Rule Base Management Using Meta Knowledge

Mehdi T Harandi
Thierry Schang
Seth Cohen

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W Springfield Ave , Urbana, IL 61801

ABSTRACT

This paper describes the rule base management strategy of an expert system environment. The environment includes a set of integrated tools which facilitate acquisition, manipulation and maintenance of knowledge. The rule base management component of the system, called RBM, assists these tasks by organizing global semantic information within the rule base. RBM extracts this semantic information from the texts included in "rule structures" and builds a semantic network of the concepts found in the rule base. The rule base is then divided into rulesets which are clusters of rules that refer to the same atomic concept. Construction of this meta knowledge is achieved through a keyword matching mechanism. The paper includes a brief description of the RBM system, the dictionary it uses for building meta-level knowledge, and its keyword matching technique.

INDEX: Rule Base, Knowledge Representation, Knowledge Management, Meta-level Knowledge, Keyword Matching.

This work is supported in part by a grant from the International Business Machines, Palo Alto Scientific Center
Thierry Schang is now affiliated with THOMSON-CSS, France
Seth Cohen is now affiliated with Teradyne Inc , Boston, MA

Introduction

It has become clear that the success of expert systems is more a function of the power of the representation and richness of the rule base than the choice of control schemes Emergence of languages especially suited for expert system construction ([1], [2], [3]) has pushed control scheme technology far ahead of knowledge acquisition technology in expert systems While formalizing the inference mechanism behind an expert system, such languages fail to address the question of how to effectively and efficiently construct and manage a knowledge base Systems which exhibit intelligent behavior do so only after many man-hours are spent building and fine-tuning the rule base

This paper discusses the rule base management facilities of an expert system environment called GPSI, originally designed for use within a programming environment to aid programmers in debugging [4] The GPSI environment provides two tools for rule base construction The first of these tools, the Network Manager (NETMAN), is geared toward rule creation, modification and testing The second tool, the Rule Base Manager (RBM), is intended for rule base management and maintenance This rule base management component of the system is used as a high-level utility in all phases of development and use of an expert system A description of the various components of the system can be found in [5] The knowledge representation scheme of the system is discussed in [6]

The GPSI rule base is composed of three types of knowledge structures *rules, classes,* and *parameters* Rules constitute the largest portion of the rule base Each rule, represented by a tree structure, contains information about how evidences support or weaken a hypothesis The root of a rule tree represents a main hypothesis, or a subgoal of one or more higher level goals The leaves of each tree are either evidences or references to other inference structures Interior nodes represent intermediate hypotheses as well as various operative and connective functions The description of each node consists of a node label, its type and a list of

associated attributes An attribute associated with many node structures is a descriptive text This text is used by the rule-base manager of the system, RBM, to extract important keywords and concepts and to construct meta-level knowledge

Evidences whose set of supporting facts fall within a shared set of facts can be grouped into a class Through a class query the system obtains facts about an entire class of evidences Parameter structures provide flexibility in dialogue management Descriptive texts are also associated with *class* and *parameter* structures Similarly, these texts are used by RBM for construction of meta knowledge

Rule Base Management

In its basic form, a knowledge base is a collection of rules designed through dialogues between a domain expert and a knowledge engineer or by domain experts themselves Although such rules are physically independent pieces of knowledge, they are highly interrelated Some rules are applicable only if others have been previously fired, several rules may need the same information in order to fire, or several rules may infer the same conclusion These relations are implicitly stored in every part of the rule-base As the knowledge base grows larger, the knowledge engineer will be unable to remember what the rule base contains (i e what the system knows) He will then be faced with three major problems inconsistencies, redundancies, and incompleteness

A rule base is inconsistent when conflicting diagnosis can be inferred from the same set of evidences Consistency is the most crucial problem found in a rule base because it can result in faulty behavior of the expert system It can sometimes be cured by removing part of a rule but it also can require major reorganization of a portion of the rule base Redundancies occur when the same diagnosis can be inferred from a set of evidences by using different rules The rule base is called incomplete when it fails to reach a conclusion from a given set of evidences

Knowledge base management is responsible for discovering and manipulating the relations that exist between elements of a rule base and explicitly representing them in a new structure By imposing new structures on the rule base, knowledge base management gives the system some knowledge about its own rules, thus allowing the system or the knowledge engineer to access or check only the small part of the rule base that is of interest at a specific moment, when a specific problem is encountered

Furthermore, when repeated structures are used in the same context, a rule-model can be deduced for use as an expectation when new rules are entered by the knowledge engineer This capability allows the system to double check a newly designed rule If an inconsistency is found between the rule and the rule-model, the system can inform the user of this fact and ask either for a confirmation or for a complete or partial redefinition of the rule This self-knowledge is commonly called *Meta-Level Knowledge* because it generally consists of creating one or more new layers of knowledge above the rule base

TEIRESIAS [7,8], the knowledge acquisition module for the MYCIN expert system [9], builds and maintains a three-level hierarchical structure The lower level, called *Object-Level Knowledge*, includes a description of the objects in the domain The second level of knowledge covers the notions of rules, attributes or predicate functions This level also contains automatically constructed rule models to group together rules which have some similarities The third level of knowledge deals with the knowledge representation itself in *data structure schema* This level is constructed as another knowledge-based system which helps the acquisition of conceptual primitives at the *Object-Level Knowledge*

In CENTAUR, a MYCIN-like expert system [10], knowledge representation is in the form of production rules augmented with prototypes which form the domain knowledge The main idea underlying prototypes is explicitly defining the context in which rules can be applied

Rule Base Management In the GPSI Environment

The task of RBM is twofold First it will organize global semantic information within the rule base This involves manipulating a semantic network containing information about the rule base itself The second task of RBM is to function as a rudimentary bookkeeping system, collecting statistics on rule base constituency, acceptance and rejection rates of rule conclusions, and determining the utility (number of references) of each rule The latter, however, is not discussed in this paper

RBM, unlike TEIRESIAS, can extract semantic meaning from the rule-base itself The structure that embodies this semantic meaning is not constructed by a domain expert Its eventual shape is the result of internal construction that is totally dependent on the innate form of the rule base In other words, RBM *inductively* creates rule base's hierarchies of meaning

This is achieved by building a semantic network of the concepts found in the rule base text The structure of the network is derived from the interrelations of the concepts as discovered by RBM The rules are divided into *rulesets* which are clusters of rules that refer to the same *atomic concept* An atomic concept is defined as a "chunk" of domain specific semantic information For example, in a program debugging rule base, it might be a domain-specific keyword like *character* representing a

lexical token, or *end*, representing a syntactical delimiter It might also be a domain specific action like *add*, *remove*, or *modify*, as used in suggestions for error repair These concepts are designated by the domain specialist who also assigns an associated weight of importance for each concept Shared substructures are recognized and lead to the hierarchical structuring of the rulesets themselves An example drawn from the Pascal programming domain is shown in figure 1

This hierarchical structure is then put to good use First of all, it is used to narrow the search space of the inference engine Just as database management systems often use hierarchical index structures to limit the number of blocks that must be accessed to find a given record, RBM's ruleset structure reduces the number of rules that GPSI has to examine RBM determines the relevant concepts that are being referred to by a given query Only a search of those rules relating to these

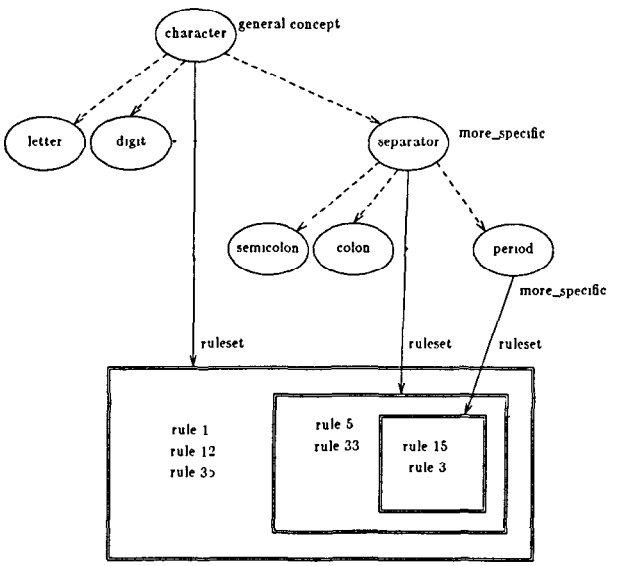

Fig 1 Ruleset Structure

concepts, and all of the substructures they are associated with, is required GPSI navigates the inference network to find out which ruleset structures are to be examined in greater detail All this leads to improved response time and lower memory requirements by the inference engine

Secondly, levels of understanding have been introduced into the knowledge base This extra knowledge is used to *understand* user queries by associating attributes of an entire ruleset with a specific concept The semantic network is also used by GPSI's help facilities to explain to the user the line of reasoning it followed in reaching a conclusion

Construction of meta-knowledge is achieved through a keyword matching mechanism Using a dictionary of domain keywords, initially created by the knowledge engineer and thereafter updated whenever necessary The text associated with query generating and hypothesis reporting structures are parsed to determine important concepts and keywords This knowledge is then used to build the rule hierarchy For each domain keyword in the dictionary there will be a list of rules which reference it

Although some ideas used in GPSI's RBM are derived from the TEIRESIAS and CENTAUR systems, major variances exists due to the dissimilarities between the GPSI and MYCIN environments The main difference between GPSI and MYCIN resides in the knowledge representation used

The knowledge representation in the MYCIN environment follows a *situation, action* pattern where the *situation* and *action* both consist of *clauses* constructed from *predicate functions* whose arguments are *associative triples* (attribute, object, value) The rigidity of this structure is exploited by TEIRESIAS to extract the semantic of the rules by having some meta-knowledge about attributes and objects In the CENTAUR system, the prototypes are included as part of the knowledge base and must therefore be designed and structured as the rules are by the knowledge engineer

In the GPSI knowledge base, much of the semantic information of rules are stored in the text attribute associated with goal and evidence nodes From this text, written by the domain expert or knowledge engineer in free format natural language, we first extract the concepts and then structure rules into a new level of knowledge, based on shared concepts

The structures that RBM builds resemble CENTAUR's prototypes It organizes the rule-base into *rulesets* which correspond to a particular *context* However, the notion of *context* is here extended to the idea of *concepts* Rules are not any more grouped according to the context in which they appear but according to the subjects they are dealing with, without any restriction on the context

The RBM Dictionary

The dictionary is divided in two parts The keyword entries and the concept entries This separation allows the knowledge engineer to easily define synonyms and homonyms as several words may map to the same concept or one word may map to several concepts

The keyword dictionary consists of one entry for each word (or group of words which cannot be separated) that might be encountered in the rule base Standard suffixes like -ing, -s, -ed, -ies, -ves, are

automatically detected and therefore, the same entry is used for gerund, participial, or plural form However, one entry is created for each possible form of irregular verbs and plurals

Each word considered important in the domain is linked to one or more concepts which represent all of its possible meanings Synonym recognition is handled by having each synonym linked to the same concept In addition, a word may be linked to different concepts depending on its grammatical property --e g verb, noun, etc This is especially useful when a noun and a verb have the same spelling but do not represent the same concept

The concept dictionary is the main interface between the keyword matcher and the ruleset builder and is accessed from the keyword dictionary It also is the main structure storing semantic information about the domain A concept might be isolated and just referred to by some keywords But, most of the time, a concept is integrated into a semantic network of interrelated concepts These relations are called *more_specific*, *more_general*, or *negation* relations The structure in figure 2 represents the net for all concepts having a relationship with *character* The arrows represent *more_specific* relations in the forward direction and *more_general* relations in the backward direction Note that the *separator* concept has more specific instances which are *semicolon*, *colon*, and *period* because these specific characters are themselves significant in a Pascal program The *letter* concept does not have more specific instances because no specific letter *a,b, ,z* has a meaning on its own

Figure 2 shows how the keyword and concept dictionaries are linked together Rectangular boxes represent a keyword entry, ellipses represent a concept entry, dashed arrows are *more_specific/ more_general* relations between concepts and normal arrows are links between the two parts of the dictionary The entry *modify* is used to recognize *modify, modifying, modifies and modified*

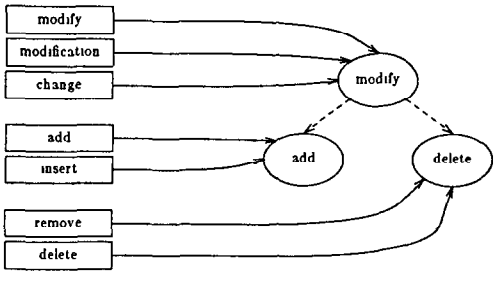

Fig 2 Dictionary Example

Dictionary Acquisition

Before any process takes place, **RBM** has a reduced dictionary containing the common unimportant words of the English language This includes articles, pronouns, prepositions, and conjunctions As one of the constraint of the system is to be domain independent, no further assumption is made without the interaction of the knowledge engineer It is his sole responsibility to first initialize and then update the dictionary with the relevant keywords that capture the jargon of the domain This task however is closely monitored by the Dictionary Acquisition Unit (DAU)

Whenever an unknown word is found by RBM, the ruleset construction process is suspended and DAU steps in to update the dictionary First, a question is asked to check if the word is accidentally misspelled If it is not, the user is asked to provide the standard form for the word and its irregular forms if any (e g -- irregular verb or unusual plural) This phase ends by asking whether the word is important in the domain or not If the answer is no, DAU stops the questioning and stores the word in the keyword dictionary without creating an entry in the concept dictionary If the answer is yes, the system starts the second phase to define the concept associated with that word If a known synonym can be found in the dictionary, the new word will be linked to the existing structure by a new entry in the keyword dictionary Otherwise, the user is queried to determine the concept associated with the word and to integrate it in an existing or new structure

At several stages, the user must define the new word by its relations to other words in the domain There again, the user might refer to an unknown word In that case, DAU is called recursively (avoiding circular calls) This scheme allows the user to define a whole structure at once and not word by word as they appear in the rule base **Ruleset Construction**

A ruleset is defined for each concept in the dictionary and consists of the list of rules which refer to that concept When dealing with rulesets, the *more_specific/more_general* links become *subset/superset* relations Figure 1 shows the structure of the ruleset when some subset/superset relations are present Each ruleset can be further partitioned by cross-referencing the rule-base Each such partition therefore may contain only rule-nodes of a certain type This additional division is useful when the knowledge engineer wants to focus its attention only on particular knowledge structures such as goal or evidence nodes

Ruleset construction can be viewed as a pipelined process as shown in figure 3 This process starts by obtaining the text messages associated with the rule base structures First, a keyword matcher parses the text and with the help of the dictionary, returns a

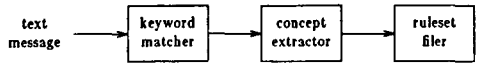

Fig 3 Ruleset Construction Process

tokenized version of it This list of tokens is then processed by the Concept Extractor whose role is to find the right concept for each important keyword The rule processed is then added to the rulesets associated with the concepts found

Most of the time, a keyword will map to a unique concept It is, however, possible that several concepts are found for a single keyword In this case, an attempt is made to resolve the conflict by looking ahead in the sentence to find other words which refine the meaning of the ambiguous word For example, when *end* is read in, several concepts might be triggered, if the word *line* can be found in the sentence, the conflict can be resolved in favor of *end_of_line* When such an attempt fails, a local syntactic analysis is used to determine the grammatical category of the word --e g verb, noun, etc Depending on this category, the number of candidate concepts is further reduced Finally, if still several concepts remain, we consider them all as correct This results in the inclusion of the rule in one or more possibly non related rulesets, but, would not result in any loss of information

When some ambiguities persist in the choice of a meaning for a word, knowing the grammatical category for the current use of the word is usually enough to disambiguate it That is because in a specific domain, most of the keywords have a unique important meaning However, this word might be used in a more common English sense with another grammatical category For example, *END* is a keyword and concept as a reserved word of the Pascal syntax But as a verb, *end* is however not fundamental and should be ignored This distinction can be made if the text is parsed grammatically Our experiments have shown that the ambiguity is most of the time localized That is, usually ambiguous words will be surrounded by words whose grammatical classifications are not ambiguous, i e -- uniquely defined

For example, in the sentence *"An END must follow "*, having an article in front of *END* will eliminate the possibility for *END* to be a verb In our dictionary, the word *end* is only known as noun or verb thus the meaning chosen will be the one associated with the grammatical category *noun* We claim that most conflicts can be resolved through local parsing only

The idea behind the local parser is to scan one word backward and one word forward and eliminate alternatives that are not possible in the English language Only consistent patterns will remain and probably only one

The local parser has a list of impossible grammatical constructions and another list of acceptable constructions The latter expresses grouping of words that if present are immediately considered correct In a preliminary experimentation with raw English text, 80% of the ambiguous structures were resolved by this method for a very low computing cost

Once keywords and concepts are extracted from the text, the ruleset updation begins The rule under consideration is added to all rulesets related to the concepts found It should be noted that if several related concepts appear in a sentence, the corresponding rule is only stored in the ruleset of the most specific concept, it can however be easily accessed from any more general concept by tracing down the *more_specific* links

Object Identification Through Keyword Matching

The task of keyword matching and ruleset formation is performed by the "Object Identifier" component of RBM The Object Identifier is divided into three modules each module responsible for talking to a different source of information This is illustrated graphically in figure 4

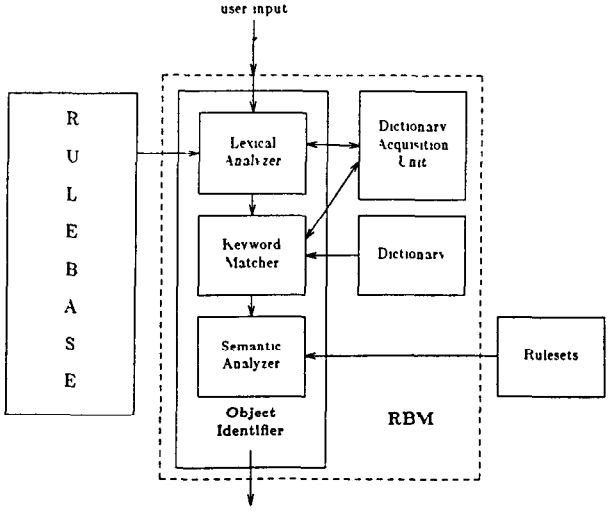

Fig 4 Object Identifier

The first module is the *Lexical Analyzer* and it does just that -- it finds the next legitimate word on a line of text The source of this text may be the fixed rule base , or interactive user input (either from the domain expert or the end-user of the system) The lexical analysis is identical for either source of input One of the chief responsibilities of this phase is the creation of a *superimposed code word* which uniquely identifies each word The end results of this phase are passed on to the

Keyword Matcher

The Keyword Matcher talks to RBM's dictionary and tries to find a match between every legitimate word found by the Lexical Analyzer and an entry in the dictionary A "match" means that the word is *known* by the system Unmatched words must be defined through an interactive dialogue with the user before any further progress can be made Once the system finds that it knows all the words in the text, the process proceeds towards '*understanding*' Each concept has a pointer to its associated ruleset that contains all rules or other structures in the rule base which refer to that concept in some way A list of pointers to these rulesets is passed to the Semantic Analyzer

The Semantic Analyzer must determine which is the rule or structure that is best described by the analyzed text, if any It does this by calculating a *probable match weight* for each rule in all of the rulesets passed to it Every concept has an importance weight assigned to it by the domain specialist The ruleset associated with a particular concept acquires the importance weight of that concept as well The probable match weight for a rule R is calculated as the summation of all weights for every ruleset that includes R as a member, divided by the summation of all possible weights Therefore, a rule which appeared in every ruleset would have a probable match weight of 100%

Word Recognition

Deciding whether or not there is a match between a token in hand and a word in the dictionary requires a search through a potentially large database This search will translate into either time or space or both on the computer These concerns are central to any database system design It should also be recalled that the *important* computation only begins *after* the appropriate record is found in the database We therefore sought a word recognition algorithm that would take these considerations in mind, and that did not require either character string manipulations or linear searches through the dictionary

Using the work of Tenczar [11] and Roberts [12] as a point of departure, an algorithm for indexing keywords using superimposed code words was evolved Superimposed coding is a technique similar to hashing and shares the latter's improved execution speed over straight sequential search The major portion of the method can be thought of as a fast preselection algorithm which is invoked prior to beginning a sequential search The basic idea is to map attributes into random k-bit codes in an n-bit field, and to superimpose the codes for each attribute present in a record

A superimposed code word (hereafter abbreviated to **scw**) is the end result of a repetitive OR'ing together of "binary code words" (hereafter abbreviated to **bcw**) A bcw of width b bits exists for every attribute of a word All bcw's are initially set to zero Then, using a pseudo-random number generator seeded with the key's value, k bits of the bcw are set to one where $1 <= k <= b$ Every item in the database has a scw associated with it These are stored as a separate file

A scw is constructed for every query, and then the scw file is compared to the query-mask Q by the test if $S_i \wedge Q = Q$ then "match", where S_i is the current item in the scw file If there is a match, then the database item that corresponds to S_i is retrieved from secondary storage and compared directly with the query

In order to use the method of superimposed codes a key-to-bcw mapping function Θ(K) must be defined, which is fast and flexible enough to detect near-misses, misspellings, and a wide range of synonym alternatives Our method of mapping the bcw and of building up the scw uses multiple criteria to set the bits in a code word The more criteria for characterizing words, the greater the likelihood of achieving a near unique mapping for each word in the dictionary Our discriminating mapping function uses the following word attributes

Length and First Character The length is a useful observation in the context of comparisons -- one can easily determine if two character strings are different by simply checking their length field The first letter of a word is a very important one for discrimination purposes, most especially for sorting The code for the length is superimposed onto that of the first character

Letter Content A different 16 bit code is created for each word depending on the letters and digits that are found to make up that word This code is determined partially by phonetics ('s' and 'z' set the same bit), partially by data from studies of common English letter usage (as 'e' is the most common letter found in English its inclusion in a word will not be as strong a discriminating factor as 'q', say), and partially by letter positioning on the typewriter keyboard (to account for mistyping) This code is calculated independently of the order in which the letters or digits appear This field will differentiate between similar words that differ only in one letter ("boy", "coy"), but not between palindromes ("evil", "live"), for example

Letter Order It is in this field that the relative ordering of the letters in the word, i e , the spelling, is taken into consideration Bits are set depending on the digraphs (adjacent letter pairs) present in the word

Pronounciation Although it is difficult to capture in one algorithm the number, sound, and stress of syllables in a word, some attempt was made to encode this information It is possible, for example, to set it so that

"ph" always maps to the bit code for "f", "kn" to an "n" and "ch" to a code that is close but not identical to the code for a hard "c", and so on

A table look up is used for each digraph in the word All fields except the length field can be set simultaneously during the lexical analysis by performing only n table look ups where n is the number of characters in a word

The final scw is created by OR'ing some fields and EXCLUSIVE-OR'ing others Our decision about which Boolean function to use to superimpose these fields was a matter of trial and error We choose a function which produced the greatest differentiation in values The result is a 16 bit integer that reduces the number of false drops that must be compared in linear fashion to, on the average, 3% This compares favorably with the 5% rate reported by Tenczar and Golden using a 41 bit algorithm [11]

We also use this same mapping technique as the basis for the hashing function that places the words in the dictionary The hashing function is simple and fast and uses as its key value the scw already calculated for the word The 97% differentiation rate keeps the collisions in the hash table to a minimum

Conclusion

Knowledge base management systems are an essential part of any expert system environment The main function of such a system is to discover interrelationships between different bodies of knowledge and structure the rule-base according to these relations This structure then can be used to support user queries concerning the contents of the rule-base, to enhance the performance of the inference engine, and to prevent or help the discovery of inconsistencies and redundancies in the rule base

In this paper we presented the design and functional aspects of such a rule base management system Using a keyword matching strategy, this tool extracts implicit semantic knowledge out of the rule base and makes it explicit through ruleset structures This meta-level knowledge provides easy access to any part of the rule base, hence enhancing the capabilities for writing, debugging and maintenance of the rules

Early experimentations with RBM have shown that the average number of concepts found in a rule is 5 For a subset of 150 rules, 60 concepts where defined, 22 of which were stand alone with no relation with other concepts The remaining 38 concepts were organized in 5 semantic nets of various sizes The number of concepts is not likely to increase drastically with the expansion of a rule base because most of the important concepts of the domain are usually defined early in the process

References

1 Clark, K L and McCabe, F G , "Prolog a language for implementing expert systems," in Machine Intelligence 10, eds D Michie and Y H Pao, 1982

2 Forgy, C L , "OPS4 User's Manual," Technical Report CMU-CS-79-132, Carnegie-Mellon University, Department of Computer Science, 1979

3 Bobrow, D G , Stefik, M The Loops Manual, XEROX PARC, 1983

4 Harandi, M T , "Knowledge-based program debugging A heuristic model", Proceedings of IEEE Conference on Software Development Tools, Techniques, and Alternatives (SOFTFAIR-I), Arlington, VA (1983)

5 Harandi, M T , " The architecture of an expert system environment", Proceedings of the 5th International Workshop on Expert Systems and Their Applications", Avignon, France (1985)

6 Harandi, M T , "A tree-based knowledge representation scheme for diagnostic expert systems," Proceedings of the 2nd Conference on Intelligent Systems and Machines, Rochester, MI , (1984)

7 Davis, R "Interactive Transfer of Expertise Acquisition of New Inference Rules", Artificial Intelligence vol 12, pp 121-157, 1979

8 Davis, R and Lenat, D B "Knowledge-Based Systems in Artificial Intelligence", McGraw-Hill Book Company, 1982

9 Shortliffe, E H , "Computer-Based Medical Consultations MYCIN", American Elsevier Publishing Co , New York, 1976

10 Aikens, J S , "Prototypes and Production Rules A Knowledge Representation for Computer Consultations ", Stanford Heuristic Programming Project, Report No STAN-CS-80-814, (1980)

11 Tenczar, P , and Golden, W , "Spelling, Word, And Concept Recognition", CERL Report X-35, Computer-based Education Research Laboratory, University of Illinois, Urbana, Illinois, October 1972

12 Roberts, C , "Partial-Match Retrieval via the Method of Superimposed Codes", *Proceedings of the IEEE*, Vol 67, No 12, December 1979