A Review of Knowledge Engineering and Expert Systems: Towards Expert Operators in Forth

Dana Redington [1]

Sleep Research Center
Department of Psychiatry and Behavioral Science
Stanford University School of Medicine
Stanford, California, 94305

Abstract

Knowledge Engineering – a rapidly growing segment of Artificial Intelligence – is transforming the way computers interact with the world. Machines can now mimic highly trained specialists in various fields, hence the designation Expert Systems. The greatest proliferation of Expert Systems will be seen in the development of microprocessor-based knowledge systems or personal expert operators. This paper focuses on personal expert operators; it briefly introduces microprocessor-based expert systems and describes how these systems can be made to "think" and "behave" in real-time; language environments that promote Knowledge Engineering are also discussed with an emphasis on Forth.

Introduction

The increasing computational power of the 16 and 32 bit personal computers provides an opportunity for applying the tools of Knowledge Engineering to real-world problems previously left to larger computer systems. These newer and smaller computers will enable the development of intelligent instruments. Hayes-Roth [HAYE84B] predicts that we can anticipate an emphasis in Knowledge Engineering on intelligent instruments that couple data collection with expert data interpretation. These instruments with machine intelligence can be referred to as *expert operators* and are the focus of this paper. The goals of this paper are to briefly review Knowledge Engineering and Expert Systems as they can be applied with microprocessor technology to real-time applications and introduce personal experts with emphasis on expert operators. Software environments that promote real-time expert systems are also touched upon.

Knowledge Engineering

The rapidly evolving field of Artificial Intelligence (AI) can be divided into several sub-fields [BARR81]. Two important sub-fields are Cognitive Science which is concerned with the science of human intelligence, and Knowledge Engineering (KE) which is concerned with the application of cognitive science to the construction of machine intelligence. Cognitive Science and Knowledge Engineering complement each other. Both sub-fields rely on the computer as a central vehicle for construction and exploration of computational models.

The computer's protean nature is such that it is a flexible machine environment which can be molded and utilized [KAY84]. For Cognitive Science the computer provides a way of understanding the physical basis of mind [PYLY84]. For Knowledge Engineering the computer provides a means of developing machine intelligence applications aimed at the real-world.

[1] Address correspondence to: D. Redington, P.O. Box 620479, Woodside, California 94062.

Intelligence can be emulated in a machine environment. Cognitive Science focuses on primitive cognitive operations [PYLY84]. Knowledge Engineering focuses on how machine programs can utilize the intelligent behavior of humans [NEGO85]. The focus in this paper is on this latter subfield of AI, although knowledge of psychology is essential to KE. (An overview of KE is found in [HAYE84A].)

Knowledge Engineering can be divided into three kinds of activities: knowledge representation, inference generation, and knowledge acquisition [BUCH83; STEF83]. Knowledge representation is concerned with the way data structures are used to depict a small domain of knowledge within a machine. Inference generation is concerned with (software) engines that work on data structures and, as a result, generate new information. Knowledge acquisition is concerned with the translation of domain specific experience of humans into usable machine representations.

Additionally, there are two requirements for Knowledge Engineering [NEGO85]. First, there must exist modularity in a domain specific knowledge. Modular knowledge enables incremental development and revision in a domain base. Second, the primary focus of programming in KE is the construction of knowledgeware in contrast to software. Knowledge programming is directed towards symbolic computational processes, for example, construction of knowledge engines and in establishing the knowledge base. The fundamental use of programming in KE is not to create software, i.e., sequences of machine instructions directed to specific tasks. An example of a computer incorporating a modular knowledge base and knowledgeware is an Expert System.

Expert Systems

Expert systems are knowledge engineered machine programs that solve "real-world" problems usually performed by (human) specialists. They simulate an expert's cognitive process. Some characteristics of Expert Systems are: a problem solving ability near or better than human experts in specific (and restricted) domains of knowledge; a heuristical and symbolic reasoning ability; and, a "natural" interaction with humans [HAYE84B]. They are in a sense the "designer jeans" of computer science [KAY84]; the idiot savants of machine intelligence [BRAC83]. Expert Systems can be divided into two generic classes for the purposes of this article.

The first class of knowledge engineered systems are referred to as expert consultants and are more widely known [DUDA83]. Examples of large Expert Systems include PUFF [AIKI84], Internest-1 [MILL84], MYCIN [BUCH84; DAVI77], DENDRAL, R1, and PROSPECTOR [BARR82]. Examples of personal expert consultants include TK!Solver [KANO84], EXPERT-2 [PARK84A], and DELTA [JOHN83]. Expert consultants emulate their human doppelganger; they converse with a human via standard input (usually a terminal) and eventually derive a solution, although not necessarily in real-time. It is important to emphasize that a consultant does not actually go out into the real-world and collect data first hand but must rely on a human consultee.

The second class of knowledge engineered systems are referred to as expert operators and are less widely known. Examples of expert operators are HEARSAY-I and HEARSAY-II [ERMA80], RC2 [GRUM83], and FORTES [REDI84]. Expert operators differ from robots – microcomputer driven industrial devices that are either programmed to perform a repetitive task, or are remotely controlled by a larger computer not in the environment. Expert operators interface directly to the real-world; they should perform problem solving in real-time and then may act on their solution and modify the environment directly. While it is interesting to interact with a consultant, problems are tackled by operators in the environment; thus, the latter generic class of Expert Systems may ultimately have a more profound impact in the real world.

Structure of Expert Systems

The structure of an Expert System can be divided into three fundamental knowledge parts: a knowledge-base, knowledge-engines, and a knowledge-slate (See Figure 1). Each part of an Expert System is composed of several components. Hayes-Roth, Waterman, and Lenat [HAYE83] indicate that no existing Expert System contains every component. However, in every Expert System some form of a knowledge-base, knowledge-engines, and a knowledge-slate are present. These three parts of a machine expert and their sub-components are briefly described.

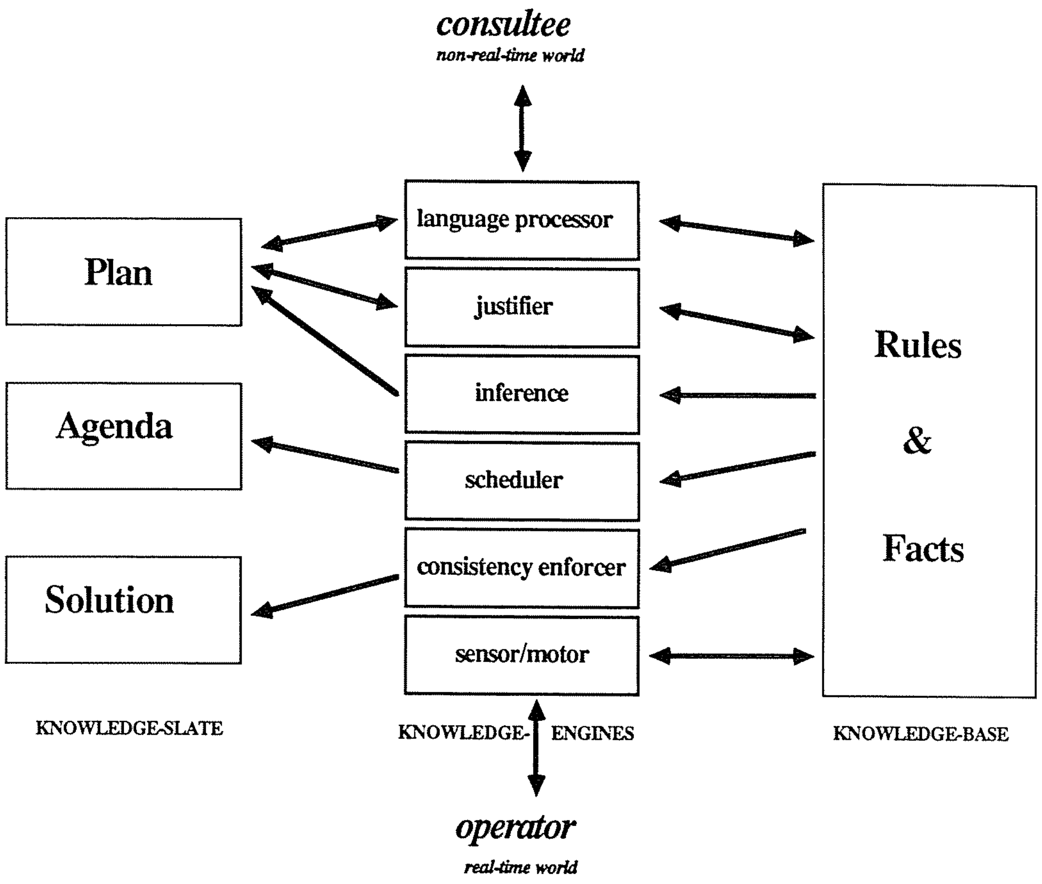

Figure 1. Expert System Structure

The knowledge-base represents the intelligence embodied in the computer and is encoded in plausible reasoning and symbolic manipulation [NEGO85]. The knowledge-base can be further divided into two sub-parts: facts and rules. Facts represent specific information about a domain of knowledge. The rule-base contains heuristics and problem solving rules that aid in planning and in the solution of a problem in a specific domain. The analogy in a human might be the sum total of experience as represented in long term memory [GEOR73; PRIB71].

The so-called knowledge-engines are the software motors that run machine intelligence. Five general engine types are found in an ideal Expert System: a language processor, a justifier, an interpreter, a scheduler, and a consistency enforcer [HAYE83]. One additional engine can be added to these, a sensor/motor engine. The language engine provides "natural" communication between the human user and the computer expert. A justifier engine explains the cognitive behavior of the Expert System. The interpreter or inference engine applies the knowledge-base rules. The scheduling engine regulates the order of rule processing. The consistency enforcer engine modifies previous conclusions as the knowledge-base expands. The sensor/motor engine extends an Expert System into the real world; it senses the world and provides feedback to the world directly. Engines can derive their power from a variety of sources including: synthesis, analysis, evaluation, search, learning, history of previous experience, analogy, parallel processing, and serendipity [LENA84].

The knowledge-slate, referred to as a blackboard [ERMA81], is a segmented workspace where intermediate hypotheses and decisions are kept. The blackboard can be further divided into three components: plan, agenda, and solution [HAYE83]. A plan describes the meta-strategy for the cognitive behavior of the machine expert. The agenda is quite simply a list (maybe a stack) of the most likely future execution behavior of the expert. The final component, the solution, is a slate of candidate hypotheses and record of past cognitive behavior. A direct analogy in humans would be the equivalent of short term memory [GEOR73; PRIB71].

The cognitive activity of an Expert System is built on different knowledge representations, e.g., formal logic, procedural representation, semantic networks, frames and scripts, and production rules [BARR81]. The latter representation scheme is the most well known and is found in Expert Systems, such as MYCIN, HEARSAY, PROSPECTOR [BARR82], and EXPERT-2 [PARK84A].

In production systems the fundamental proposition of a machine expert's thought is the IF-THEN rule construct. Knowledge is represented in a conditional rule where an antecedant – data in short-term memory – is directed to some consequence – an action or a conclusion. The interpreter or inference engine evaluates propositions and in one sense this process can be viewed as machine thinking.

There are two simplistic directions of machine thought or inference mechanisms in a deductive knowledge-based system. The direction depends on how rules are evaluated. The inference engine applies a rule-base to specific cases as those cases arise; the engine selects which rules to evaluate. In the selection of rules the inference engine can be backward driven, a consequent driven inference mechanism. Rules with the current case as the consequence are selected. Each rule is evaluated to confirm that the antecedant portion is present, a process of backward chaining. Examples of consequent driven knowledge-based systems are EMYCIN, AGE [WATE83], and EXPERT-2 in Forth [PARK84A].

An alternative to backward chaining is to establish inference in the opposite direction, i.e., to be forward chained or antecedant (data) driven. Here, the occurrence of a particular situation is used to infer a likely consequence. The forward-chaining inference engine selects one rule to execute at a time based upon the current data frame and previous "cognitive state". Real-time evaluation of a person's wake/sleep state [REDI84] is an example. In this case, a person's physiological data is captured periodically, that is, a particular data frame is grabbed at certain intervals. The data, as an antecedant case, must be found among the set of sleep staging rules; each rule with a resulting state as a consequence is evaluated. In all Expert Systems some form of backward or forward chaining is present. In some cases both directions of chaining are present, e.g., DELTA [JOHN83] or ONCOCIN [BUCH84].

Considerations for Expert Operators

There are considerations for machine expert operators. The size of the computer system is a major concern. A computer must be "big" enough to act intelligently within a reasonable amount of time to avoid the Turing tar pit [KAY84]. The contrast between a consultant and an operator is that the latter must respond in real-time. One measure of a computer's speed from the perspective of Knowledge Engineering is the number of logical inferences processed per second (LIPS). A logical inference is a primitive cognitive process, e.g., the evaluation of a simple IF-THEN proposition. Clearly, a super computer such as the CRAY X-MP, super minicomputers, or microcomputers using custom processors (for example LISP and FORTH machines) are the best candidates for "state-of-the-art" AI applications. One aim of fifth generation computers is to perform in the millions of LIPS range [FEIG83]. However, such computers cannot be directly placed in remote or unprotected environments; they could be used to direct robots. Additionally, the more important problem for AI is to resolve the limitations in Expert System performance due to knowledge acquisition and representation. Thus, ultimately "big" (logical) computers will be limited to a small number of real-world and real-time applications. Custom and microprocessor based personal computers, on the other hand, may open real-world expert applications if they can be made to "think" in real-time, i.e., perform a reasonable number of LIPS.

Another consideration for machine expert operators is the type of programming language environment. Beyond C there are six programming languages that are well-established with a sizable population of programmers: Cobol, Basic, Pascal, APL, Lisp, and Forth. [TESL84]. An AI application can be developed, in principle, with any programming environment, e.g., Basic [AIKI84] – although Basic is not an optimal programming language for implementing Expert Systems: it does not support recursion [DUDA81]. However, the considerations of programming effort, development time, degree of interaction, and execution speed on a microprocessor come into play in selecting the most likely candidates: the latter two languages, Lisp and Forth, are the best candidates and will be examined more closely.

Lisp is the penultimate programming environment for AI applications on "big" computer systems [BARR82; WINS84; WINS81]. Almost every significant knowledge based system has been developed via Lisp or its derivatives [CLAN84]. Lisp is a "list processing" language; both program and data are structured as lists. More importantly, Lisp has only one kind of statement, the function call. As Tesler [TESL84] suggests, Lisp's great source of power is derived from the fact that the value returned by a function can be another function. An additional feature of Lisp is a built-in ability for recursion.

Lisp has not been widely applied on smaller microprocessor systems. This is due primarily to speed and memory requirements. The current evolution of 16 and 32 bit personal computers should promote the usage of Lisp; several micro-computer implementations of Lisp are available [BORT84][2].

A significant disadvantage of Lisp is that AI applications consume a great deal of memory and there is a continual requirement for "garbage collection" – the freeing up of memory. However, vast memory usage may be a requirement for symbol manipulation.

A recent Lisp derivative is a logical programming language, Prolog; it is being used in Europe, Japan, and the United States for AI applications. However, Prolog suffers from two deficiencies [FEIG83]. It uses a first-order predicate calculus to represent knowledge, but knowledge is tersely encoded in Prolog. The problem-solving process in Prolog – this language environment solves problems through evaluation of theorems in its calculus – is, for the most part, hidden from the knowledge user. Thus, Prolog may become a *cul de sac* in the development of Expert Systems as the problems of knowledge acquisition and representation become more highly evolved.

Forth is a concise microcomputer language environment. Programs are not written in Forth, instead an application evolves as words are added to the dictionary in contrast to "subroutines" in Fortran and Basic, "procedures" in Pascal, or "functions" in C. Reading programs in Forth is much like learning a new language for each application; by understanding a dialect, or comprehending the words and their interaction, the Forth application becomes known.

Like Lisp, Forth is ideally suited for AI applications, although for strikingly different reasons; it has been used to implement Lisp [SPEN80; TRAC85]. Forth is an example of an interactive threaded interpretive language [LOEL81]. The history of the system's vocabulary is backward chained with respect to "age of the system". Forth is extensible; the environment is molded to fit an application. In the current case the environment is configured for micro-machine intelligence.

[2] Examples of personal computer implementations of Lisp include: ExperLisp (ExperTelligence, Santa Barbara, CA), Golden Common Lisp (Gold Hill Computers, Cambridge, MA), IQ-Lisp (Integral Quality, Seattle, WA), and TLC-Lisp (The LISP Company, Los Gatos, CA).

Although recursion is not, yet, a standard feature of Forth, an ability for recursion is quite easily installed [GOTS82; GWIL84; SOMM83]; recursion is a candidate for standardization [FORT83][3].

Forth enables near total and direct access to the computer, which may be its most detrimental feature to novices, yet, its most salient ability for the initiated. The rigidity of system software is minimized in Forth; there is the ability to redefine words (procedures) which is not as easily accomplished in other languages, such as Basic or Pascal [REDI83]. This implies that the system may evolve – learning by another name – at the cost of additional memory space. Consequent-reasoning [PARK84B], representation of facts [REDI85], and steps towards natural language parsing [PARK85] demonstrate the extensibility of Forth towards incorporating artificial intelligence. Hofstadter's strange loops aptly describe Forth: "an interaction between levels in which the top level reaches back down towards the bottom level and influences it, while at the same time being itself determined by the bottom level." [HOFS79, p 709]. Finally, Forth is stack oriented, fast, and ideally suited for real-time applications [MOOR74].

The use of Forth as an efficient environment for knowledge-based systems is rarely acknowledged and infrequently applied in AI research.[4] However, Forth becomes the most feasible implementation environment when a knowledge-based system, typically in Lisp, is transferred into a micro-processor system [JOHN83]. The application of Forth to the development of Expert Systems is a fledging area; Table 1 shows generic Expert Systems in Forth: ANIMALS [CASS83], DELTA [JOHN83], RC2 [GRUM83], EXPERT-2 [PARK83A & 83B], MacKit[5], and FORTES [REDI84] (see this Volume). Most applications have been in developing expert consultants; some that are used in the field, e.g., DELTA. Forth also provides a unique opportunity to develop expert operators that interact with the environment in an intelligent way, as in laboratory data collection, e.g., FORTES [see this Volume], and systems that can respond to real time events [DRES85].

Table 1: Generic Expert Systems in Forth.

Name	**Chaining Type**		**System**	**Forth Type**	**Application**
	Forward	**Backward**			
ANIMALS	no	yes	consultant	16-bit	Animal identification
DELTA	yes	yes	consultant	16-bit	Advice on Diesel Electric Locomotive Repair
RC2	yes	yes	operator	16-bit	Preliminary word recognition using HEARSAY architecture.
EXPERT-2	no	yes	consultant	16-bit	Weather prediction example
MacKIT	no	yes	consultant	32-bit	Production Rule Development System.
FORTES	yes	no	operator	32-bit	Sleep staging.

[3] The word RECURSE is part of the main vocabulary in some 32-bit implementations of Forth, e.g., PC-Forth+ (Laboratory Microsystems, Inc., Marina del Rey, CA).

[4] Computer searches on Knowledge Index in COMP1 yields 509 citations indexing Expert Systems; only 4 of these cross index Forth (15 January 1985). Recent computer searches (28 June 1986) in COMP1 yield 2408 citations indexing Expert Systems and 2019 citations indexing Forth and only 10 citations indexing both Expert Systems and Forth.

[5] MacKIT is produced by Knowledge System Environments, Inc., (Dillsburg, PA).

Conclusion

The future of intelligent machine behavior as disseminated by computer software will be influenced by computer language environments. All Expert Systems require a large software kernal upon which the knowledge base is built to satisfy specific applications; the KE kernal for MYCIN has been applied to a variety of related expert applications [FAGA84]. There is very little difference in an Expert System kernal for real-time sleep staging – the laboratory monitoring of a human subject and simultaneous assessment of sleep state – and other applications such as real-time process control [SPEC84] or the monitoring and alignment of laser beams. The large kernal may be relatively stable over different applications in much the same way as Forth is now.

An emerging trend in the development of Artificial Intelligence is to use a large fairly stable kernal: classically a Lisp (-like) environment. However, as smaller although not necessarily less powerful computers develop, other environments will become available. A newly emerging, yet stable, alternative for Knowledge Engineering is Forth. Larger KE applications using classical AI environments have been successfully translated to microprocessor systems using Forth, e.g., DELTA [JOHN83] and RC2 [GRUM83]. Thus, Forth may provide a viable alternative of pursuing machine intelligence and focusing on the development of knowledge-bases for future expert consultants, and more importantly, for the application of expert operators once a KE architecture becomes standardized. Ultimately, a knowledge environment in Forth may process natural language [PARK85] and act in real-time thus enabling true interaction between human and expert operator.

Acknowledgements

I would like to thank Lawrence P. Forsley for his suggestions and for his encouragement of this article. I would also like to thank John Bumgarner, Dave Boulton, and Kim Harris for their continued input, and to Jack Park for our discussions and mutual interest in Artificial Intelligence.

Manuscript received February 1985.

Dana Redington was the first applications software engineer (the "13th" employee) at Apple Computer, Inc., during a year long leave in mid 1977 from his graduate program. He returned to complete a doctorate in Psychophysiology from the University of California, Davis. He has been involved in applying computer technology to mind and body problems mostly in biofeedback and sleep research, since 1978. His interests include applications of microcomputer technology and artificial intelligence to music, martial arts, and to the study of states of consciousness.

References

[AIKI84] Aikins, J. S., Kunz, J. C., Shortliffe, E. H., & Fallat, R. J. PUFF: An expert system for interpretation of pulmonary function data. In: W. J. Clancey & E. H. Shortliffe (Eds), *Readings in Medical Artificial Intelligence, The First Decade*. Menlo Park, California: Addison-Wesley Publishing Company, 1984, pp. 444-455.

[BARR81] Barr, A., & Feigenbaum, E. A. *The Handbook of Artificial Intelligence. Volume 1*. Los Altos, California: William Kaufmann, Inc. 1981.

[BARR82] Barr, A., & Feigenbaum, E. A. *The Handbook of Artificial Intelligence. Volume 2*. Los Altos, California: William Kaufmann, Inc. 1982.

[BORT84] Bortz, J., & Diamant, J. Lisp for the IBM Personal Computer. *Byte*, **9**(7),1984, 281-291.

[BRAC83] Brachman, R. J., Amarel, S., Engelman, C., Engelmore, R. S., Feigenbaum, E. A., & Wilkins, D. E. What are expert systems? In: F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), *Building Expert Systems*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1983, pp. 31-57.

[BUCH83] Buchanan, B. G., Barstow, D., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., & Waterman, D. A. Constructing an expert system. In: F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), *Building Expert Systems*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1983, pp. 127-167.

[BUCH84] Buchanan, B. G., & Shortliffe, E. H. *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley. 1984.

[CASS83] Cassady, J. "Expert systems using Forth." paper presented at the Fifth FORML Conference, November 23-25, 1983, Asilomar, California.

[CLAN84] Clancey, W. J., & Shortliffe, E. H. Medical artificial intelligence programs. In: W. J. Clancey & E. H. Shortliffe (Eds), *Readings in Medical Artificial Intelligence, The First Decade*. Menlo Park, California: Addison-Wesley Publishing Company, 1984, pp. 1-17.

[DAVI77] Davis, R., Buchanan, B. G., & Shortliffe, E. H. Production rules as a representation for a knowledge-based consultation program. *Artificial Intelligence*, 1977, **8**, 15-45.

[DRES85] Dress, W., B. A Forth implementation of the heap data structure. *Journal of Forth Application and Research*. **3**(2), 1985, 135-138.

[DUDA81] Duda, R. O., & Gaschnig, J. G. Knowledge-based expert systems come of age. *Byte*, **6**(9), 1981, 238-278.

[DUDA83] Duda, R. O., & Shortliffe, E. H. Expert systems research. *Science*, **220**(4594), 1983, 261-267.

[ERMA80] Erman L. D., Hayes-Roth, F., Lesser, V., & Reddy, D. Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, **12**(2), 1980, 213-253.

[ERMA81] Erman, L. D., London, P. E., & Fickas, S. F. The design and an example use of HEARSAY-III. *IJCAI*, **7**, 1981, 409-415. (cited in Hayes-Roth, Waterman, and Lenat (Eds) [1983]).

[FAGA84] Fagan, L. M. Shortliffe, E. H. & Buchanan, B. G. Computer-based medical decision making: from MYCIN to VM. In: W. J. Clancey & E. H. Shortliffe (Eds), *Readings in Medical Artificial Intelligence, The First Decade*. Menlo Park, California: Addison-Wesley Publishing Company, 1984, pp. 241-255.

[FEIG83] Feigenbaum, E. A., & McCorduck, P. *The Fifth Generation*. Menlo Park, California: Addison-Wesley Publishing Company. 1983.

[FORT83] Forth Standards Team. *Forth-83 Standard*. Mountain View, CA: Forth Standards Team. 1983.

[GEOR73] George, F. H. *The Brain as a Computer*. New York: Pergamon Press. 1973.

[GOTS82] Gotsch, B. The art of recursion. *Forth Dimensions*, **4**(2), 1982, 24-26.

[GRUM83] Grumbach, A., & Ducasse, M. RC2 - A new expert system to interpret signals. *Informatique et. Gustion*, **147**, 1983, 76-84.

[GWIL84] Gwilliam, M., & Zammit, R. Recursion of the Forth kind. *Forth Dimensions*, **5**(5), 1984, 28.

[HAYE84A] Hayes-Roth, F. The knowledge-based expert system: A tutorial. *Computer*, **17**(9), 1984, 11-28.

[HAYE84B] Hayes-Roth, F. Knowledge-based expert systems. *Computer*, **17**(10), 1984, 263-273.

[HAYE83] Hayes-Roth, F., Waterman, D. A., & Lenat, D. B. An overview of expert systems. In: F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), *Building Expert Systems*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1983, pp. 3-29.

[HOFS79] Hofstadter, D. R. *Godel, Escher, Bach: An Eternal Golden Braid*. New York: Basic Books. 1979.

[JOHN83] Johnson, H. E., & Bonissone, P. P. Expert system for diesel electric locomotive repair. *Journal of Forth Application and Research*, **1**(1), 1983, 7-15.

[KAY84] Kay, A. Computer software. *Scientific American*, **251**(3), 1984, 53-59.

[KANO84] Kanopasek, M., & Jayaraman, S. Expert systems for personal computers. *Byte*, **9**(5), 1984, 137-156.

[LENA84] Lenat, D. B. Computer software for intelligent systems. *Scientific American*, **251**(3), 1984, 204-213.

[LOEL81] Loeliger, R. G. *Threaded Interpretive Languages*. Peterborough, New Hampshire: Byte Books. 1981.

[MILL84] Miller, R. A., Pople, H. E., & Myers, J. D. INTERNIST- 1, an experimental computer-based diagnostic consultant for general internal medicine. In: W. J. Clancey & E. H. Shortliffe (Eds), *Readings in Medical Artificial Intelligence, The First Decade*. Menlo Park, California: Addison-Wesley Publishing Company, 1984, pp. 190-209.

[MOOR74] Moore, C. H. Forth: A new way to program a mini-computer. *Astronomy and Astrophysics Supplement*, **15**, 1974, 487- 511.

[NEGO85] Negoita, C. V. *Expert Systems and Fuzzy Systems*. Menlo Park, California: Benjamin/ Cummings Publishing Company, Inc. 1985.

[PARK83A] Park, J. "Expert systems in Forth". Paper presented at the Fifth FORML Conference, November 23-25, 1983, Asilomar, California.

[PARK83B] Park, J. *MVP-Forth Expert System Tool Kit*. Mountain View Press. 1983.

[PARK84A] Park, J. Expert systems and the weather. *Dr. Dobb's Journal*, **9**(4), 1984, 24-31.

[PARK84B] Park, J. A consequent-reasoning inference engine for microcomputers. Paper Presented at the Sixth FORML Conference, November 23-25, 1984, Asilomar, California.

[PARK85] Park, J. An approach to natural language parsing. Paper presented at the Seventh FORML Conference, November 29-December 1, 1985, Asilomar, California.

[PRIB71] Pribram, K. H. *Languages of the Brain: Experimental Paradoxes and Principles in Neuropsychology.* Englewood Cliffs, New Jersey: Prentice-hall, Inc. 1971.

[PYLY84] Pylyshyn, Z. W. *Computation and Cognition.* Cambridge, Massachusetts: Bradford Books, MIT Press. 1984.

[REDI83] Redington, D. Stack-oriented co-processors and Forth. *Forth Dimensions,* **5**(3), 1983, 20-22.

[REDI84] Redington, D. Outline of a Forth oriented real-time expert system for sleep staging: A FORTES polysomnographer. Paper presented at the Sixth FORML Conference, November 23-25, 1984, Asilomar, California.

[REDI85] Redington D. Knowledge representation in Forth: "What is a fact?" ... Well, it depends on the time..." Paper presented at the Seventh FORML Conference, November 29-December 1, 1985, Asilomar, California.

[SOMM83] Sommers, R. W. Vectored execution and recursion. *Forth Dimensions,* **5**(4), 1983, 17.

[SPEC84] Spector, A. Z. Computer Software for process control. *Scientific American,* **251**(3), 1984, 175-186.

[SPEN80] Spencer, J. A Lisp implementation in Forth. Paper presented at the FORML Conference, November 26-28, 1980, Asilomar, California.

[STEF83] Stefik, M., Aikins, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F., & Sacerdoti, E. The Architecture of expert systems. In: F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), *Building Expert Systems.* Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1983, pp. 89-126.

[TESL84] Tesler, L. G. Programming languages. *Scientific American,* **251**(3), 1984, 70-78.

[TRAC85] Tracy, M. J. A Forth Lisp. Paper presented at the Seventh FORML Conference, November 29-December 1, 1985, Asilomar, California.

[WATE83] Waterman, D. A., & Hayes-Roth, F. An investigation of tools for building expert systems. In: F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), *Building Expert Systems.* Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1983, pp. 169-215.

[WINS84] Winston, P. H. *Artificial Intelligence, 2nd ed.* Menlo Park, California: Addison-Wesley Publishing Company. 1984.

[WINS81] Winston, P. H., Klaus, B., & Horn, B. *Lisp.* Menlo Park, California: Addison-Wesley Publishing Company. 1981.