RULE-BASED SYSTEMS

Rule-based systems automate problem-solving know-how, provide a means for capturing and refining human expertise, and are proving to be commercially viable.

FREDERICK HAYES-ROTH

Rule-based systems (RBSs) constitute the best currently available means for codifying the problem-solving know-how of human experts. Experts tend to express most of their problem-solving techniques in terms of a set of situation–action rules, and this suggests that RBSs should be the method of choice for building knowledge-intensive expert systems. Although many different techniques have emerged for organizing collections of rules into automated experts, all RBSs share certain key properties:

1. They incorporate practical human knowledge in conditional if–then rules,
2. their skill increases at a rate proportional to the enlargement of their knowledge bases,
3. they can solve a wide range of possibly complex problems by selecting relevant rules and then combining the results in appropriate ways,
4. they adaptively determine the best sequence of rules to execute, and
5. they explain their conclusions by retracing their actual lines of reasoning and translating the logic of each rule employed into natural language.

RBSs address a need for capturing, representing, storing, distributing, reasoning about, and applying human knowledge electronically. They provide a practical means of building automated experts in application areas where job excellence requires consistent reasoning and practical experience. Table I lists some application areas currently addressed by RBS technology.

The hallmark of these systems is the way they represent knowledge about plausible inferences and preferred problem-solving tactics. Typically, both sorts of know-how are represented as conditional rules. Figures 1 and 2 illustrate rules that apply in a variety of applications and that employ some of the basic syntactic conventions of RBSs.

We can define RBSs as *modularized know-how systems*, where know-how is practical problem-solving knowledge. Such knowledge consists of various kinds of information, including

1. specific inferences that follow from specific observations;
2. abstractions, generalizations, and categorizations of given data;
3. necessary and sufficient conditions for achieving some goal;
4. likeliest places to look for relevant information;
5. preferred strategies for eliminating uncertainty or minimizing other risks;
6. likely consequences of hypothetical situations;
7. probable causes of symptoms.

TABLE I. Applications of RBSs

Problem	System functions
Equipment maintenance	Diagnosing faults and recommending repairs
Component selection	Eliciting requirements and matching parts from an electronics catalog
Computer operation	Analyzing requirements, and selecting and operating software
Product configuration	Eliciting preferences and identifying parts that satisfy constraints
Troubleshooting	Analyzing situations, suggesting treatments, and prescribing preventative measures
Process control	Spotting problematic data and remedying irregularities
Quality assurance	Assessing tasks, proposing practices, and enforcing requirements

Today's RBS technology provides the first practical methodology and notation for developing systems capable of knowledge-intensive performance. Although artificial intelligence (AI) researchers have developed several alternatives, only the RBS approach consistently produces expert problem solvers. This reflects a feature of the current state of the art in automatic reasoning—that RBSs can directly incorporate rules that emulate the effective special-case reasoning characteristic of highly experienced professionals. General-purpose deductive schemes do not emulate experts and therefore lack the efficiency necessary for solving complex practical problems. Because each rule in an RBS approximates an independent nugget of know-how, these systems have two characteristic features: First, existing knowledge can be refined, and new knowledge added, for incremental increases in system performance. Second, systems are able to explain their reasoning, making their logic practically transparent and allowing them to satisfy the widely recognized need for understandability in computer systems.

By incorporating know-how acquired in an incremental and transparent manner, RBSs open up key computing applications not readily addressable by alternative techniques. Some of these applications are in areas where the supply of quality human workers is insufficient. There are a number of possible reasons for this: The skill level among trained workers may not be consistently high enough, experts may perform at a level far beyond that of average workers, or conventional means of training and automation may fail to produce adequate performance. Automating expertise in specialized tasks generally requires a few hundred to a few thousand heuristic rules. With existing technology, this makes good economic sense in hundreds of application areas.

Of course, in spite of such relevance and potential,

```
If the plaintiff did receive an eye injury
    and there was just one eye that was injured
    and the treatment for the eye did require surgery
    and the recovery from the injury was almost complete
    and visual acuity was slightly reduced by the injury
    and the condition is fixed,
increase the injury trauma factor by $10,000.

If the plaintiff's injury did cause
        (a temporary disability of an important function)
    and the plaintiff's doctors were not certain about
        the disability being temporary
    and the plaintiff's recovery was almost complete
    and the condition is fixed,
increase the fear factor by $1,000 per day.

If the plaintiff did not wear glasses before the injury
    and the plaintiff's injury does require
        (the plaintiff to wear glasses),
increase the faculty loss factor by $1,500
    and increase the inconvenience factor by $1,500.
```

The ROSIE programming system, developed at RAND, provides a stylized English-like syntax for expressing conditions and actions. Each rule expresses an independent chunk of know-how.

FIGURE 1. The Representation of Legal Heuristics for Product Liability in a ROSIE Program

```
Rule408:
  C is a car.
  If:   the pattern observed by attaching an oscilloscope
        to the charging circuit of the car C is
        fluctuating.arches, and
        the alternator of the car C responds properly to
        different loads,
  Then: there is strongly suggestive evidence <.9> that
        the cause of the problem with the car C is
        voltage.regulator.bad.

Rule428:
  C is a car.
  If:   the pattern obtained by attaching an oscilloscope to the
        charging circuit of the car C is straight.line, and
        the result pulling out the field connector is no.flash, and
        the field connector has does not have a voltage, and
        the input of the voltage regulator does not have a voltage, and
        the dashboard lights do not glow when their
        ground circuit is completed, and
        the fusable link is getting voltage, and
        the fusable link is not conducting power,
  Then: it is definite <1.0> that the cause of the problem with the car C
        is fusable.link.bad.
```

As cars incorporate more electronic subsystems, they become more difficult for the average technician to repair. Teknowledge's S.1 expert-system building tool provides a structural framework for organizing and applying thousands of rules. General Motors plans to aid its service technicians with several large-scale RBSs.

FIGURE 2. The Representation of Automotive Troubleshooting Rules in an S.1 Program

RBS technology does have its shortcomings. As a new technology, it will require years of perfecting and fine tuning. Proposed applications must be assessed carefully for feasibility and deployability. Only a small fraction of potential applications can be addressed today with off-the-shelf products. Nevertheless, RBSs constitute the best means available for building expert systems that incorporate large amounts of judgmental, heuristic, experiential know-how. Informal surveys indicate that approximately 50 percent of the Fortune 500 companies are investing in RBSs, and that about 10 percent have applications under development.

The operating concept of RBSs differs radically from von Neumann architectures. Intelligent problem solving with RBSs involves an iterative cycle of (1) identifying from experience the heuristic rules that bear on a problem at hand, and (2) applying one of those rules to solve or simplify the problem. The technology for building RBSs supports this cycle by providing a dynamic working memory for partial results, a device to identify relevant rules, and selective means for applying desirable rules. Many people conjecture that human problem-solving activity follows the RBS model. Whether or not that proves true, human experts generally find it easy to express methods for solving problems in their application areas by using a rule formulation.

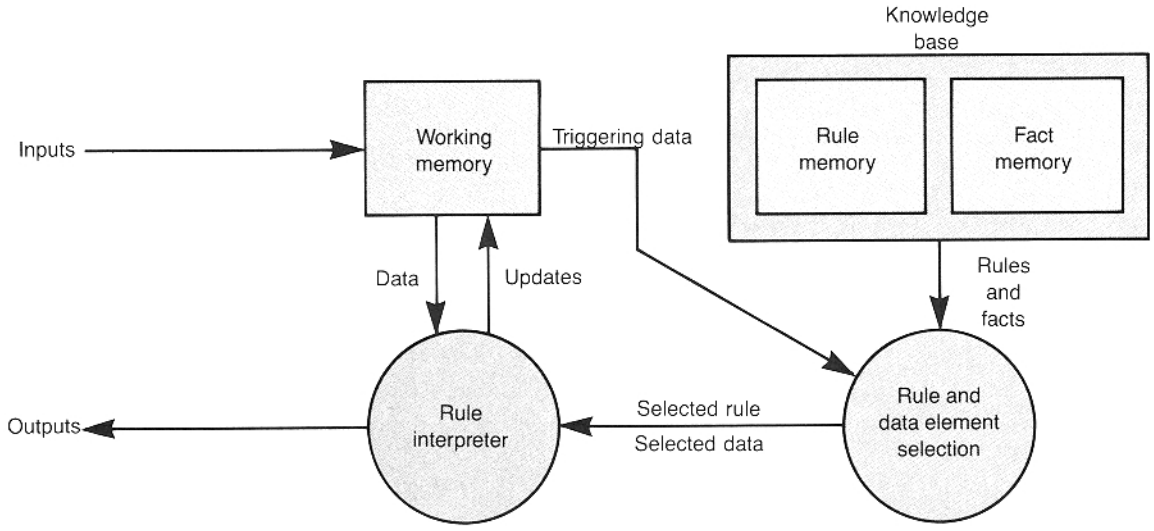
The technology for building RBSs has progressed significantly in the last 10 years, as many people have analyzed the technology and assessed its relevance for a variety of tasks. Today, we can see that rule-oriented components are becoming central in many advanced computing applications.

AN OVERVIEW OF RBSs

Roughly speaking, an RBS consists of a *knowledge base* and an *inference engine* (see Figure 3). The knowledge base contains rules and facts. Rules always express a conditional, with an antecedent and a consequent component. The interpretation of a rule is that if the antecedent can be satisfied the consequent can too. When the consequent defines an action, the effect of satisfying the antecedent is to schedule the action for execution. When the consequent defines a conclusion, the effect is to infer the conclusion.

Because the behavior of all RBSs derives from this simple regimen, their rules will always specify the actual behavior of the system when particular problem-solving data are entered. In so doing, rules perform a variety of distinctive functions:

1. They define a parallel decomposition of state transition behavior, thereby inducing a parallel decomposition of the overall system state that simplifies au-



A simple RBS consists of storage and processing elements, which are often referred to respectively as the knowledge base and the inference engine. The basic cycle of an RBS consists of a select phase and an execute phase. During the execute phase, the system interprets the selected rule to draw inferences that alter the system's dynamic memory. System storage includes components for long-term static data and short-term dynamic data. The long-term store, which is the knowledge base, contains rules and facts. Rules specify actions the system should initiate when certain triggering conditions occur. These conditions define important patterns of data that can arise in working memory. The system represents data in terms of relations, propositions, or equivalent logical expressions. Facts define static, true propositions. In contrast to conventional data-processing systems, most RBSs distribute their logic over numerous independent condition–action rules, monitor dynamic results for triggering patterns of data, determine their sequential behavior by selecting their next activity from a set of candidate-triggered rules, and store their intermediate results exclusively in a global working memory.

FIGURE 3. The Basic Features of an RBS

diting and explanation. Every result can thus be traced to its antecedent data and intermediate rule-based inferences.

2. They can simulate deduction and reasoning by expressing logical relationships (conditionals) and definitional equivalences.
3. They can simulate subjective perception by relating signal data to higher level pattern classes.
4. They can simulate subjective decision making by using conditional rules to express heuristics.

Several key techniques for organizing RBSs have emerged. Rules can be used to express deductive knowledge, such as logical relationships, and thereby to support inference, verification, or evaluation tasks. Conversely, rules can be used to express goal-oriented knowledge that an RBS can apply in seeking problem solutions and cite in justifying its own goal-seeking behavior. Finally, rules can be used to express causal relationships, which an RBS can use to answer "what if" questions or to determine possible causes for specified events.

An RBS can only solve problems if it incorporates rules that use symbolic descriptions to characterize relevant situations and corresponding actions. The language employed for these descriptions imposes a conceptual framework on the problem and its solution. The rules may be precise or gross; the intermediate partial solutions abstract or detailed. Efforts to solve the problem may proceed top-down, outside-in, bottom-up, or in some other way. The meaning, importance, and contribution of each rule depend on its effectiveness as a contributor within the entire set of rules available for solving a problem.

Facts, the other kind of data in a knowledge base, express assertions about properties, relations, propositions, etc. In contrast to rules, which the RBS interprets as imperatives, facts are usually static and inactive—implicitly, a fact is silent regarding the pragmatic value and dynamic utilization of its knowledge. Thus, although in many contexts facts and rules are logically interchangeable, in the context of RBSs they are quite distinct.

In addition to its static memory for facts and rules, an RBS uses a working memory to store temporary assertions. These assertions record earlier rule-based inferences. We can describe the contents of working memory as *problem-solving state information.* Ordinarily, the

data in working memory adhere to the syntactic conventions of facts. Temporary assertions thus correspond to dynamic facts.

The computing environment for rule interpretation consists of current facts and the inference engine itself. Together, these provide a context for interpreting the current state, understanding what the rules mean, and applying relevant rules appropriately. Evidence of this implicit frame of reference can be found in Figures 1 and 2. The legal rules specify the changes that are to be made to various "factors" under various conditions, and the auto repair rules draw conclusions about the causes of problems. Of course, these rules are not universally valid. Each depends on many unstated assumptions in its particular frame of reference. The validity of these rules depends critically on their being interpreted in the right context. Thus, RBSs cannot obviate all the concerns of conventional computer programming (e.g., state sequences and variable scoping) because someone must still ensure that an RBS applies rules appropriately and in meaningful contexts. Many people mistakenly assume that RBSs can turn unstructured heaps of universally valid, independent rules into effective problem solvers. That is a serious misinterpretation of current technology. Furthermore, rule writers must consider the rule interpretation environment to ensure that a rule or its applications can be translated into appropriate natural language.

The basic function of an RBS is to produce results. The primary output may be a problem solution, the answer to a question, or an analysis of some data. Whatever the case, an RBS employs several key processes in determining its overall activity. A "world" manager maintains information in working memory, and a built-in control procedure defines the basic high-level loop; if the built-in control provides for programmable specialized control, an additional process manages branches to and returns from special control blocks.

THE RBS NICHE IN COMPUTING

RBSs address a number of shortcomings in conventional programming technology, among them

1. the nonspecifiability of programs,
2. the rapid changes in principles of operation that can arise during development,
3. the lack of user/expert participation in operations specification,
4. the lack of experimental development for computer-based competence, and
5. the lack of expertise in exploiting computer capabilities.

Among the features that allow them to do this are

1. modular know-how;
2. knowledge bases for storing rules and facts that directly determine decisions;
3. the capacity for incremental development with steady performance improvements;
4. explanations of results, lines of reasoning, and questions asked;
5. intelligibly encoded beliefs and problem-solving techniques;
6. inference chains assembled dynamically by built-in control procedures that can often perform efficient searches.

Given this wide range of important applications, it seems probable that the role of RBSs in program development will be expanding.

THE RULE AS AN OBJECT

We speak of a rule as a relatively independent piece or *chunk* of know-how. Psychologists have for some time emphasized the importance of chunks as elementary patterns in perception and thinking. Chunks are a distinctly subjective psychological phenomenon: They reflect the learned, appropriate, effective distinctions that people use to make sophisticated high-level decisions. A rule can serve as just such a chunk of problem-solving know-how.

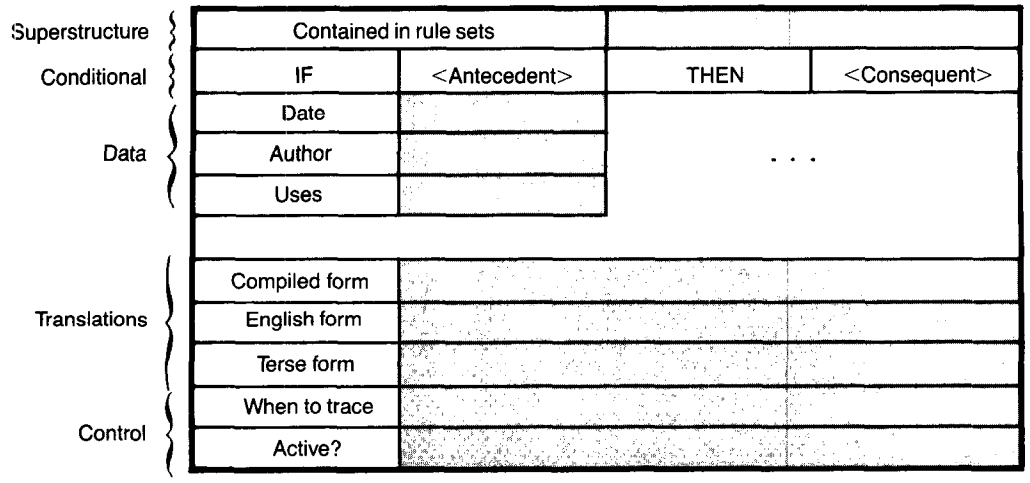
As used by most RBSs, rules specify chunks of analytic problem-solving knowledge. A rule is a datum employed by an inference engine to infer a solution to its goal problem. Thus, a rule writer who expresses know-how in rule format is offering one possible path to reducing a goal to subgoals, to drawing a plausible inference from plausible data, or to transforming an expression. This information about the rule typically comprises its familiar if–then components. However, as the number of rules in an RBS grows, a need arises for rule components that can support multiple functions (see Figure 4) and thereby extend and maintain the knowledge base. For this reason, many additional facets or attributes are introduced to represent data about a rule's analytic knowledge and its preferred manner of use.

RBS ARCHITECTURE

An RBS is generally a complete computing system, which is to say that it can produce an output by applying memory and processing to an input. An architectural inventory of RBS technology would include the following four basic elements:

Rules. From an architectural perspective, rules are data that generally conform to highly specialized grammars capable of using symbolic expressions to define conditions and actions. Current systems differ primarily in the generality and notational convenience their symbologies support.

Interpreters. The rule interpreter matches a rule component to working memory data. Generally this requires pattern matching to find constants in working memory that match identical constants or unbound variables in rule patterns. Existing systems differ primarily in the methods they use to simplify rule definition and pattern matching. The action of the rule is produced by another part of the rule interpreter. Ac-



Rules can contain more information than can be found in a simple if–then conditional. The antecedent and consequent specify data sufficient for inferring a conclusion or performing another action, while other parts of a rule serve additional important roles. For instance, most large RBSs benefit from hierarchical structuring, whereby individual rules can belong to one or more higher order collections. These *rule sets* aggregate and differentiate rules according to their function within the system. Thus, an RBS is capable of ignoring rule sets that might be irrelevant to a particular problem at a particular time. For instance, such data as who wrote a rule and when would be relevant to testing, evaluation, explanation, and maintenance, but would be irrelevant when the rule was actually being used to make an inference. Typically, each rule exists in several alternative representations or *translations* that suit different purposes. For instance, one machine-oriented translation might serve the need for high-performance at run time, another human-oriented form might use English to support publication and explanation, a third could exploit terseness to make reading and editing easier, and other facets of the rule structure might determine how the inference engine should treat the rule. The system may need to trace rule evaluations and applications, justify a rule's relevance, or even selectively ignore it under certain conditions.

FIGURE 4. The Organization of a Rule into Components to Support Multiple Functions

tions generally fall into one of two categories: changes to working memory or changes to external actions like I/O.

Translations. Nearly all RBSs allow for multiple representations of rules—one representation might be for data entry, another for interpretation, and another for explanations. Typically, all rules are maintained in one preferred representation and translated as needed for other purposes.

Explanations. The hallmark of RBSs has been their ability to explain their conclusions. Explanations have been generated by translating the rules that contributed to a decision into natural language. This requires that a history of working memory changes and their causes be kept that can be searched as needed for explanations.

Although RBSs have been organized in a variety of ways, they all share a basic configuration—they are sets of decisions about what meaning to give rules, and how and when to interpret them. Two organizations are most common: stimulus-driven or *forward-chaining* systems, and goal-directed or *back-chaining* systems. In a forward-chaining system, a rule is triggered when changes in working memory data produce a situation that matches its antecedent component. Some RBSs allow rules to fire repeatedly as long as the working data still match the rule, but most process a specific working memory data configuration only once for each rule. In a back-chaining system, the RBS begins with a goal and successively examines any rules with matching consequent components. These candidate rules are considered one at a time. The unmet conditions of the antecedent are extracted from each plausibly applicable rule, and these conditions are in turn defined as new goals. The back-chaining control procedure then shifts attention recursively toward the new goal. The effort terminates when the top goal has been reduced to a set of satisfied subgoals.

From the point of view of computer architecture, two kernel facilities distinguish RBSs from conventional systems. First, RBSs make heavy use of pattern matching between rule components and working memory. Second, they quickly identify rules that become relevant as working memory changes. This means that there must be a way to access rules by pattern-matched values. Most RBSs meet this need with software, although some current hardware efforts are attempting to improve performance for these tasks. Figure 5 shows a representative sophisticated RBS.

IMPACTS AND APPLICATIONS OF RBS TECHNOLOGY

Evolutionary System Development

RBSs have proved invaluable as a practical means for evolving poorly understood knowledge into a coherent knowledge base. Although today's RBSs are no substitute for a full range of mature data-processing (DP) application-building technology, they do offer a number of unique advantages that are missing from the conventional DP tool kit. We should anticipate that the essential ingredients of RBSs will be imported into DP technology, as the technology matures, to assist in rapid prototyping, improving extensibility, and enhancing software maintenance and support.

Searches

The focus on knowledge in applied AI systems represents a reaction to the unsuccessful attempt to solve important problems using general-purpose or weak methods. As the importance of knowledge became clear, many AI researchers became *knowledge engineers.* These individuals set themselves the task of picking high-value problems with symbolic solutions, identifying corresponding human experts, and debriefing these experts to find out what they knew.

In many cases, expertise is the ability that some people have to use shortcuts and labor-saving techniques that less experienced persons would not know about. Experts are thus able to reduce the equivalent of a large search space for a general problem-solving program to that of the small search space of a specialized, knowledge-intensive program. So although some RBSs do perform searches, most rely mainly on a representation of the problem and chunks of the solution to simplify a task. Search is more of a last resort for these problem solvers. In fact, most RBSs perform little or no search.

Programming

We have already stated that RBS technology requires a rule writer to consider and understand the organization and operation of the target system. Rule writing can in fact be described as a special kind of programming. Like conventional programming, effective rule programming requires mental modeling of state changes, syntactic and semantic checking of rule conditions and execution effects, and heuristic methods for validating and verifying a proposed system. In contrast to conventional programming, however, rule-based programming requires an author to think more analytically than procedurally. Most programmers have some difficulty with this for a few weeks. Instead of first appreciating the relevant goals and heuristic methods and then implementing a corresponding customized problem-solving program, rule-based programmers must first understand the general method of rule-based problem solving and then describe a problem and its related heuristic methods in a form consistent with the available knowledge base and inference engine. This is a different skill entirely. The rule programmer must formulate the heuristics and problem features explicitly, but can depend on the RBS to automate almost everything else necessary for solving the problem.

We should anticipate that the complementary strengths of conventional programming and RBSs will motivate research efforts to bring the two technologies together so that applications will be able to exploit the advantages of both.

THE CONCEPTUAL EVOLUTION OF RBSs

The RBS of today incorporates many influences and has partaken of many related technological developments (see Figure 6). Its essence derives from the *production system* model used in automaton theory and psychology. The basic model was the stimulus–response association presumed by some to underlie all animal behavior. In a similar manner, computing theorists have sometimes found it convenient to describe all computational behavior in terms of state transition tables that define rules for moving between states. Oliver Selfridge's early model, Pandemonium, viewed human signal interpretation activity in terms of the actions of independent pattern-action modules called "demons." Each demon listened to the "shouts" of subordinates that were able to recognize constituent features that were necessary for the demon's higher order percept. After hearing all necessary inputs, the demon shouted its own message, thus indicating the perception of a higher level pattern.

Many researchers have gravitated toward rule-based representations of knowledge for two other reasons. First, rules seem like a natural way to express the situation–action heuristics evident in the thinking-aloud problem-solving protocols of experts. Second, researchers have been able to develop learning procedures capable of inferring rules from experience. RBSs can thus often accept and assimilate newly learned rules merely by incorporating them into the knowledge base.

Specialized RBS architectures have evolved to address different target applications. Each specialty seems to benefit from a slightly different emphasis. Today, special formalisms and supporting systems have been developed in such areas as

1. rule-based programming,
2. rule-based signal understanding,
3. rule-based cognitive simulation,
4. teachable and learnable RBSs,
5. systems for learning rules, and
6. systems for building commercial rule-based expert systems.

By now, the key ideas of RBSs have been incorporated in such other areas of computing as

1. rule-based subsystems for communications architectures;
2. rule formalisms for representing military doctrine, standard policies, and historical precedents;
3. rule-based controlled deduction;

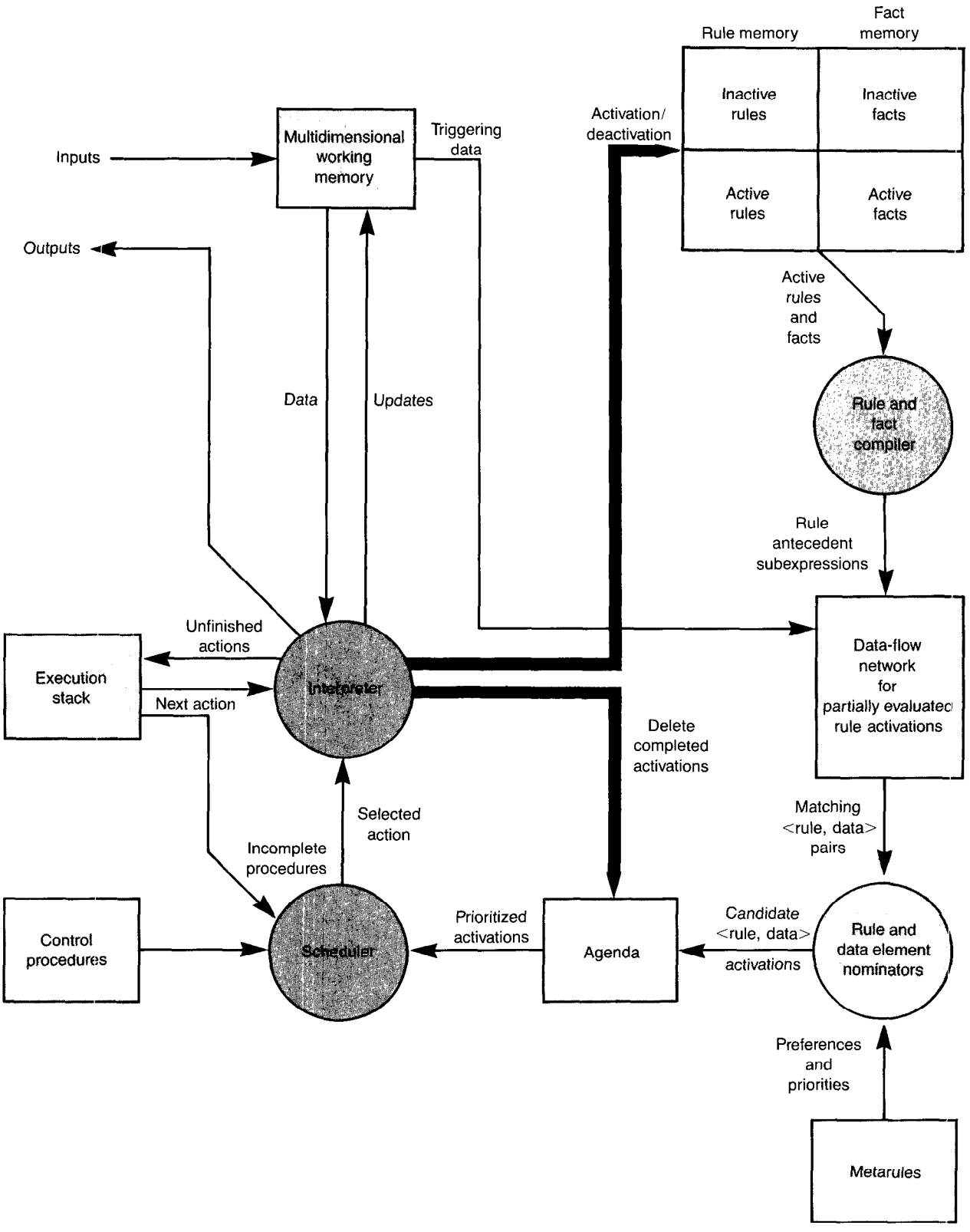


FIGURE 5. A Representative Sophisticated RBS

FIGURE 5. A Representative Sophisticated RBS

The basic RBS is just a knowledge base and an inference engine. The trend, however, is toward more complex and sophisticated systems like the one pictured here. This trend is motivated by two goals: greater language clarity and better run-time performance.

Knowledge clarity has to do with expressibility and intelligibility. Experts must be able to convey their knowledge to an RBS as thoroughly and as efficiently as possible, and the RBS in turn must be able to convey its knowledge and related reasoning to humans. Knowledge clarity also facilitates the modification and extension of knowledge bases. Many of the features evident in the diagram, such as the multidimensional working memory (which distinguishes such dimensions as space, time, or level of abstraction) and the separation of rules from metarules and control procedures, improve knowledge clarity.

The performance goal also motivates many of the embellishments of the sophisticated system. A rule compiler converts the triggering data conditions into a data-flow network that optimizes the computing required to identify executable rules. The sophisticated system exploits several additional mechanisms that help to determine which rule should be executed next. The prioritized list of rules awaiting execution constitutes the *agenda*. Higher level rules known as *metarules* express preferences that can influence the priority of specific candidates on the agenda. The *scheduler* examines the agenda of waiting rules and considers the applicability of any specialized control procedures the system includes. It then selects either a new procedure, a procedure continuation, or the action of a high-priority rule for execution.

4. pattern-directed modules, or macrorules, for distributed architectures and systems of cooperating experts;
5. metarules for heuristic adaptive control of resource-limited systems;
6. rules as a basis for enforcing constraints.

THE EVOLUTION OF RBS TECHNOLOGY

RBS technology incorporates many ideas from diverse sources. A brief and highly simplified recounting of this development follows, focusing on the principal developments in computer science that have most advanced the RBS field.

The starting point was decision tables and compilers. This technology, which emerged about 20 years ago, provided a representation of decision logic for transaction processing and report generation. Decision table entries define condition–action rules that execute sequentially on the current input data. The context effects are immediate because there are no working data. The only knowledge-base entries are the rules, which must represent simple Boolean conditions. The shortcomings of decision tables were, in retrospect, considerable: Large rule tables are quite complex, the rigid order of rule evaluation often proves unsatisfactory, and the inability to describe complex symbolic patterns or to combine intermediate results dynamically limits the range of applications.

Early AI problem-solving languages like PLANNER, which was developed at MIT, provided a way to represent rules within the context of programmable theorem provers. Workers at Carnegie-Mellon were the first to build RBSs with thousands of rules and to develop efficient compilers and translators. One such RBS, known as XCON, became the first expert system to earn a multimillion dollar profit. It was used to eliminate errors in Digital Equipment Corporation VAX orders. The general rule-based programming system known as OPS, which was used for XCON, has since been used for several other RBS applications.

Workers at Stanford developed the MYCIN family of RBSs. MYCIN was the first RBS the expertise of which was acknowledged as such by experts. It was able to perform expert-level subjective reasoning with uncertain data and knowledge and to explain its reasoning in English. A similar system called PROSPECTOR, which was developed by SRI to automate knowledge of mineral deposits, is credited with producing multimillion dollar benefits for at least one mine operation. Subsequent work at Stanford on TEIRESIAS and MRS emphasized metarules for expressing explicit knowledge about control.

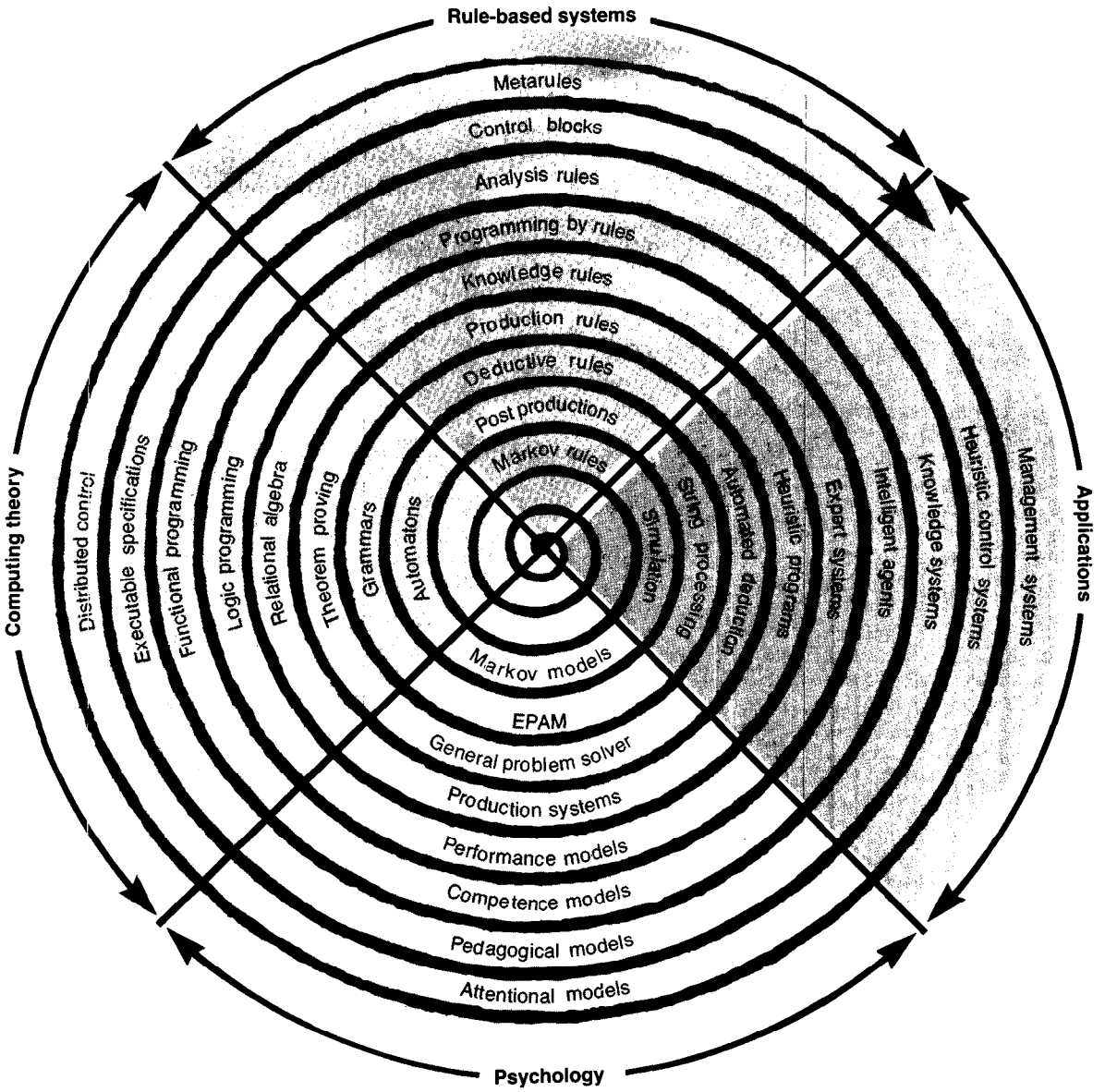
Systems at Stanford and CMU that were originally developed to reason about signal data have evolved to handle large macrorules known as *specialists, knowledge sources*, or *pattern-directed modules*. These systems, among which are HEARSAY-II, HASP, AGE, and BB-1, often pack a great deal of knowledge into a single module. Each module has a condition and an action, and as overall systems, they behave like other RBSs we have considered. They differ from more typical RBSs in using local memory in their computations. In this regard, they have much in common with object-oriented architectures like Smalltalk and Ada® packages.

PROLOG was the first general-purpose logic-based programming language. PROLOG is essentially an RBS that uses stored facts and rules to deduce solutions to goal patterns. It was designed for theorem proving, but has proved attractive for a wider range of AI tasks.

The RITA and ROSIE systems developed by RAND advance the use of RBS methods for conventional programming. These systems blend rule-based program representation with flexible I/O. They are thus very attractive for designers of automated intelligent assistants for computer-based communication tasks.

The M.1® programming system developed at Teknowledge incorporates techniques for tolerating uncertainty and combining evidence in a general-purpose rule-based programming system that operates on an IBM personal computer. M.1 marries the rule-based programming capabilities of PROLOG, RITA, and ROSIE to the evidence-combining capabilities of MYCIN.

Ada is a trademark of the U.S. Department of Defense.
M.1 is a trademark of Teknowledge.



RBS technology is the result of efforts to apply general concepts from psychology and computing theory to the simulation of expertise. This figure, which is in the form of a nautilus, shows how RBS technology, psychology, computing theory, and various application areas are interconnected. Each new development in one area is shown to depend on developments in related areas, as well as on generations of previous developments. Different pathways through the spiral illustrate different historical perspectives on technological history. Tracing the spiral clockwise recapitulates the successive cycles of concurrent activity in the four sectors. Traversing a radial spoke outward from the center recapitulates the successive changes in concept and zeitgeist within a discipline.

At the outset, Markov rules provided a simple technique for defining stochastic processes with probabilistic rules that mapped any current state into its possible successors. Mathematical theorist Emil Post showed how machines that followed simple string-matching condition–action rules could perform all computations. Subsequently, theorem provers were developed for automating deduction. The dual goals of performing human problem-solving tasks and avoiding some of the gross inefficiencies of general-purpose deduction led to condition–action production-rule systems. These often emphasized low-level and detailed activities, though, and so knowledge rules were developed that could embody chunks of expert know-how. With the initial success of knowledge rules for expert systems, many people began developing general-purpose rule-based programming systems. A single rule in these systems could often combine analytical knowledge about a problem domain with control knowledge about ways to achieve problem-solving efficiency. These two forms were subsequently distinguished, with control blocks expressing imperative knowledge in procedural form, and metarules representing other forms of knowledge about knowledge in a rule-based format.

The cognate areas have both supported and adapted to developments in RBS technology. Computing theory has

FIGURE 6. The Evolution of RBS Concepts

```
High-level.problem-solving.approach:
   In order to diagnose and repair a car, follow this procedure:
       C is a car.
       Display the following:
          Welcome to the Car Charging Diagnosis and Repair Adviser.
       Find out about a car called C.
       Determine the initial symptoms of the car C.
       Determine the cause of the problem with the car C.
       Determine the recommendations for fixing the problem
         with the car C.
       Show the recommendations to fix the problem with the car C.
```

S.1 provides a procedural syntax for expressing imperative knowledge. By distinguishing control blocks and rules, the sophisticated RBS enhances intelligibility and produces improved explanations of its lines of reasoning.

FIGURE 7. A Control Block Expressing Procedural Know-How in a Sophisticated RBS

The S.1® expert-system building tool, also developed at Teknowledge, advanced previous RBS technology by differentiating representations for analytic and imperative knowledge. S.1 employs rules for analytic knowledge and procedural control blocks for imperative knowledge. It provides a built-in back-chaining control mechanism with points of escape for user-supplied control blocks. By separating these two forms of knowledge, users can create more intelligible knowledge bases that are capable of providing much improved automated explanations. Figure 7 shows a control block that defines an approach to organizing rule-based reasoning for diagnosing automotive problems. It would employ the automotive diagnostic rules shown in Figure 2.

IMPLEMENTATION AND AVAILABILITY

Table II is a list of supported tools that are currently available for RBSs. Current goals in RBS-related R&D, as defined by the DoD-sponsored Strategic Computing Initiative, are

1. increasing the size of practical rule bases to 10K or more,
2. increasing the speed by two orders of magnitude or more,
3. broadening the set of inference techniques used by interpreters,
4. improving the methods for reasoning with uncertainty,
5. simplifying the requirements for creating and extending knowledge bases, and
6. exploiting parallel computing in RBS execution.

S.1 is a trademark of Teknowledge.

provided a foundation in automatons, grammars, theorem proving, relational algebra, applicative programming styles, executable specifications, and distributed control. Psychology has contributed some extremely general and simple views of intelligent functions (e.g., Markov models, the Elementary Perceiver and Memorizer (EPAM), and the General Problem Solver (GPS)) to successively more knowledge-intensive and elaborated views of cognition. Later conceptual generations distinguish what is known from how it can be applied, taught, or made more efficient. The primary focus of applications has shifted from general-purpose simulation, string processing, and automated deduction, to more pressing, higher value, and knowledge-intensive concerns. Successively, these tasks have emphasized the need for heuristic solutions, specialized expert systems for problem solving, autonomous intelligent agents, knowledge systems for storing and distributing large quantities of institutional knowledge in electronic form, and heuristic systems for adaptive control and other management tasks.

ASSESSMENT: THE REPUTATION VERSUS THE REALITY

Because RBSs have played an important role in demonstrating the importance and practicality of knowledge systems, they have received much attention. It should be pointed out that RBSs are not a panacea, either for DP problems or for AI problems. They do, however, represent a new technology of broad and important applicability and will undoubtedly play an increasingly important role in these fields in years to come.

Today, rule-based components are becoming standard in advanced applications. DEC and NCR have incorporated them in their XCON and OCEAN order-entry and configuration systems. General Motors has undertaken several rule-based expert systems for manufacturing and service functions. Numerous aggressive development programs under way throughout the world, including the Fifth Generation program in Japan, the Alvey program in England, and the Strategic Computing and MCC programs in the United States, all aim to make significant improvements in the performance and generality of this technology over the next five to ten years.

We conclude by listing the key strengths and weaknesses of RBS technology, as well as some long- and short-term objectives. First the strengths:

TABLE II. A Representative Set of RBS Software Products

	Product	Host	Vendor
Tools for commercial development	M.1	IBM PC	Teknowledge
	OPS	VAX	Digital
	S.1	VAX, Symbolics	Teknowledge
Tools for research and experimentation	ART	Symbolics	Inference
	ROSIE	VAX, Xerox	RAND

1. RBSs can represent problem-solving know-how in a manner suitable for application by computers;
2. they modularize chunks of knowledge;
3. they support incremental development;
4. they make decision making more intelligible and explainable;
5. they provide a fertile framework for conceptualizing computation in general;
6. they open new opportunities by providing a non–von Neumann schema that can exploit parallelism in computer systems;
7. specialized RBS architectures have emerged that constrain and simplify application methods;
8. recent advances in RBS technology distinguish imperative and analytic know-how even as they integrate them to produce more effective, cogent, and maintainable knowledge bases;
9. rule-based reasoning can provide a conceptual basis for the analytic formulation of imperative know-how.

RBSs are still without several features that would help to make them more suitable as a general computing approach; specifically, they lack

1. a precise analytic foundation for deciding which problems are solvable,
2. a suitable verification methodology or a technique for testing the consistency and completeness of a rule set,
3. a theory of knowledge organization that would facilitate scaling up without loss of intelligibility or performance,
4. high-grade rule compilers and specialized hardware accelerators, and
5. methods for integrating easily and seamlessly into conventional DP systems.

The near-term research objectives include

1. integration with conventional software systems,
2. modularization and reuse of components, and
3. shareability of knowledge bases among several related applications.

Beyond these near-term objectives, some long-term goals are

1. improved hardware for storage and execution tasks,
2. improved standard software and algorithms for common functions,
3. improved architectures for using metaknowledge efficiently,
4. automatic translation of diverse forms of knowledge into rule form, and
5. optimizing compilers for performing global data-flow optimizations and exploiting multiprocessor opportunities.

Acknowledgments. I am grateful to the many colleagues who have shared their experiences and perceptions regarding RBSs with me. While there are too many people to name, most are at Carnegie-Mellon, Stanford, RAND, and Teknowledge. I am also grateful for the generous support from Teknowledge that has made the contribution of this paper possible.

FURTHER READINGS

The suggested readings span a range from introductory to state of the art. The article by Duda and Gasching is a readable, simple introduction. Erman et al. describes the motivations behind S.1 and the techniques that were used to distinguish analytical rules from imperative prescriptions. Hayes-Roth et al. surveys a variety of issues in representing and implementing know-how; it covers but is not limited to RBSs. Shortliffe describes one of the seminal projects in RBS technology. Waterman and Hayes-Roth surveys the RBS field and the closely related but more general pattern-directed inference systems.

Duda, R.O., and Gaschnig, J.G. Knowledge-based expert systems coming of age. *BYTE 6*, 9 (Sept. 1981), 238–278.

Erman, L.E., Scott, A.C., and London, P.E. Separating and integrating control in a rule-based tool. In *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems* (Denver, Colo., Dec.). IEEE, 1984, 37–43.

Hayes-Roth, F., Waterman, D.A., and Lenat, D.B. *Building Expert Systems.* Addison-Wesley, Reading, Mass., 1983.

Shortliffe, E.H. *Computer Based Medical Consultations: MYCIN.* Elsevier North Holland, New York, 1976.

Waterman, D.A., and Hayes-Roth, F. *Pattern-Directed Inference Systems.* Academic Press, New York, 1978.

CR Categories and Subject Descriptors: I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*deduction (e.g., natural, rule-based)*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; I.2.5 [**Artificial Intelligence**]: Programming Languages and Software
Additional Key Words and Phrases: rule-based systems

Author's Present Address: Frederick Hayes-Roth, Teknowledge, Inc., 525 University Avenue, Palo Alto, CA 94301.