues of evaluating any software product apply to expert system tools:

- Is it actually available (for real, not just the glossy brochures)?
- Does it really work with acceptably few errors?
- Is it well documented? Is training, maintenance, assistance in using it available?
- Can you trust the tool's developers with the future of your project? Are they going to be there in the long run?
- Is it fast enough to form a good basis for your system, or will the tool's inefficiencies make your system a dog? Are there built-in limitations on the size of the knowledge base, number of frames or rules, etc., that your application may exceed? (This is particularly critical in

many PC-based tools, which may have to fit within the 640KB of the PC's address space.)

- Is its price commensurate with its value? If you plan to sell the system you are building, can a stripped-down (run-time only) version of the tool be bundled inexpensively to your customers?
- What hardware environment does it demand? In particular, is there a reasonable prospect that it will run on PC-class machines, at least the newer 80386 or 68020/68030 machines? Does it depend on, support, or make impossible networking and communications you or your clients are likely to be using?

Ultimately, of course, the key question is whether the tool appears to provide the kinds of features that you believe your intended expert system to require.

In considering this list, it is probably worthwhile to keep in mind that expert system tools are typically more problematic to use and evaluate than programming languages. This is because for most programming languages at least the language is clearly defined, multiple implementations by competing vendors are typically available, test suites and benchmarks have often been prepared, and there is a large group of programmers among whom you can seek employees and consultants. Virtually each expert system tool is, by contrast, unique. Even if its capabilities are similar to those of another tool, they are slightly different in meaning or appearance. Usually only very limited examples of proper use are available in a tutorial manual, and there may be few if any skilled tool users.

Finally, in evaluating expert system tools, it is important to keep in mind that for a large project that stretches the capabilities of existing tools, it may well be worthwhile producing a custom tool specifically tailored to the needs of the particular application. The rationale is that any project will typically need a specific combination of capabilities, which can be more elegantly designed or be made more efficient knowing that other capabilities are not needed and can therefore be designed out. The cost of such an approach is very high, however. It requires designers and programmers capable of building, not only of using tools. Therefore it is often advisable, even if building your own tools seems appropriate, to do prototyping work with an existing tool and then, once the requirements of the target system are well understood, make a careful assessment of whether the clarity and efficiency to be achieved by the custom tool and the cost saving and lesser

reliance on outsiders offered by it is worth the time, expense and effort involved.

The Future of Expert System Tools

Given the rosy outlook for the expert system tools market, many many vendors are engaged in the continual improvement of existing tools, the migration of the most sophisticated capabilities into smaller and cheaper tools, and the adaptation of tools running on high-cost personal computers such as Lisp machines to more conventional workstations. In keeping with the tone of this review, I will not try to extrapolate along this line, except to note that the next generation of conventional workstations will have roughly the computing power of today's $50,000 Lisp machines, several Lisp machine manufacturers have plans to build and market single-chip machines, and thus is seems safe to predict that, by one or the other of these paths, what are today the high-end tools should become available for general use on $5–10,000 workstations.

Looking at more fundamental issues, it seems that there are two lines of research likely to have a significant impact on expert system tools. The first attempts to identify, within the many tasks for which expert systems are being built, recognizable standard types of problems to which already-built solutions can be applied or at least adapted. The second continues the historical precedent in expert systems research: to develop new one-of-a-kind problem solvers capable of reasoning that current systems cannot do, and then to generalize such systems to form the basis for new tools.

Generic Tasks and Generic Methods

Even with the best of today's tools, it is still quite a difficult task to build a new expert system. Naturally, the fruits of experience embodied in the tools makes such a task easier than if one had to begin simply with a programming language such as LISP or PROLOG, because at least many of the decision about issues described above have already been made and suitable implementations have been provided. Is it possible to do better than this, to improve the productivity of future expert systems builders?

The key observation is that many expert systems are really similar to ones already built. In the simplest sense, a system to configure one line of computer systems should be able to profit directly from previous work done on a system to configure a different line. The particular knowledge in the two cases should be different, but one would expect most of the methods to be the same. Ranging a little farther afield, one might expect that an automobile manufacturer facing the problem of assembling a large variety of custom options for a car line should also profit (if less strongly) from the experiences of the computer configuration system. After all, building an automobile, just as building a computer, consists of assembling a number of selected components and making sure that they obey physical, electrical and mechanical constraints. Naturally, cars don't have buses and backplanes (except insofar as they are becoming computerized), and physical constraints are more likely to be critical than electrical ones. Nevertheless, the two configuration tasks share significant features in common. Generalizing even further, one can recognize the similarities of these tasks to all sorts of synthesis or assembly problems.

Now when the first computer-configuration expert program was built [McDermott 82], its builders began with OPS[Brownston et al. 85] and a set of ideas that needed to be codified in this programming language. In such a project, the requirements of practicality and speedy prototyping often prevent an adequate generalizable design. By the time the fifth or sixth such system is built, however, the problems of representing the computer configuration task should be well-enough understood that one would expect to be able to re-use much of what had been accomplished in the most recently-built such system. Further, if the problem of computer configuration comes to be thoroughly understood, we should expect that understanding to transfer to other related tasks.

To exploit this insight, various researchers have begun to try to classify problem domains and problem-solving methods into generic categories. Clancey [Clancey 85], for example, suggests the taxonomy of problem types shown in Figure 5.

The hypothesis underlying such a taxonomy is that it will be possible to identify particular approaches to problem-solving that are most suitable to each of the above classes of problems. Then, generic tools that provide such approaches as built-in facilities can be directly applied to new problems once the problem type is identified within the classification hierarchy. Indeed, one such problem-solving method, *heuristic classification* appears to underlie the operations of many existing expert systems.

Heuristic classification is a problem-solving method applicable to many interpretation problems. It is characterized by a task in which one starts

- Systems Synthesis Problems
    - Construct a Solution
        - Specify (constrain)
        - Design (generate)
            - Configure (structure)
            - Plan (process))
        - Assemble (manufacture)
            - Modify (repair)
- Systems Analysis Problems
    - Interpret a Situation
        - Identify (recognize)
            - Monitor (audit, check)
            - Diagnose (debug)
        - Simulate (predict)
        - Control

Figure 5. Clancey's proposed taxonomy of problem types to which various problem-solving methods may be applied.

with a pre-enumerated set of possible solutions, a quantity of data either already given or capable of being elicited, and the goal of finding the single solution that best characterizes the data. The fixed set of possible solutions is key to distinguish this from other methods. According to one analysis [Clancey 85], heuristic classification employs three sub-methods:

Data Abstraction Classifies particular data values reported to the program into conceptual categories that are important for further reasoning. Examples are turning quantitative values into qualitative judgments (e.g., a blood pressure of 90 over 60 is "significantly low"), recognizing the presence of states defined in terms of data values (e.g., "hypotension" means "low blood pressure," and applying generalizations. Long [1983] points out that judgments leading from quantitative to qualitative values cannot be made in isolation from other data. For example, the above-mentioned "low" blood pressure may in fact be "normal" for a patient whose system has adjusted to such a long-term state.

Heuristic Match Makes the "leap" from a suitable abstraction of the data to an abstract version of the solution. These associations are often based on experience, skip over numerous intermediate steps, and are subject to error. This is the step often identified as the "rule of thumb" in expert systems.

Refinement Fills in the details of the abstract solution suggested by the heuristic match.

It is perhaps unrealistic to assume that each problem type will have a unique best method of solution. In fact, it may be that problem-solving methods can also be classified into a taxonomy based on their similarities, and then one can develop a two-dimensional relationship among problem types and problem-solving methods, with an indication of the appropriateness of each method to each problem type. Attempts at such analyses have been made by Stefik, *et al.* [1983] and Chandrasekaran [1985].

Efforts to identify generic tasks and methods are still very much limited to research projects. In the commercial environment, the first indication that these ideas may become practicable comes from the announcement of "application packages" for use with large-scale expert system tools. Such packages augment the built-in facilities of the tool with techniques particularly important to some types of problems. An example is a simulation-oriented package for use with KEE. If research along these lines is successful, we can expect much more specific methods provided in tools, perhaps in industry-specific versions.

New Techniques from Applied Artificial Intelligence Research

Despite some impressive successes and almost universally-held optimism for future capabilities, artificial intelligence and the methods of reasoning and knowledge representation in expert system are still in their infancy. In the long term, therefore, it is inevitable that the greatest improvements in expert systems tools will come from advances in artificial intelligence research. A better understanding of "common-sense reasoning," for example, would be not only a major advance in the science of artificial intelligence but would directly benefit all expert systems by allowing them to interact in a more realistic way with their users. This central problem in AI research is not about to fall (although there are interesting recent proposals that may point in the right directions [Minsky 86]), but other, more limited advances should significantly improve the capabilities of future artificial intelligence systems.

My crystal ball is sufficiently opaque that I cannot make good predictions about where among the many avenues of AI research the most significant advances are likely to come. I will simply enumerate a list of interesting research efforts now underway.

In ordinary human terms, we think we understand a problematic situation when we have a *causal explanation* for the undesirable characteristics of that situation that allows us to assign ultimate causes and to elucidate causal pathways leading from those causes to the caused characteristics. To support such reasoning, systems must include a moderately sophisticated theory of causality, which will, in turn, rest on more basic ideas of how to represent the physical world,[5] including its objects and processes, how to simulate the behavior of systems thus described, and how to reason back from a description of a current state to the defects that might account for its problems.

Programs that reason about defects in the behavior of complex systems (ranging from electronic circuits to human bodies) require an ability to analyze the behavior of a system in terms of its physical realization—in other words, to derive *function from structure*. We believe that ultimately all misbehavior of a system can be traced to some defect in the structures that make up that system: it doesn't work (correctly) either because some piece of it is not working or because the pieces are not connected correctly. Given a complete account of how every ultimate part works and rules that determine the behavior of an aggregate from the behaviors of the parts, one can, in principle, perform numerous tasks such as design and diagnosis in terms of building up behavioral models of large system parts from the behaviors of their smallest pieces. To do this well, we need good methods of describing both parts and the behaviors. In addition, there are many difficult methodological concerns, intended to guarantee that no unwarranted

[5]See [Bobrow 85] for a good recent collection of papers concerning this topic.

assumptions should be made in the course of such analyses [de Kleer and Brown 83].

For practical and cognitive reasons, it seems inappropriate to reason about complex systems by reducing them to their most basic components. One really cannot understand why a computer system fails to respond to console input by tracing the anticipated signal path gate by gate throughout the system. Instead, it seems crucial to permit a reasoning system to deal with problems at *multiple levels of detail* and perhaps from *multiple viewpoints*. A medical diagnosis program, for example, can incorporate both a simple association model relating common diseases to their common manifestations and deeper causal models (perhaps at several levels of detail). Then, the associational model can be used to generate possible intermediate diagnostic conclusions quickly, and the more detailed models can be consulted later to confirm the hypothesized diagnosis or to help make sense of a complex case for which the associational model was misleading or inconclusive [Patil et al. 81]. In addition to the hierarchic organization of detail, other organizing principles may also allow more efficient computation. In analyzing electronic circuits, the circuit diagram, highlighting planned electrical interconnections, is often a good vehicle for reasoning. As Davis [1984] points out, however, the physical layout of chips and circuit-board leads is a much better basis for finding bridge-faults caused by excess solder, and the three-dimensional arrangement of parts inside a chassis may be the best representation for localizing heat-dependent problems. Pople [1982] suggests a clever set of heuristics to exploit simultaneous constraints from two different viewpoints on a medical diagnosis problem.

People seem comfortable thinking about many problems in *qualitative* terms. Whether making judgments of physical distance ("how high will this ball fly before starting to drop back to earth?"), likelihood ("how likely is one of the cars in the Indy 500 to crash this year?"), temperature ("how hot is the water available from my water-heater?"), etc., we never expect precise numerical answers (45 feet, 0.08, or 185°F). Instead, some qualitative description of such magnitudes appears adequate and even more useful. The relevant characterization of water temperature if I am about to bathe, for example, is probably into some three-range scale: too cold for a bath, acceptable, and scalding. The distinguished points in such a "quantity space" [Forbus 84] are those at which processes of interest (shivering, pain or damage to skin) begin to operate. Reasoning with qualitative descriptions only is sometimes adequate to determine the approximate behavior

of physical systems. For example, if I have drawn my bath and know tha it is too hot, I can determine that either adding cold water (using a mixin process) or simply waiting for the water to cool (heat transport process are acceptable ways to keep from getting scalded, without reference to th actual temperature of the water, the heat-conductivity of the tub and air or my threshold of pain for hot water.

In many situations, however, such simple qualitative arguments los too much information to be usable. For example, if I add both hot and col water to the bathtub, I cannot tell if its temperature will rise or fall without knowing something about the various temperatures and water volume involved. Pure qualitative reasoners therefore often generate ambiguous interpretations of some situation, even though a more detailed analysis would yield a definite solution. Progress is now being made toward building systems with the advantages of both qualitative and quantitative reasoners. In such a hybrid system, the key is to allow a representation of quantities and relationships that says just so much as there is information to support. In a system called QMR [Sacks 85], for example, functional relationships can be represented either precisely in terms of mathematical expression, or, with less information, in terms of parameterized algebraic expressions, or, even more qualitatively, by simply indicating the behavioral trend of a function, such as "monotonically increasing" on some interval. The system guarantees to combine functional forms without introducing new ambiguities. Similar approaches have also been developed for reasoning about uncertainty and utility, where it may not be the absolute magnitude of some value that matters as much as relative rankings of alternative courses of action [Wellman 85].

One of the most critical problems with today's means of building expert systems is that virtually each new system must be custom-crafted. Earlier, I have described attempts to speed up this process by identifying generic tasks and methods for expert problem solving. However, even as such categorizations succeed, they will leave to the system builder the need to gather and formalize the knowledge of the domain of application and to build it into the system. Ultimately, why can't the computer itself play a much larger role in that process of gathering and formalizing knowledge? The problems of *learning* have received a great deal of attention throughout the history of AI research, and there has been a recent revival of interest and active research [Michalski et al. 83, 86]. In addition to the statistical learning approaches I touched upon earlier, there are many efforts to relate learning to structural descriptions of tasks and the capabilities of the reasoner, and efforts to describe learning as guided search through a space of possible theories [Mitchell 82].

As the amount of knowledge in an expert system grows larger and more comprehensive, the problem of *control of reasoning* becomes ever-greater. After all, if a system knows how to do only one task and knows just enough to accomplish it, then even exhaustive methods of searching through its knowledge may be quite acceptable for good problem-solving performance. As we try to make systems more comprehensive, however, and as we try to build into them a deeper understanding of their domains, it becomes harder and harder to control the directions in which the system explores. Attempts at solving this problem range from novel inference methods that can be used effectively in alternate ways (*logic programming*, *constraints*) to programming frameworks in which one can explicitly reason about the reasoning task itself [de Kleer et al. 77, Doyle 80, Genesereth 83, Nii 86a, 86b].

Even more significant changes may arise in expert systems as AI research begins to understand the capabilities inherent in parallel processing and in models of intelligence based on massive parallelism. Computer architectures such as the Connection Machine, with 64,000 processors representing active nodes in a semantic network open up completely new ways of implementing reasoning systems [Hillis 85]. Today we do not yet understand how to harness this power effectively except for very regular tasks (image understanding, perhaps). The availability of such powerful computing engines, however, will certainly lead to good engineering approaches to exploiting its capabilities. In addition, new theories of knowledge and reasoning, such as *connectionism*, should generate many new ideas and approaches to expert system construction.

For each of the above topics, I think, successes in improving our abilities to solve problems and to represent knowledge in these ways clearly points to a greatly improved potential for future expert systems tools.

Acknowledgments

This paper and the presentation on which it is based is an expansion of a portion of a tutorial taught at the 1985 International Joint Conference on Artificial Intelligence by the author and Dr. William J. Clancey, whose help in formulating these ideas I gratefully acknowledge. The author's research is supported by Supported by National Institutes of Health grants R24 RR 01320 from the Division of Research Resources, R01 HL 33041 from the National Heart, Lung and Blood Institute, R01 LM 04493 from the National Library of Medicine, and by a fellowship gift from the Sperry Corporation.

References

Bobrow, D. G., 1985, *Qualitative Reasoning about Physical Systems*, MIT Press.

Brachman, R. J., and Levesque, H. J., 1984, "The tractability of subsumption in frame-based description languages," *Nat. Conf. on Artificial Intelligence*, pp. 34–37.

Brachman, R. J., and Schmolze, J. G., 1985, "An overview of the KL-ONE knowledge representation system," *Cognitive Science*, 9, pp. 171–216.

Brownston, L., Farrell, R., Kant, E., and Martin, N., 1985, *Programming Expert Systems in OPS: An Introduction to Rule-Based Programming*, Addison-Wesley.

Chandrasekaran, B., 1985, "Generic tasks in knowledge-based reasoning: characterizing and designing expert systems at the 'right' level of abstraction," *IEEE Computer Society Artificial Intelligence Applications Conference.*

Clancey, W. J., 1985, "Heuristic classification," *Artificial Intelligence*, 27, pp. 289–350.

Cohen, P. R., 1985, *Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach.* Volume 2 of *Research Notes in Artificial Intelligence*, Pitman.

Davis, R., 1984, "Diagnostic reasoning based on structure and behavior," *Artificial Intelligence*, 24, pp. 347–410.

de Dombal, F. T., Leaper, D. J., Staniland, J. R., McCann, A. P. and Horrocks, J. C., 1972, "Computer-aided diagnosis of abdominal pain.," *British Medical Journal*, 2, pp. 9–13.

de Kleer, J., 1986, "An assumption-based TMS," *Artificial Intelligence*, 28, pp. 127–162.

de Kleer, J. and Brown, J. S., Aug. 1983, "The origin, form and logic of qualitative physical laws," *Eighth Int. Joint Conf. on Artificial Intelligence*, pp. 1158–1169.

de Kleer, J., Doyle, J., Steele, G. L., Jr., and Sussman, G. J., 1977, "AMORD: Explicit control of reasoning," *ACM Symposium on Artificial Intelligence and Programming Languages*, pp. 116–125.

Doyle, J., 1979, "A truth maintenance system," *Artificial Intelligence*, 12 no. 2, pp. 231–272.

Doyle, J., 1980, "A Model for Deliberation, Action, and Introspection," MIT Artificial Intelligence Laboratory TR 581.

Feigenbaum, E. A., Buchanan, B. G., and Lederberg, J., 1971, "On generality and problem solving: a case study using the dendral program," *Machine Intelligence 6*, edited by B. Meltzer and D. Michie, American Elsevier, pp. 165–190.

Fieschi, M., Joubert, M., Fieschi, D., Botti, G., Michel, C., and Proudhon, H., 1985, "The sphinx project," *Artificial Intelligence in Medicine*, edited by I. de Lotto and M. Stefanelli, North Holland, pp. 83–93.

Forbus, K. D., 1985, "Qualitative Process Theory," MIT Artificial Intelligence Laboratory TR 789.

Genesereth, M. R., 1983, "An overview of meta-level architecture.," *Nat. Conf. on Artificial Intelligence*, pp. 119–124.

Gordon, J. and Shortliffe, E. H., 1985, "A method for managing evidential reasoning in a hierarchical hypothesis space," *Artificial Intelligence*, 26, pp. 323–357.

Hawkinson, L. B., 1975, "The representation of concepts in OWL," *Fourth Int. Joint. Conf. on Artificial Intelligence.*

Heckerman, D., 1986, "Probabilistic interpretations for MYCIN's certainty factors," *Uncertainty in Artificial Intelligence*, edited by L. N. Kanal and J. F. Lemmer, North Holland.

Hillis, W. D., 1985, *The Connection Machine*, MIT Press.

Kanal, L. N. and Lemmer, J. F., 1986, *Uncertainty in Artificial Intelligence*, North Holland.

Long, W. J., 1983, "Reasoning about state from causation and time in a medical domain," *Nat. Conf. on Artificial Intelligence*, pp. 251–254.

Mathlab Group, Jan. 1983, *Macsyma Reference Manual*, MIT Laboratory for Computer Science.

McDermott, J., 1982, "R1's formative years," *Artificial Intelligence Magazine*, 2 no. 2, pp. 21–29.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., 1983, *Machine Learning*, Tioga Publishing Company.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., 1983, *Machine Learning* Volume 2, Tioga Publishing Company.

Miller, R. A., Pople, H. E., Jr., and Myers, J. D., 1982, "Internist-1, an experimental computer-based diagnostic consultant for general internal medicine," *New England Journal of Medicine* , 307, pp. 468–476.

Minsky, M., 1986, *The Society of Mind*, Simon and Schuster.

Mitchell, T. M., 1982, "Generalization as search," *Artificial Intelligence*, 18, pp. 203–226.

Nii, H. P., Summer 1986, "Blackboard systems, part 1: the Blackboard model of problem solving and the evolution of Blackboard architectures," *AI Magazine*, 7 no. 2, pp. 38–53.

Nii, H. P., August 1986, "Blackboard systems, part 2: Blackboard application systems, Blackboard systems from a knowledge engineering perspective," *AI Magazine*, 7 no. 3, pp. 82–106.

Patil, R. S., Szolovits, P., and Schwartz, W. B., 1981, "Causal understanding of patient illness in medical diagnosis," *Seventh Int. Joint Conf. on Artificial Intelligence*, pp. 893–899.

Pauker, S. G., Gorry, G. A., Kassirer, J. P., and Schwartz, W. B., 1976, "Towards the simulation of clinical cognition: Taking a present illness by computer," *Amer. Jour. of Medicine*, 60, pp. 981–996.

Pearl, J., 1986, "Fusion, propagation and structuring in belief networks," *Artificial Intelligence*, 29, pp. 241–288.

Pople, H. E., Jr., 1982, "Heuristic methods for imposing structure on ill-structured problems: the structuring of medical diagnostics," *Artificial Intelligence in Medicine*, edited by P. Szolovits, Westview Press, pp. 119–190.

Quinlan, J. R., 1986, "Induction of decision trees," *Machine Learning*, 1, pp. 81–106.

Quinlan, J. R., Compton, P. J., Horn, K. A., and Lazarus, L., 1986, "Inductive knowledge acquisition: a case study," *Second Australian Conf. on Applic. of Expert Systems*, pp. 183–204.

Sacks, E. P., 1985, "Qualitative mathematical reasoning," *Ninth Int. Joint Conf. on Artificial Intelligence*, pp. 137–139.

Shafer, G., 1976, *A Mathematical Theory of Evidence*, Princeton University Press.

Shortliffe, E. H., 1976, *MYCIN: Computer-based Medical Consultations*, American Elsevier.

Stefik, M., 1981, "Planning with constraints (MOLGEN: part 1)," *Artificial Intelligence*, 16 no. 2, pp. 111–140.

Stefik, M., 1981, "Planning and meta-planning (MOLGEN: part 2)," *Artificial Intelligence*, 16 no. 2, pp. 141–170.

Stefik, M., Aikins, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F., and Sacerdoti, E., 1983, "The architecture of expert systems," *Building Expert Systems*, edited by F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, Addison-Wesley.

van Melle, W., 1981, *System Aids in Constructing Consultation Programs*, UMI Research Press.

Wellman, M. P., May 1985, "Reasoning about preference models," MIT Laboratory for Computer Science TR 340.

Woods, W. A., 1975, "What's in a link: Foundations for semantic networks," *Representation and Understanding*, edited by Bobrow and Collins, Academic Press, pp. 35–82.

Zadeh, L. A., 1978, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy Sets and Systems*, 1, pp. 3–28.

A Case Study on Evolution of System Building Expertise: Medical Diagnosis

Ramesh S. Patil

Introduction

The widespread use of expert system techniques in application software development is a recent phenomenon. The majority of projects have been under development for less than five years and very few of them have reached a level of maturity where they can be usefully applied. To anticipate the future progress of these systems and the field in general, it is useful to study those few example domains which are of long-standing interest.

The application of artificial intelligence techniques to medicine and, in particular, to medical diagnosis is one such area. In this paper we will study the evolution of computational techniques in the area of medical diagnosis. I will present a number of systems with increasing capabilities and complexity with particular emphasis on the interaction between knowledge-representation and reasoning strategies, and on how our understanding of the nature of diagnostic expertise has changed over time. We will then look at the current outstanding problems and the approaches under implementation to solve them.

AI in the 1980s and Beyond

An MIT Survey

Edited by W. Eric L. Grimson and Ramesh S. Patil

**The MIT Press
Cambridge, Massachusetts
London, England**