To appear in
IEEE Transactions on Fuzzy System
1998

FuzzyShell: A Large-Scale Expert System Shell using Fuzzy Logic for Uncertainty Reasoning*†

Juiyao Pan, Guilherme N. DeSouza, and Avinash C. Kak
Robot Vision Laboratory
1285 E.E. Building
Purdue University
West Lafayette, IN 47907-1285

Abstract

There exist in the literature today many contributions dealing with the incorporation of fuzzy logic in expert systems. But, unfortunately, much of what has been proposed can only be applied to small-scale expert systems, that is when the number of rules is in the dozens as opposed to in the hundreds. The more traditional (non-fuzzy) expert systems are able to cope with large numbers of rules by using Rete networks for maintaining matches of all the rules and all the facts. (A Rete network obviates the need to match the rules with the facts on every cycle of the inference engine.) In this paper, we present a more general Rete network that is particularly suitable for reasoning with fuzzy logic. The generalized Rete network consists of a cascade of three networks: the Pattern Network, the Join Network, and the Evidence Aggregation Network. The first two layers are modified versions of similar layers for the traditional Rete networks, and the last, the aggregation layer, is a new concept that allows fuzzy evidence to be aggregated when fuzzy inferences are made about the same fuzzy variable by different rules.

1 Introduction

During the last decade, fuzzy expert systems have been proposed as a viable and user-friendly approach for reasoning and control in the presence of uncertainty for a variety of problem domains. It can no longer be disputed that fuzzy logic allows humans to interface with machines in a language that, at the human end, consists of expressions not unlike those we use in our

*For testing and evaluation, the code for FuzzyShell is available on WWW at http://RVL1.ecn.purdue.edu/fuzzy-shell/fuzzy-shell.html

†This work was supported by the Office of Naval Research under Grant ONR N00014-93-1-0142.

PURDUE UNIVERSITY

January 14, 1998

Dr. Clifford G. Lau
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5000

Dear Dr. Lau,

Enclosed is our final report for the ONR Grant N00014-93-1-0142.

Sincerely,

Avi Kak

A. C. Kak
Professor, School of Electrical and Computer Engineering

Enclosure

cc: Director, NRL
DTIC (Two copies)
AGO, ONR
DSP, Purdue

ACK:md

Robot Vision Lab
1285 EE Building
West Lafayette, IN 47907-1285

Phone: 765-494-3551
Fax: 765-494-6440
E-mail: kak@ecn.purdue.edu
WWW: http://RVL4.ecn.purdue.edu/~kak/

everyday discourse and that, at the machine end, possesses numerical attributes necessary for quantification. That fuzzy logic should be able to fulfill simultaneously the needs of both the humans and the machines is made possible through the association of fuzzy membership functions with symbolic or conceptual variables, the reasoning itself consisting of inferring the membership functions for some of the variables as evidence becomes available for some others.

While the use of fuzzy logic for control has mushroomed over the years – there are now thousands of papers on the subject – its use in expert systems, especially when long chains of inference are involved, has lagged. Some researchers would even go so far as to claim that fuzzy logic has not worked out at all for expert systems [11, 12]. The claim is made despite the fact that there now exist organizations offering fuzzy expert system shells. We believe that one of the better known such shells is the FuzzyCLIPS system, developed recently by the National Research Council, Canada[17]. In addition to the shells, there also now exist in the literature numerous reports on fuzzy expert systems for specific applications, such as CADIAG-2[1, 2], FRIL[3], RUM[5], FLOPS[7], FESS II[8], REVEAL[19], SYSTEMZ-II[21], FLISP[25], FESP[26], Fault[27].

Because of challenges unique to fuzzy expert system design, especially when multiple levels of inference [1] are involved, there is always the question of whether the developers of a shell or an expert system took any liberty with the logic itself in order to get the entire software system to work. The special difficulties that fuzzy reasoning presents – difficulties that do not exist for the case of crisp reasoning – are two-fold:

1) All the rules that contain the same linguistic variable on the consequent side must be fired and evidence aggregated for that linguistic variable in accordance with the method of inference chosen before a fact asserted by one such rule is allowed to trigger another rule. Said another way, an asserted fact about a linguistic variable cannot be allowed to trigger another rule if there are other rules yet to be fired that would assert further facts about the same linguistic variable.

2) Since fuzzy terms often have overlapping membership functions, a fact such as (robot speed fast) should be able to match with a rule antecedent such as (robot speed medium) if there exists an overlap between the membership functions for the fuzzy terms *fast* and *medium*.

While it is clear that all the fuzzy expert system shells, FuzzyCLIPS being a case in point, are able to handle the second difficulty, the same cannot be said for the first difficulty. In fact, we are not aware of any fuzzy expert system shell that handles cleanly the first difficulty. [2]

We must hasten to add that if the goal was to design a shell for a small-scale expert system (one containing a dozen or so rules), it would be trivial to write a computer program that would handle both the above difficulties cleanly. The inference engine of such a program would consist of a simple do-loop that, for each cycle, would match all the facts with all the rules; for each cycle, a list of all the rules that can be triggered by the facts currently available would be constructed;

[1] "Multiple levels of inference" is synonymous with the notion of "chaining of rules."

[2] Besides the two difficulties mentioned above, there also exist many other challenges that must be surmounted for building a fully functional fuzzy expert system, challenges such as knowledge acquisition, choice/ representation of membership function, etc. However, practicalities of research make it impossible to simultaneously attack all these problems.

the rules would then be batched so that all the rules with the same linguistic variable on the consequent side would be in the same batch; and, finally, all the rules in the same batch would be fired one after another before any other activity of the inference engine, save for the incremental aggregation of the membership function for the consequent linguistic variable as each rule is fired. **Unfortunately, such do-loops cannot be used for large expert systems.**

As years of experience with traditional expert systems have demonstrated, a simple-minded approach that entails that all the facts be matched with all the rule antecedents in each cycle of the inference engine results in computational complexity that is exponential in the number of rules. It is for this reason that the heart of all the commercial expert systems of the traditional non-fuzzy type, such as ART, CLIPS[9], OPS83[16], etc., that are designed to handle hundreds or thousands of rules consists of a Rete network or a variant thereof. Rete networks were originally advanced by Forgy [14, 15] to enhance the computational efficiencies of the traditional expert systems, that is expert systems that are based on Classic Logic.

Therefore, if fuzzy expert systems are to become as successful as their more traditional counterparts, it is imperative that techniques such as Rete network be generalized from classic logic to fuzzy logic and that is exactly we have done in this paper. The main contribution of this paper is to show how a Rete network can be designed so as to carry out fuzzy inference with computational efficiencies that are comparable to those of the now well-known expert system shells such as OPS5, OPS83, CLIPS, etc. without running afoul of the principles of fuzzy logic.

As the rest of this paper makes clear, generalization of Rete networks to handle fuzzy inference has proved to be non-trivial, even when we restrict ourselves to the case of Mamdani-type inference [22]. As described elsewhere in this paper, our work has required complex data and control structures. While the Rete network for a traditional expert system uses basically two networks, one for matching facts with rule antecedents and the other for reasoning about the consistency of instantiations for the variables in the different antecedent statements of the rules, the generalized Rete network for fuzzy reasoning has required three networks. The additional network is needed for the incremental aggregation of fuzzy evidence as each rule is fired. In addition, as the reader will see, the other two networks for the case of fuzzy reasoning are quite different from such networks for the crisp case.

While it is true that the current implementation of the generalized Rete network, as reported in this paper, is for the specific case of Mamdani-type fuzzy inference, it can easily be modified to accommodate most other types of fuzzy inference.

In what follows, we will use a specific example to illustrate a fundamental problem with the currently available shells for fuzzy expert systems. We will then review the relevant concepts from fuzzy logic, focusing especially on Mamdani-type inference, and provide a brief review of production systems and Rete networks. That will be followed with a detailed discussion of how Rete networks can be designed to enable them to carry out Mamdani-type inference for our new reasoning architecture, named FuzzyShell. Finally, we will use an example to illustrate the workings of FuzzyShell.

2 A Specific Example Illustrating a Serious Shortcoming of the Current Fuzzy Expert System Shells

A fundamental problem with practically all the currently available shells for fuzzy reasoning is that they are surface-level modifications of existing rule-based programming shells. To us, trying to modify an expert system shell intended for crisp reasoning so that it can carry out fuzzy reasoning is a perfect example of fitting a square peg into a round hole. For reasoning that involves chaining of rules, these systems invariably violate the principles of fuzzy logic, as will be clear from subsequent discussion. In other words, they can be used with confidence only for control applications where in most cases only a single level of inference is needed.

To drive home this point, consider, for illustration, the following three rules:

rule1: IF (X is A) THEN (Y is B)
rule2: IF (X is A') THEN (Y is B')
rule3: IF (Y is B) THEN (Z is C)

Assume that the terms A and A' associated with the linguistic variable X have overlapping membership functions, and that the same is the case for the terms B and B' associated with the linguistic variable Y. Assuming that the working memory contains the fact (X is A), let's now examine how a traditional expert system modified to carry out fuzzy reasoning would make inferences.

The fact (X is A) would enable rule1 and, after this rule is fired, the fact (Y is B) would be asserted into the working memory. Under the commonly used LEX and MEA [3] conflict resolution strategies – strategies in which the time recency of the enabling fact plays a decisive role in deciding which rule to fire next – this new fact would now enable rule3. The problem is that if the dictates of fuzzy logic are not to be violated, then rule2 must also be fired before rule3 is picked. (Recall, the fact (X is A) will also enable rule2 due to the overlap between membership functions for the terms A and A'.) In other words, all the enabled rules that have anything to say about the linguistic variable Y must all be fired and their conclusions aggregated prior to selecting any rules for firing whose antecedents containing this linguistic variable.

It is true that with a different strategy in a traditional production system, such as the breadth-first strategy, it would be possible to ensure that both rule1 and rule2 are fired before the selection of rule3. But it is often not possible to use the breadth-first strategy. For example, backward reasoning can only be carried out with the MEA strategy[6].

Therefore the upshot is that today one must either resort to highly inefficient systems that carry out a brute-force implementations of fuzzy-logic in which every fact must be matched with every rule for every cycle of the inference engine, or one might try to use modifications of the currently-available computationally-efficient expert system shells, but at the risk of implementing

[3]These acronyms stand, respectively, for the lexicographic and the means-end-analysis strategies. These are two of the more common strategies for conflict resolution in a traditional production system. In OPS like expert system shells – that includes the CLIPS shell – the MEA strategy is particularly useful for backward reasoning[6].

faulty logic. While the former types of systems can only be used when the number of rules is small, usually in the tens as opposed to hundreds, the use of the latter is fraught with the possibility of error in the inferences drawn.

3 Fuzzy Logic

Despite the existence of voluminous literature on the subject [4, 13, 20, 22, 23, 28, 29, 30, 32], we feel it is necessary for us to include a very brief review of fuzzy logic for two reasons: We wish to inform the reader as to which fuzzy logic – especially what compositional operators for propagating evidence – we have used for the new class of Rete networks presented in section 5. Additionally, we believe that a brief fuzzy logic review here would be helpful to the readers who are from the traditional expert system community.[4]

Central to fuzzy logic is the notion of a linguistic variable, such as *distance*, whose values are known as the terms, such as *very-far, far, very-near, near, close, very-close.* The range of numerical values spanned by a linguistic variable is called the *universe of discourse* for that variable. Associated with each term is a *fuzzy membership function*, a function whose range is the closed interval [0,1] and whose domain is a subset of the universe of discourse. A term together with its membership function is also called a *fuzzy set.* The three basic operations on fuzzy sets are those of union, intersection, and complement. There are various ways of generalizing the definitions of these operations from crisp sets to fuzzy sets. We will now briefly state the definitions on which our work is founded. Suppose for a linguistic variable U we have two fuzzy sets A and B, with membership functions given by μ_A and μ_B, respectively, the membership function of the union is given by

$$\mu_{A \cup B}(u) = \max[\mu_A(u), \mu_B(u)]$$

The *max* operator above is just one of the many S-norm functions [10] commonly used in fuzzy logic for implementing the union and it represents the "most pessimistic" of them. Similarly, for the definition of intersection, let A and B be the terms of the same linguistic variable U, the membership function of the intersection is given by

$$\mu_{A \cap B}(u) = \min[\mu_A(u), \mu_B(u)]$$

The *min* operator is one of the T-norm functions that define intersection and represents the "most optimistic" of them. Both the *max* operator for union and the *min* operator for intersection are arbitrary but commonly used choices among all possible ways of defining such operators.

[4]The explanation of our work is made complicated by the fact that it represents a bridge between two rather disparate fields of knowledge, fuzzy logic and Rete networks. To make our work accessible to readers from the traditional expert systems community, we must provide at least a brief review of the fuzzy logic ideas, although at the risk of sounding pedestrian to the fuzzy logic community. And, similarly, to make our work accessible to the fuzzy logic community, we must not gloss over too quickly the concepts from Rete networks. Hence the material reviewing Rete networks.

Finally, the complement can be defined by

$$C(\mu_A(u)) = 1 - \mu_A(u)$$

Besides these basic operations, fuzzy inference requires that we associate a fuzzy membership function with each implication. Along the lines discussed in[22] , for the rule "IF A THEN B" , a possible fuzzy membership function is

$$\mu_{A \to B}(u, v) = \min(\mu_A(u), \mu_B(v))$$

where u and v span the universe of discourse corresponding to the terms A and B. Note that $\mu_{A \to B}(u, v)$ is a two-dimensional set. To drive home this very important point, consider the following rule

IF distance is close THEN brake hard

In Fig.1, we have shown the fuzzy membership function, $\mu_{distance:close \to brake:hard}$, in the form of a matrix of numbers that are boxed when μ_{close} is as shown just below the horizontal axis (on the line labeled μ_A) and μ_{hard} as shown to the left of the vertical axis. Shown in Fig.2 are the membership functions for the different terms for the linguistic variables *distance* and *brake*.

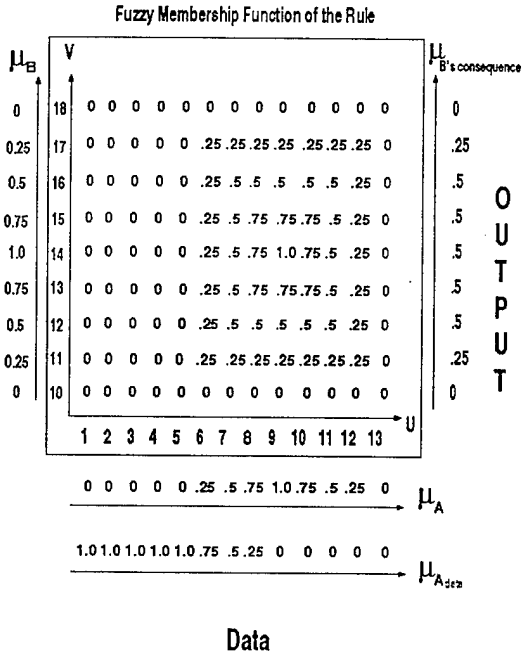

Figure 1: A pictorial illustration of the compositional rule of inference.

As suggested by Zadeh, given data A_{data}, fuzzy evidence can be propagated through a rule by using what is usually referred to as a *Compositional Rule of Inference* (CRI)[31]. To explain the

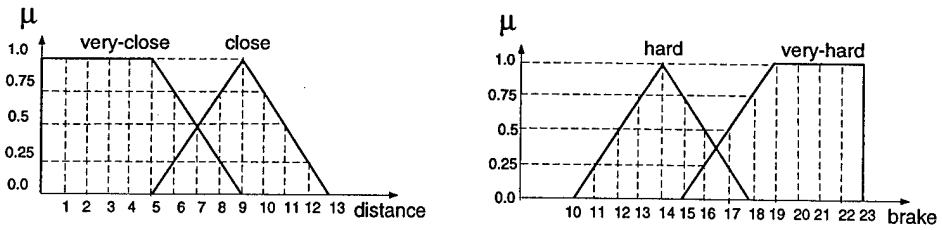

Figure 2: Fuzzy membership functions for *close*, *very-close*, *hard* and *very-hard*.

CRI we use, let's say the given rule is "IF A THEN B" and the data available is A_{data}. Our goal now is to infer a new membership function for the term B in the consequent of the rule in the light of the available fact A_{data}. The following CRI is common in the literature

$$\mu_B(v) = \sup_u \{\min_u [\mu_{A_{data}}(u), \mu_{A \to B}(u, v)]\}$$

where u spans the universe of discourse for the term A and v the same for the term B. Obviously, the terms A and A_{data} must span the same universe of discourse. To illustrate this further with our example, we have shown in Fig.1 the sup-min operation demanded by the formula above leading to the inferred membership function for the consequent variable *hard*. The inferred membership function is shown as a column of numbers to the right of the box. In this case, A_{data} is *very-close* (see Fig. 2). This inferring process is denoted by

$$B = A_{data} \circ (A \to B)$$

In practice, there is a faster way to implement the above CRI. As shown in Fig.3, we get the same result if we first take

$$\sup \min_u [\mu_{A_{data}}, \mu_A]$$

and use the number thus obtained to truncate the membership function for the consequent term B. We will refer to the number obtained from the sup-min operation above as the degree-of-match between A_{data} and A. Using the example membership functions shown in Fig.2, the reader can easily verify the veracity of this statement. So while it is common to show figures like the one in Fig.3 to explain the propagation of fuzzy evidence, for theoretical analysis one has to resort to the compositional formulas shown above.

We will now discuss the case when fuzzy evidence must be propagated through multiple rules simultaneously and the results aggregated. Consider the case when we have the following two rules that can both be triggered by the same data.

Rule1: IF distance is close THEN brake hard
Rule2: IF distance is very-close THEN brake very-hard

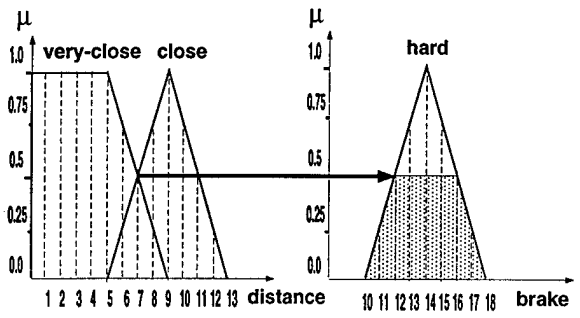

Figure 3: The degree of match between *close* and *very-close* is used to truncate the membership function of the consequent term *hard* associated with the linguistic variable *brake*.

For example, if the measured distance, considered as a discrete value, has a value which has a non-zero membership in both the sets *close* and *very-close*, then both the rules shown above can be fired. If the measured distance is itself a fuzzy set – on account of the uncertainties associated with the sensor – and this fuzzy set has a non-zero intersection with the fuzzy sets *close* and *very-close*, then again both these rules could be fired. Two important issues need to be resolved with regard to the propagation of evidence through multiple rules: a) Is the order in which the rules are invoked important? and b) How to combine the conclusions drawn from the all the rules fired from a given set of facts? The following lemmas answers both questions. For proofs, the reader is referred to [20].

Lemma 1: Given a fact A that can match the antecedents of rules $R_1, R_2, ..., R_n$, each rule being of the form $A_i \rightarrow B_i$, the overall conclusion from the rules satisfies the following distributive relationship:

$$B = A \circ \bigcup_{i=1}^{n} R_i = \bigcup_{i=1}^{n} A \circ R_i$$

Note that a set of rules that can all be fired by a given fact have been represented by a union operator; also note that B is the fuzzy set obtained by aggregating together the appropriately modified conclusion fuzzy sets $B_1, B_2, ..., B_n$. Representing a set of rules by a union operator is justified because the different rules represent alternative ways of asserting the conclusion. The lemma above tells us that if we wish to propagate a fuzzy fact through, say, two different rules, we should propagate the fact through each rule separately and then take fuzzy union of the conclusion fuzzy sets.

While the above lemma teaches us how to deal with multiple rules, there is only one term in each rule antecedent. We will now allow each rule to contain multiple terms, expressed as conjunctions, in its antecedent. In what follows, we will show two more lemmas, derived by Lee [20], which will tell us how to carry out the fuzzy inference in one rule, and then how to obtain the final conclusion of a linguistic variable for a set of firing rules.

Lemma 2: Given two facts A and B, corresponding to two different linguistic variables, that can match the two facts in the antecedent of a single rule of the form "IF A_i and B_i THEN C_i",

the conclusion fuzzy set for C_i can be obtained by the following formula:

$$C_i = [A \circ (A_i \rightarrow C_i)] \cap [B \circ (B_i \rightarrow C_i)]$$

Lemma 3: Given two facts A and B, corresponding to different linguistic variables, that can match the two facts in each of rules $R_1, R_2, ..., R_n$, each rule being of the form "IF A_i and B_i THEN C_i", the overall conclusion that can be drawn from the rules obeys the following distributive relationship:

$$C = \bigcup_{i=1}^{n} [A \circ (A_i \rightarrow C_i)] \cap [B \circ (B_i \rightarrow C_i)]$$

This lemma tells us that the final conclusion is to be made by propagating the fuzzy evidence through each rule separately and then taking a fuzzy union of the resulting conclusions. The readers should find it easy to extend the above lemmas for the case with multiple inputs and multiple outputs.

An Important Issue Related to Fuzzy Evidence Aggregation

The different terms for a linguistic variable will ordinarily be defined by a human user. Unfortunately, if two different rules assert two different terms for the same linguistic variable, then the composite fuzzy membership function for that linguistic variable may not correspond to any of the pre-defined terms. Consider the following fact and rules.

Fact-1: (distance is near)

Rule-1: IF (distance is close) THEN (speed is slow)
Rule-2: IF (distance is far) THEN (speed is medium)
Rule-3: IF (speed is fast) THEN (. . .)

The membership functions of fuzzy terms of *speed* and *distance* are shown in Fig.4. Rule-1 and Rule-2 will be activated by Fact-1 since the membership function for *near* has an overlap with those of *close* and *far*. It is clear that after Rule-1 and Rule-2 have fired, the composite membership function for the consequent linguistic variable will look like what is shown in Fig.5. If a purely numerical approach to fuzzy evidence accumulation was used – as would be the case for small-scale systems – such a composite membership function would pose no problems even though it does not correspond to any of the terms shown for the consequent linguistic variable. However, for a large-scale system, there needs to be a mechanism for associating a label with any arbitrary membership function that the process of evidence accumulation might result in.

Therefore, in FuzzyShell, for each linguistic variable we define a special term, called the *aggregate_term*. This term is a shell-created special term for each linguistic variable for which fuzzy evidence is asserted via the consequents of two or more rules. Evidently, this term does not possess the usual semantics; for example, for the linguistic variable *speed*, the usual terms, *slow,*

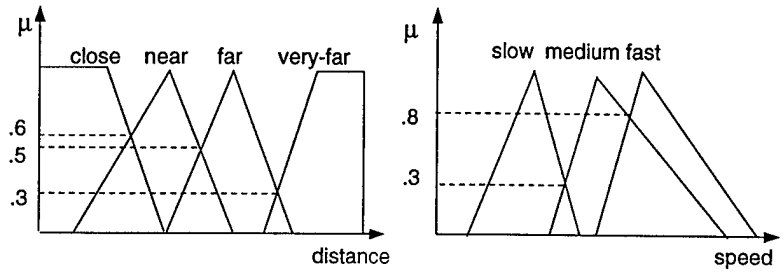

Figure 4: Membership Functions for the terms *distance* and *speed.*

medium and *fast*, possessing obvious meanings, stand in contrast to the term *aggregate_term* that admits any membership function.

For the example shown above, it is obvious that after the evidence is accumulated from Rule-1 and Rule-2, Rule-3 would fire due to the overlap between the accumulated in the form of the asserted membership function for the term *aggregate_term* and the membership function for the term *fast.*

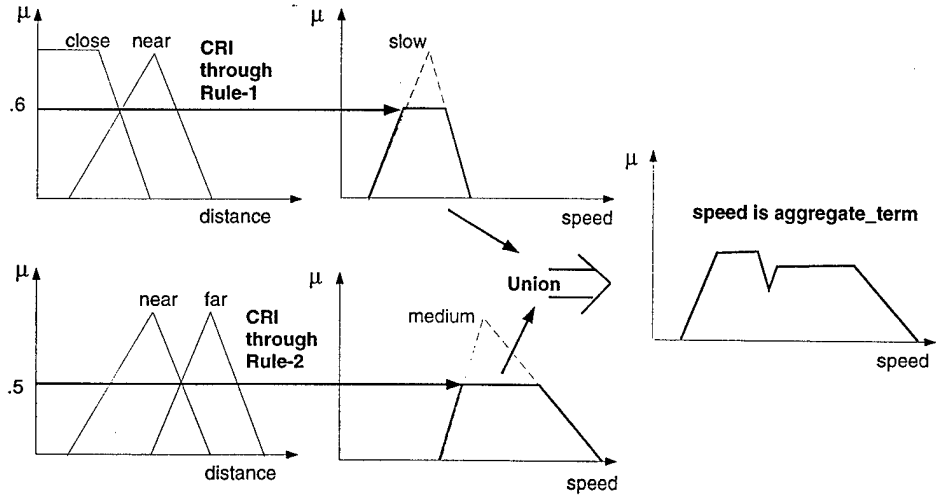

Figure 5: The composite membership function obtained for the consequent linguistic variable after the fuzzy evidence is combined from Rule-1 and Rule-2.

4 Production Systems and Rete networks

All large-scale expert-system shells of note, such as the CLIPS system from NASA, the commercial systems ART, OPS5, OPS83 and many others, are founded on the concept of production systems, a concept that was first promulgated by Newell and his co-workers [24] and subsequently made useful for practical problem solving by the seminal work of Forgy on Rete networks [14, 15].

The overall organization of a production can be explained very quickly with the help of Fig.6.

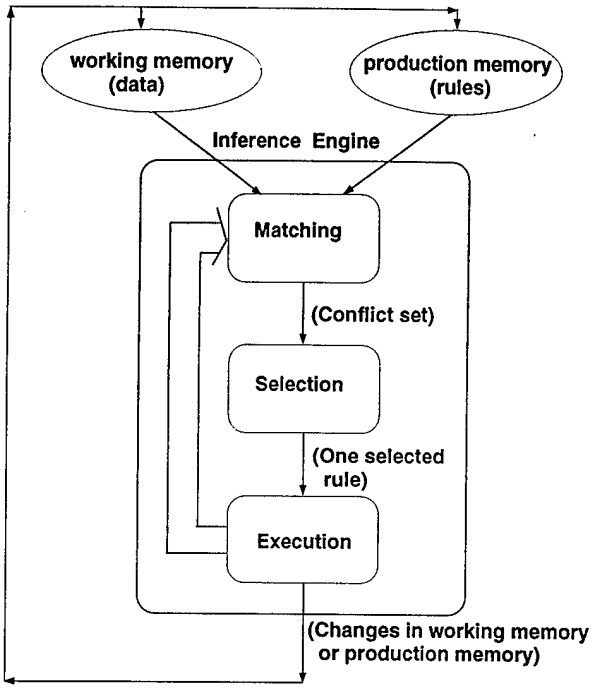

Figure 6: Architecture of a production system.

The three principal components of a production system are a *working memory* for data, also called the facts; a *production memory* for rules; and an *inference engine*, whose function is to infer new facts from existing facts and the rules and then to assert the new facts into the working memory, and then to continue this process of discovering new facts via the rules through the new store of facts in the working memory, until of course no further facts can be inferred. Traditionally, the statements on the left hand side of a rule are called the *condition elements* in the argot of production systems and the statements on the right hand side the *action elements*. What gives an expert system the flavor of a production system is the manner in which the inference engine operates. Each cycle of the inference engine consists of three steps, each corresponding to one of the modules shown inside the inference engine box in Fig.6. The first step consists of propagating the latest changes to the working memory through the Rete networks – its operation will be explained shortly – and figuring out which rules are enabled by the latest facts. This set of rules is called the *conflict set* or the *agenda*. The second step consists of ordering the rules posted on the agenda on the basis of considerations such as whether or not the same rule was fired before, the recency of the enabling facts from the working memory, the simplicity/complexity of the rule, etc. The top-ranked rule is then selected from the agenda thus ordered. The last step of the inference engine then executes the *actions* listed on the right-hand-side of the fired rule. This action may consist of either asserting the newly inferred facts into the working memory, or invoking functions or procedures that are external to the production system. [5] This then is the explanation in a nutshell of a production system. We are now ready to explain the operation of

[5]For further discussion on the different possible strategies for ordering the rules posted on the agenda – these strategies carry names like LEX, MEA, Depth-first, etc., – the reader is referred to [6, 18].

Rete networks.

Rete networks have been used in the traditional expert systems to ameliorate the computational burden that would otherwise be associated with matching all the rules with all the facts on each cycle of the inference engine. A regular Rete network compiles all the rules into a tree-structured sorting network of feature tests. Only the rule-antecedents are used for the creation of a Rete network. Generally speaking, a Rete network contains two networks that are the Pattern Network and the Join Network, the former for the purpose of representing all the condition elements in all the rules (in a manner that eliminates duplication between the rules), and the latter for comparing the bindings of those variables that are common to the different condition elements for the same rule. To illustrate the organization of a Rete network, consider the following example where we have shown the condition elements of two rules:

(rule-name rule-1
    (robot (m̂ission navigation)(ŝensor ?x))
    (sensors(p̂riority low)(ŝensor ?x))
⇒
    (. . .))

(rule-name rule-2
    (robot (m̂ission navigation) (ŝensor ?x))
    (sensors (p̂riority high) (ŝensor ?x))
⇒
    (. . .))

In the Rete network, shown in Fig.7, each condition element of each rule is represented by a sequence of tests, each test corresponding to one node, called the pattern node, in the Pattern Network. For example, the first condition element of rule-1 is represented by the pattern nodes a, b, and c, whereas the second condition element of the same rule is represented by d, e, and f. The join node, marked J1, for rule-1 holds the test on the variable that is shared by the two condition elements of the rule. In this case, the test ensures that the bindings for the variable ?x are consistent.

When a new fact is asserting into the working memory, the fact becomes a token that traverses through the Pattern Network, and subsequently may be propagated to the Join Network to find out which rules can be enabled. The syntax of a token is represented as follows.

(tag Fact-ID)

where the tag could be "+" or "−" for representing the addition or deletion for the tokens, respectively. The Fact-ID contains the working memory elements or sequences of working memory elements. To explain how the tokens are propagated, assume that the following fact is asserted into the working memory.

Fact-1: (robot (m̂ission navigation)(ŝensor range-sensor))

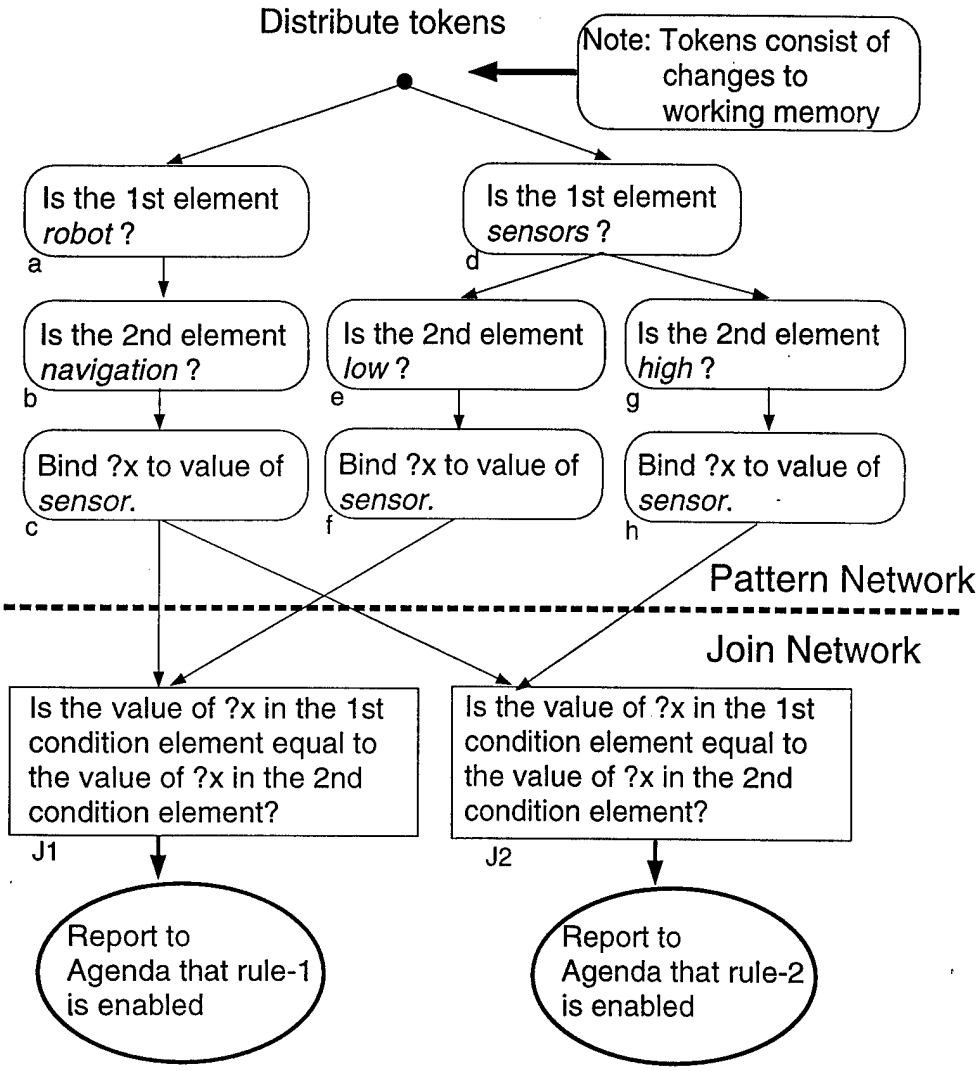

Figure 7: Rete network for rule-1 and rule-2.

A token (+ Fact-1) will be created, and then propagated into the pattern network along arcs that contain tests relevant to the fact. [6] In this case, the first element of the token will pass the test in pattern node *a* since this node tests for whether or not the first field contains the symbol *robot*. For similar reasons, the token would also pass through the node *b*. As the token passes through the pattern node *c*, the variable ?x will be bounded to be range-sensor. Finally, the token will be saved in what is referred to as the left memories of the join nodes marked J1 and J2. This would remain the status of the Rete network if no further information were to be injected into the production system. Now assume that the following fact is asserted into the working memory.

Fact-2: (sensors (p̂riority high)(ŝensor range-sensor))

This will cause the token (+ Fact-2) to be injected into the Rete network at the root node. In a manner similar to what happened for the previous token, the new token will travel along the path composed of the pattern nodes d, g, and h, and will then come to rest in the right memory for the join node J2. Given the tokens residing in its left and the right memories, the join node J2 is now able to carry out the test assigned to it and declare that the rule-2 is to be placed on the agenda.

The important features to note about a Rete network are that it eliminates temporal and structural redundancies for figuring out what rules would be enabled at each cycle of the inference engine. Since all the changes to the working memory are propagated into the network in the form of positive and negative tokens, a fact once absorbed into the network remains there until a negative token regarding the same fact is propagated into the network. A negative token is propagated when a fact is retracted. So, unless a fact is retracted, computations for matching the fact with the condition elements need to be carried out only once. This is referred to as elimination of temporal redundancies in the matching process. Another important feature of a Rete network is that it eliminates the structural redundancies, that is redundancies caused by structural similarities between the different statements of rule-antecedents. For example, the pattern node *d* is shared by rule-1 and rule-2. By eliminating the redundancies mentioned above, a Rete network can reduce the computational complexity for matching the facts with the rules from exponential time complexity in the number of rules to polynomial time complexity [14].

5 Architecture of FuzzyShell

We will now present our new architecture for a fuzzy expert system shell, FuzzyShell, that employs a new class of Rete networks which make possible fuzzy inference for large-scale expert systems. Since this system is based on the same principles as a production system, its computational efficiencies parallel those of such well-known systems as CLIPS, OPS83, etc.

[6] To speak more precisely, the search for which arcs to use for propagating a token takes place in a depth-first fashion.

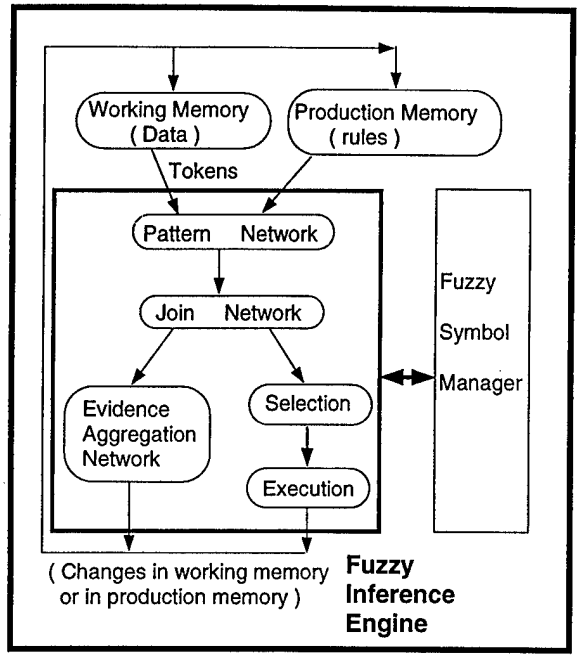

Figure 8: Architecture of FuzzyShell

Fig.8 depicts the overall flow of control and the various modules of the FuzzyShell. All the symbols, fuzzy and crisp, are interned in a module called the the Fuzzy Symbol Manager where the latest membership functions for the fuzzy terms are stored. Therefore, when the firing of a rule calls for a change in the membership function of a variable, that change takes place in the Fuzzy Symbol Manager.

Besides the Fuzzy Symbol Manager, what makes FuzzyShell truly different from the more traditional production systems is the new three-layer Rete network shown in Fig.8. The first two layers, the Pattern Network and the Join Network, are similar but not at all identical to such layers in, say, CLIPS. *But the third layer, the Evidence Aggregation Layer, is new and designed solely to account for the fact that, in fuzzy inference, the membership function of a given fuzzy term can be modified by multiple rules and that all these changes must be aggregated together before that variable participates in subsequent cycles of the inference engine.*

In what follows, we will show how the working memory elements are represented in FuzzyShell. We will then discuss the representations used for rule antecedents in the Pattern and the Join Networks; we will show representations that allow the antecedents to be of arbitrary length. We will also discuss how the Pattern Network calculates the degree of match between a data membership function and one associated with a matching term in a rule antecedent. Next, we will discuss how the Join Network enforces consistency across different antecedent elements in a rule. The last subsection will then present the workings of the Evidence Aggregation Network.

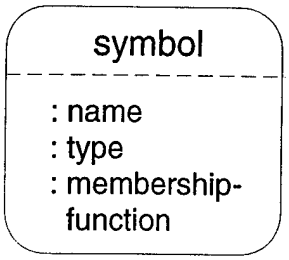

Figure 9: Shown here is the representation for a symbol in the working memory.

5.1 Representation of Working Memory Elements

Each Working Memory Element is composed of symbols, fuzzy and crisp. Each symbol, fuzzy or crisp, is represented by instantiating the data structure shown in Fig.9. This representation consists of a field for *name*; a field for *type*, indicating if the symbol is crisp or fuzzy; and a field for *membership-function* that points to a data structure where the membership function of the symbol resides. The membership function is allowed to be piecewise linear continuous, as shown in Fig.10, where the membership function is represented by a linked list of nodes such that each node contains a value from the universe of discourse and a membership grade corresponding to that value. To compute the degree-of-match between two membership functions, the system computes the intersection of the membership functions comparing each linear edge in one function against all the linear edges in the other. For example, to compute the intersection of the two membership functions shown in Fig.11, the edge a of one of the functions will be compared with all the edges of the other; and then edge b will be compared in a similar manner; and so on. If N is the number of linear segments in a membership function, the complexity of this approach is obviously $O(N^2)$, but given that N is usually a very small number, the resulting computational burden is minimal. After the intersection points between the two membership functions are computed, the point that has the largest value for the intersection is retained; this largest value, as for example represented by the μ_d value of the point p in Fig.11, corresponds to what would result from the sup-min operation.

As we mentioned in the previous section, a special term *aggregate_term* is used to represent the accumulated evidence for a linguistic variable during the inference process. For FuzzyShell to work in complex domains, a linguistic variable may have different *aggregate_term* in different but related contexts. To drive home this point, assume that we wish to assert the following facts into the working memory:

Fact-a: (robot (ñame Peter)(ŝpeed fast))
Fact-b: (robot (ñame Alvin)(ŝpeed medium))
Fact-c: (robot (ñame Peter)(ŝpeed medium))
Fact-d: (robot (ñame Alvin)(ŝpeed slow))

Obviously, the different terms for the linguistic variable *speed* in Fact-a and Fact-b should not be aggregated into a single membership function since the facts relate to different robots. However,

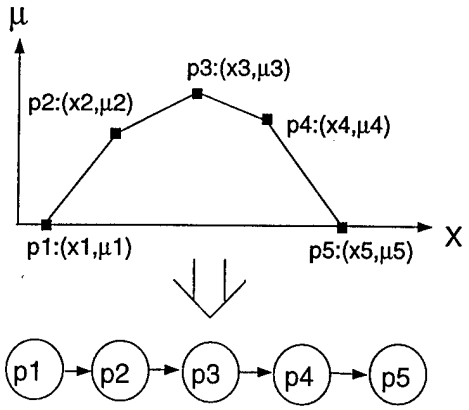

Figure 10: The linked list shown in the lower portion allows the membership function shown in the upper portion to be piecewise linear continuous.

the fuzzy evidence in Fact-a and Fact-c should be aggregated into a single membership function function for *speed*. The same would be the case for Fact-b and Fact-d. Assume that the following two facts represent the accumulated evidences, Fact-1 for Fact-a and Fact-c, Fact-2 for Fact-b and Fact-d:

Fact-1: (robot (ñame Peter)(ŝpeed aggregate_term))
Fact-2: (robot (ñame Alvin)(ŝpeed aggregate_term))

Evidently, the two instantiations of the data-structure of Fig.9 for the same fuzzy term *aggregate_term* will be different. Each will have a pointer from the data structure representing the entire working memory element.

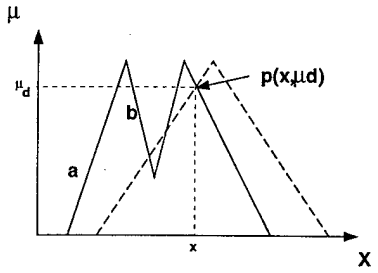

Figure 11: Shown here are two membership functions for two separate fuzzy sets to illustrate how to compute the degree of match between them.

5.2 Pattern Network

As mentioned in Section 4, in a traditional (non-fuzzy) expert system the Pattern Network is used for creating and then maintaining matches between all the rule antecedents and the currently available facts. As was mentioned there, each node of this network carries out a test that, in

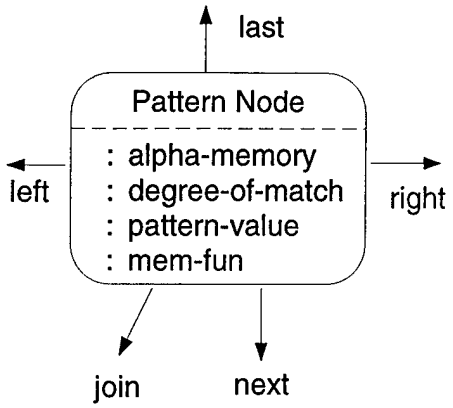

Figure 12: Representation of a pattern node.

effect, compares a symbol from a rule antecedent with a symbol in the same position in a fact. In other words, each "arm" of the Pattern Network is a string matcher for comparing condition element strings with fact strings.

For obvious reasons, when fuzzy predicates are involved there is much more to matching than carrying out a position-by-position comparison of symbols. Now we must also compare fuzzy sets associated with the symbols and ascertain the degree of match between a fact symbol and a condition element symbol. This introduces complications into the design of a Pattern Network. In order to explain how the Pattern Network works in FuzzyShell, we will first introduce the pattern node (Fig.12), the basic building block of the Pattern Network. This node is connected to the rest of the Pattern Network via the five links shown, although not all of them may be used for any particular node, depending on the position of the node in the network. For example, the link, *join*, is used at only those nodes that feed information into the Join Network. Also shown in Fig.12 is all the information that resides at each pattern node. The first one of these, *alpha-memory*, is used only in the terminal pattern nodes; these are the same nodes that use the *join* link. This information consists of the token that has propagated to that point. The second piece of information at each node, *degree-of-match*, represents either the sup-min of the fuzzy membership function for the variable that is under test and the membership function of the corresponding entity in the fact, or the least of such sup-min values for upstream variables. The next field, *pattern-value*, is the fuzzy term in the pattern. Finally, there is a field, *mem-fun*, for representing the membership function of the node if the node stands for a fuzzy variable. The following explanation will shed further light on how these different kinds of information are used.

To explain further, consider first the following three condition elements in a rule-antecedent.

Pattern1 : (X (p̂rop1 a) (p̂rop2 b))
Pattern2 : (X (p̂rop1 a) (p̂rop3 c))
Pattern3 : (Y (p̂rop4 d) (p̂rop5 e))

where the fist pattern could, for example, stand for a real pattern like

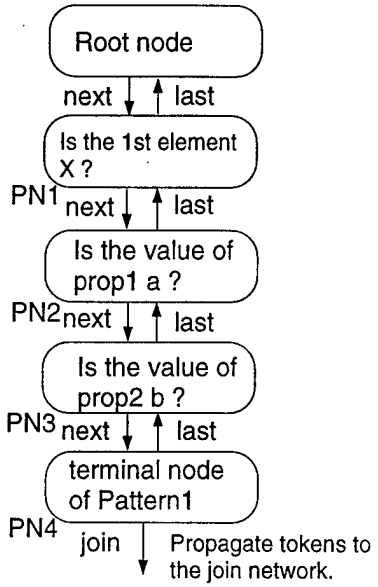

Figure 13: This example illustrates the Pattern Network for Pattern1.

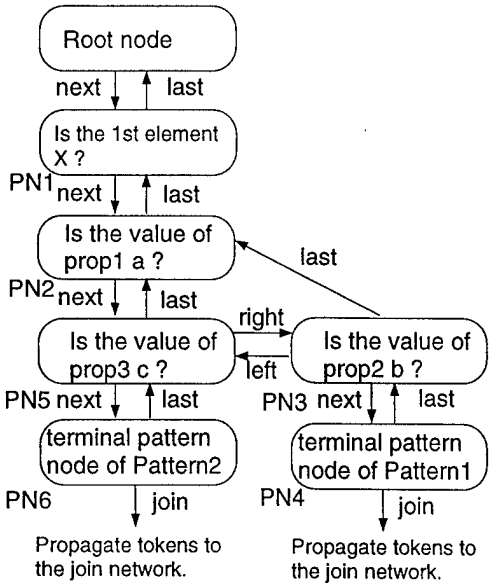

Figure 14: This example illustrates the Pattern Network for Pattern1 and Pattern2.

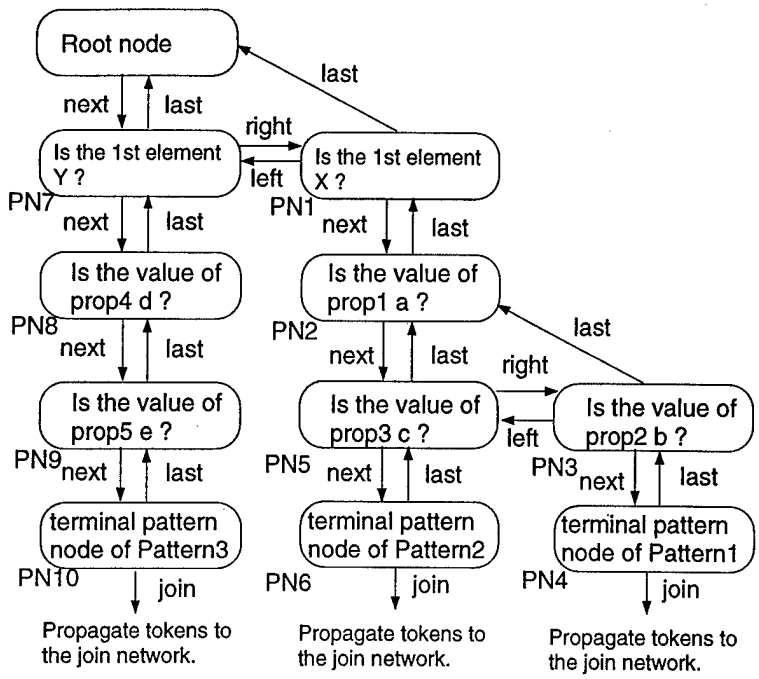

Figure 15: This example illustrates the Pattern Network for Pattern1, Pattern2 and Pattern3.

(Robot d̂istance medium ŝpeed low)

These patterns are compiled in the order in which they are read in. Upon reading the first pattern, a network consisting of the nodes PN1, PN2, PN3 and PN4, as shown in Fig.13, is constructed. [The pointers *next* and *last* between the nodes make it possible for a token to traverse easily between nodes. More on this subject later.] When the second pattern is read in, this network is extended in the manner shown in Fig.14 by the creation and inclusion of the nodes labeled PN5 and PN6. Note that this extension was achieved by first breaking the link *next* between PN2 and PN3 in Fig.13; this link now points from PN2 to the new node PN5. At the same time, the links *right* and *left* connect the two siblings nodes PN3 and PN5. Both PN3 and PN5 have PN2 as their common parent since the nodes prior to and including PN2 are common to both the patterns being compiled here. An important point to note is that when a new pattern is compiled, the network already constructed is searched in a depth-first fashion to seek out the nodes that can be reused. The pointer *right* is used to mark the bifurcation point, a point prior to which the nodes are common to two or more patterns. It is in this manner that the Pattern Network eliminates the structural redundancies in the matching process. The reader has probably already noted that of the two sibling links *left* and *right*, it would be sufficient to use only the latter to get rid of all the aforementioned redundancies. The link *left* is needed to eliminate those portions of a compiled network that correspond to a deleted rule. For example, if we wished to delete Pattern1 (supposedly, this pattern is the antecedent of some rule we wish to excise from the database) after the structure shown in Fig.14 has already been created. The deletion of nodes must of necessity begin from the bottom of the Pattern Network, since

the lower down one goes into a Pattern Network, the more specific the nodes are to individual patterns and rules. So, in this case, the system would first delete node PN4. At the same time, through the *last* link of PN4, the system would discover that PN3 belongs to the same pattern. So this node would be deleted next. Simultaneously, the deletion algorithm would discover that the just-deleted node was a bifurcation point; this, through the link *left*, would result in the discovery of the sibling PN5. The last step undertaken by the deletion program would be to drop the *right* pointer from PN5.

Now let's see what happens when Pattern3 is compiled. As shown in Fig.15, since none of the existing pattern nodes is reusable for embedding Pattern3 in the network, the new nodes, PN7 through PN10, are created. Comparing Figures in Fig.14 and Fig.15, note that the link *next* emanating from the Root node is broken with PN1; it is now directed to PN7. Therefore, the new entry point into the network will now be PN7. When a token corresponding to, say, Pattern1 is now propagated into the network, it would clearly fail to descend down the column of nodes headed by PN7. This token will instead use the *right* pointer at PN7 to hop over to the PN1 node.

Before we discuss how to build the Join Network, we will now describe how fuzzy evidence gets propagated through the Pattern Network. As we mentioned earlier, for each pattern node, there is a degree-of-match value stored in the node, the value obtained using the applicable principles of fuzzy inference. To be more specific, given a fact like (X is A) and a rule antecedent like (X is A'), if the membership functions of A and A' have an overlap, the match occurs and the value of *degree-of-match* is computed and stored in the corresponding node of the Pattern Network to indicate the extent to which the fact (X is A) matches the rule-antecedent (X is A'); this degree-of-match then becomes a part of the token that is transmitted downstream in conjunction with the fact. The syntax of this token is

(+ (fact-ID), degree-of-match)

Consider, for example, the following patterns which are the antecedents of different rules. Additionally, assume that the fact in the working memory is (robot ($\hat{d}$istance near) ($\hat{s}$peed fast)).

Pattern1: (robot ($\hat{d}$istance close) ($\hat{s}$peed fast))
Pattern2: (robot ($\hat{d}$istance far) ($\hat{s}$peed medium))
Pattern3: (robot ($\hat{d}$istance very-far) ($\hat{s}$peed slow))
Pattern4: (robot ($\hat{d}$istance far) ($\hat{s}$peed slow))

Fig.16 shows how the fuzzy evidence is propagated through the Pattern Network given the fact *(robot ($\hat{d}$istance near) ($\hat{s}$peed fast))*. The number inside a pattern node is degree-of-match between the fuzzy term that corresponds to that node and the corresponding fuzzy term in the token. For example, at the node for the linguistic variable *distance* in the leftmost column in Fig.16, the degree-of-match corresponding to a match of the term *close* with the term *near* in the token is 0.6, the number shown inside the node. The minimum of this number and the degree-of-match number associated with the token is what gets passed down to the next node in the network. So, for the same node as before, the incoming token degree-of-match value is 1.0 and

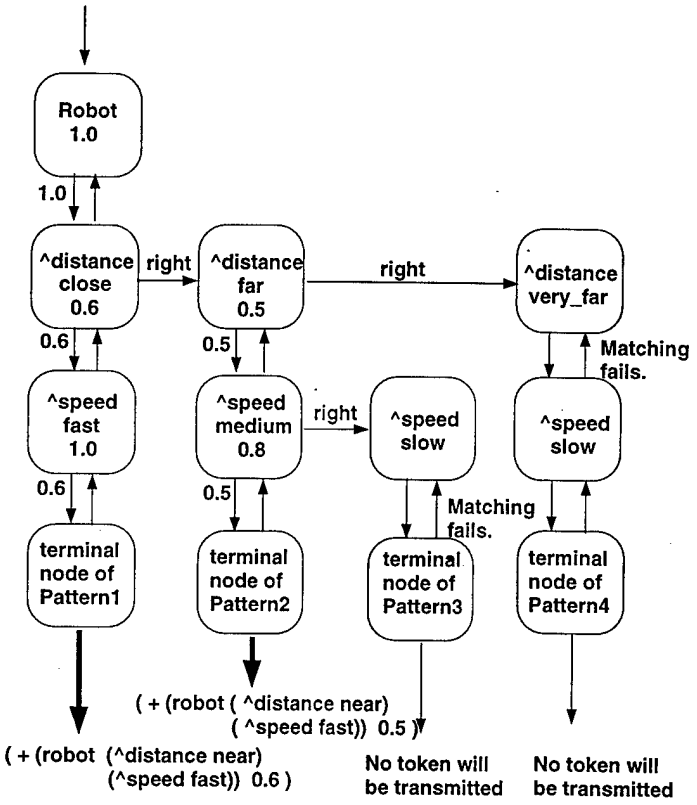

Figure 16: An example that shows how fuzzy evidence is propagated through the Pattern Network.

the local degree-of-match value is 0.6. So the token degree-of-match is reset to 0.6, as depicted by the number shown next to the downward arrow emanating from the node. The operation of taking the minimum of the local and the token degree-of-match at each node corresponds to the intersection operation discussed in section 3. Now consider the case of Pattern3. As shown in Fig.16, since *very-far* is not the overlapping term of *near*, there is no token produced by this branch of the network. The tokens will be directly propagated from the terminal pattern nodes to the join network.

5.3 Join Network

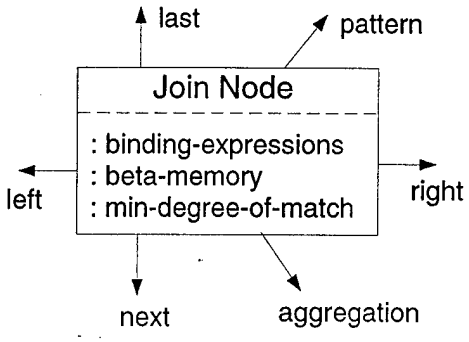

Figure 17: Representation of a join node.

In the previous section, we showed how the Pattern Network part of a Rete network can be generalized to handle matching for fuzzy strings. We will now do the same for the Join Network. Recall from Section 4, the purpose of a Join Network in a traditional expert system is to ensure that the bindings for the same variable in the different condition elements of a rule are consistent. In a fuzzy system, two different bindings for the same variable would be consistent if there is an overlap of the fuzzy sets for the two bindings.

Consider the following rule antecedent. The two separate arms of the Pattern Network, corresponding to the two condition elements shown, would come together at a join node as shown in Fig.18.

(rule-name rule-1
  (robot (n̂ame Peter) (d̂istance ?x) (ŝpeed ?y))
  (robot (n̂ame Alvin) (d̂istance ?x) (ŝpeed ?y))
⇒
  (. . .))

Now assume that the following two facts are in the working memory.

Fact-1: (robot (n̂ame Peter) (d̂istance far) (ŝpeed fast))
Fact-2: (robot (n̂ame Alvin) (d̂istance near) (ŝpeed medium))

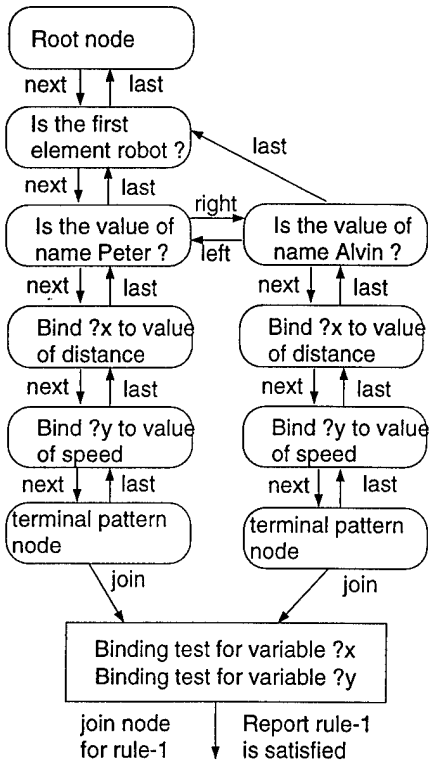

Figure 18: The join node for rule-1 will ensure that the bindings for the variables ?x and ?y are consistent.

Evidently, the match of Fact-1 and the first condition element will produce the instantiation *far* for the variable ?x and the instantiation of *fast* for the variable ?y. Similarly, the match of Fact-2 with the second condition element will yield the instantiation *near* for ?x and the instantiation *medium* for ?y. The join node shown in Fig.18 must now decide whether or not the different instantiations for the same variable are reconcilable on the basis of the overlap between the respective fuzzy sets. The join node carries out the following two steps:

1. It compares the fuzzy sets of all the different fuzzy sets associated with the same linguistic variable, such as *speed* in the above example, and then takes sup-min of the fuzzy membership functions.

2. The node calculates the minimum of all the sup-min numbers for all the different shared variables in the two inputs to the join node. This minimum must be non-zero for the join node to produce any output.

How a join node accomplishes all of the above will be clear from its representation shown in Fig.17. There are six pointers employed by a join node. Four links, *last*, *next*, *left*, and *right*, are used for connecting the nodes within the Join Network. The other two links, *pattern* and *aggregation*, provide the pathways to the Pattern Network and the Evidence Aggregation Network, respectively.

There are three kinds of information residing at each join node. The first, *binding-expressions*, is a pointer to the feature tests for the two inputs to ensure that the bindings for the variables shared by the rule antecedents are consistent as mentioned above. The second field, *min-degree-of-match*, is for storing the minimum of three such values, two coming from the two input tokens to the join node and one corresponding to the fuzzy match between the bindings for the variable in question. The value of *min-degree-of-match* for a join node is transmitted along with the token to the next level of the Join Network if the tests associated with that join node are passed. The last field, *beta-memory*, is the set of working memory elements associated with consistent variable bindings that have been tested by the join nodes up to and including the current one. Later we will have more to say about these different types of information stored at each join node.

In our case, a rule with N rule-antecedents will have N join nodes, with each join node having either one or two input nodes, as shown in Fig.19 for the case of two patterns from the first of the rule fragments shown below. The join nodes that feed rule instantiations into the agenda require two inputs, usually referred to as the left and the right memories in the literature on production systems; the *right* memory consists of what is pointed to by the *pattern* pointer and the left memory of what is pointed to by the *last* pointer. The join nodes that are more in the interior are of one input kind. These nodes are usually just below the terminal nodes of the Pattern Network and owe their existence to the resulting ease in programming.

With the help of an example we will now illustrate how the Join Network takes advantage of any structural similarity that might be present across different rules through shared use of join nodes. In general, the following three conditions must be satisfied to share a join node: First, all the previous join nodes must be shared. Second, the join node to be shared must be entered

from the same direction. And third, the binding expressions generated by what would otherwise be a new join node must be identical to the expressions generated when an already existing join node is instead shared. Consider the following three rules:

```
(rule-name Rule1
    Pattern1
    Pattern2
⇒
    ( . . . ))

(rule-name Rule2
    Pattern2
    Pattern3
⇒
    ( . . . ))

(rule-name Rule3
    Pattern1
    Pattern3
⇒
    ( . . . ))
```

where Pattern1, Pattern2 and Pattern3 are the different condition elements in the rules; these patterns result in a network like the one shown in Fig.15, with the leaves of this network being called the terminal pattern nodes. The join part of the network that results from processing Rule1 is shown in Fig.19.

While the field *pattern* is for creating links between the Pattern Network and Join Network, the fields *last*, *left*, *right*, and *next* in a join node are essential for creating the needed pathways for propagating the tokens. As mentioned before, at each join node the pointers *last* and *pattern* are pathways for the left and right memories, the former supplying all the partial matches in all the patterns encountered so far and the latter in the new pattern. Each join node is capable of querying the higher level nodes through these pathways and pulling in the partial matches. As for the downward pointers in Fig.19, they permit the system to focus on just those join nodes that are affected by the latest changes to the working memory. For example, if after the generalized Rete network is all assembled, two facts that can match Pattern1 and Pattern2 are placed in the working memory. Through the downward pointers *join* and *next*, Join2 node will be intimated that partial matches are available in its left memory. Via the *join* link, Join2 node will discover that information is also available in its right memory. With both memories being made available at Join2 node, it will go ahead and decide whether or not Rule1 should be placed on the agenda.

After processing Rule2, the Join Network will look as shown in Fig.20. Note how the *join* field of "terminal pattern node of Pattern2" is broken for inserting another join node, Join3, and the *right* pointer of Join3 node made to point to the Join2 node. This allows both the Join2

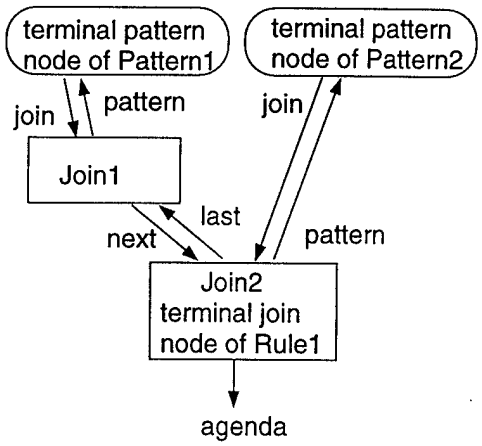

Figure 19: Join Network for Rule1.

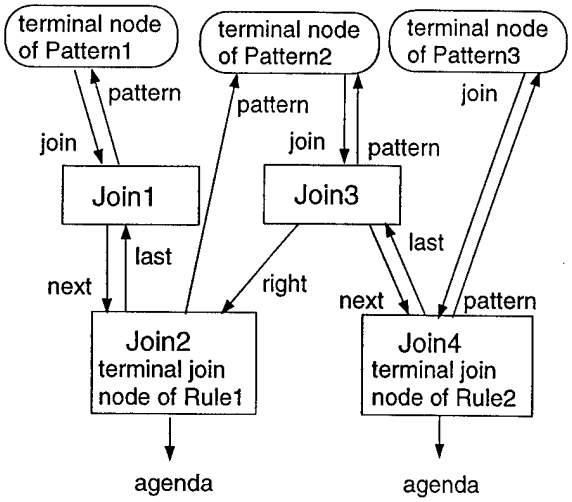

Figure 20: Join Network for Rule1 and Rule2.

and Join4 nodes to inherit all the nodes that correspond to Pattern2. At a still higher level of complexity, now consider what happens when Rule3 is also processed. The new network is as shown in Fig.21. The new rule shares Pattern1 with the first rule and Pattern3 with the second rule. Rule3 causes redirection of the *join* pointer of "terminal pattern node of Pattern3" node from Join4 node to the new node Join5. Note that this redirection leaves undisturbed the pathways to the left and the right memories at Join4 node, since the former is via the *last* field and the latter via the *pattern* field, both these remaining unchanged.

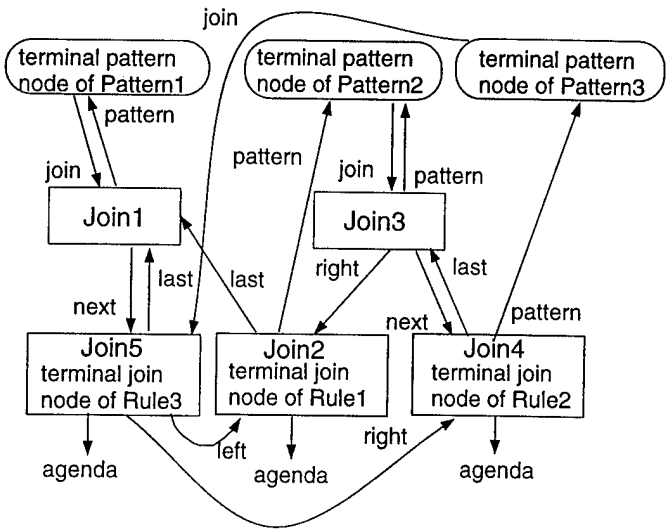

Figure 21: Join Network for Rule1, Rule2 and Rule3.

Before we describe how to propagate the fuzzy evidence through the Join Network, we'd like to mention the different roles played in this process by the *left* and the *right* links. When two different rules share the same join nodes from the left input (example: Join1 is shared on the left hand side by Join5 and Join2), that is represented by *left* link from, in this case, Join5 to Join2. On the other hand, when two join nodes share a pattern on the right hand side (example: Join5 and Join4 share Pattern3 on the right input), that is represented by link *right* from, in this case, Join5 to Join4.

As mentioned earlier, the fuzzy evidence is propagated from the terminal pattern nodes to the Join Network. Each join node contains the value of min-degree-of-match that was mentioned previously. This value will also be transmitted to other join nodes along with the tokens. Using the same strategy as in the Pattern Network, the tokens traverse the Join Network depth first. To explain further, for each join node, if the input token is received from the left input, a new token is propagated first via the link *next* assuming the tests on the variables are satisfied. The depth-first strategy will cause this token to be subsequently propagated via the link *left*. Similarly, if a token is received at the right input, a new token will be propagated first via the link *next*; subsequently, the same token will be propagated via the link *right*. Backtracking in this depth-first propagation is accomplished with the help of the pathways provided by the link *last*.

To illustrate the workings of join nodes in the network and the fuzzy inference operations that take place in these nodes, consider the following two-rule example and the facts *(robot (d̂istance far) (ŝpeed fast))*, *(mission (ĝoal find-object) (ŝpeed medium) (ŝonar-reading safe))*, *(sonar-reading safe)* and *(turn-angle right-30)*:

(rule-name rule1
(robot (d̂istance near)(ŝpeed ?x))
(mission (ĝoal find-object) (ŝpeed ?x)(ŝonar-reading ?y))
(sonar-reading ?y)
⇒
(. . .))

(rule-name rule2
(robot (d̂istance near)(ŝpeed ?x))
(turn-angle right-20)
⇒
(. . .))

Shown in Fig.4 are the membership functions of the fuzzy terms for *distance* and *speed*. We assume that the degree-of-match between *right-20* and *right-30* is 0.6. The section of the Join Network that would correspond to these two rule antecedents is shown in Fig.22. As shown in the figure, each node for the "terminal pattern node" of some pattern is equivalent to one condition element in a rule. There is a terminal join node where the condition-element branches come together for each rule and the output of the join node tells us whether or not a rule can be fired. Recall that the min-degree-of-match to be sent by a join node is set to be the minimum of the min-degree-of-match values for both inputs and the degree-of-match computed between those bindings for the variables. For example, the min-degree-of-match of the left input, Join1, for the Join2 shown in Fig.22 is 0.5 and the degree-of-match of the right input for the Join2 is 1.0. In addition, the sup-min number between *fast* and *medium* for the linguistic variable *speed* in Join2 is 0.8. Thus, the min-degree-of-match of Join2 is 0.5. With both inputs available in Join3, Join3 will check weather or not the bindings for the variable ?y are consistent. The sup-min number for the variable bindings of Join3 will be 1.0 since the bindings of the variable ?y for both inputs are bound to a crisp term, *safe*. Therefore, the min-degree-of-match of Join3 is 0.5. Since Join3 is a terminal join node, this min-degree-of-match associated with Join3 will be sent to the Evidence Aggregation Network by the link *aggregation*. Via the link *left*, the node Join4 can also receive the token sent by Join1. Since Join4 receives 0.5 and 0.6 from the two inputs for the fuzzy evidence, the min-degree-of-match of Join4 is 0.5. Subsequently, Join4 will report that rule2 is enabled. So far, we have described how to build the Pattern and the Join Networks for a set of rules and how to propagate the fuzzy evidence through these networks without sacrificing the computational efficiencies. We now show how to aggregate the fuzzy evidence when multiple rules make inferences about the same linguistic variable.

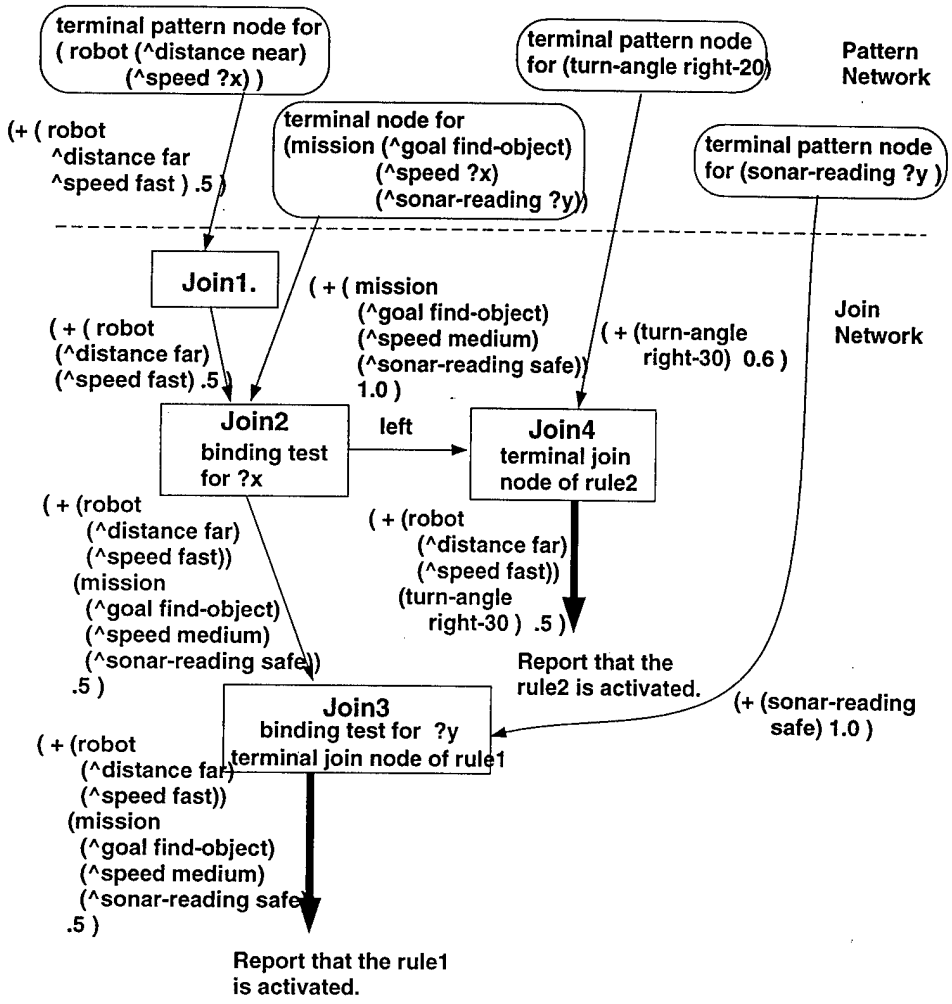

Figure 22: An example for propagating fuzzy evidence through the Join Network.

5.4 Evidence Aggregation Network

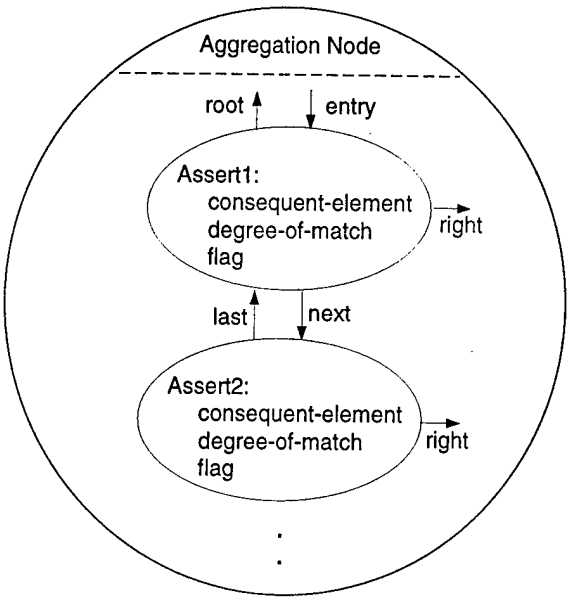

Figure 23: The representation of an aggregation node.

For a non-fuzzy expert system, the Pattern and the Join Networks perform the function of ascertaining which rules will be enabled by the facts in the working memory. As was mentioned earlier, this is all a Rete network has to do for a traditional expert systems. However, the Rete network for a fuzzy-logic based system must go one step further to solve the difficulty 1) introduced in Section 1 and make sure that all the fuzzy evidence for a given linguistic variable is aggregated before that linguistic variable is allowed to trigger any further rules. In this section we will show how this is accomplished by incorporating a third network, called the Evidence Aggregation Network, in the Rete network. It is the Evidence Aggregation Network that allows FuzzyShell to aggregate all the terms in all the consequents of the rules in the Agenda. This is accomplished before any of the rule is actually fired by the inference engine.

Basic to the Evidence Aggregation Network is an aggregation node, shown in Fig. 23. An aggregation node can have multiple inputs from the terminal join nodes. During the network building process, the action elements of rules are scanned and parsed, and the rules whose consequents make assertions about the same linguistic variable are made to point to the same aggregation node. At the same time, a sub-node for each consequent element is created and made internal to the node for the respective linguistic variable. All such sub-nodes inside an aggregation node constitute a doubly linked list. In addition to the forward and backward pointers of a doubly linked list, each sub-node in an aggregation node may also contain what we refer to as a *right* pointer. The purpose of this pointer is to allow us to deal with multiple rules whose consequents make assertions about multiple linguistic variables. Also, each sub-node contains the following slots: *degree-of-match*, *consequent-element*, and *flag*. The purpose of the *right* link and the slots will now be explained with the help of the following example.

Consider, for illustration, the following three rules:

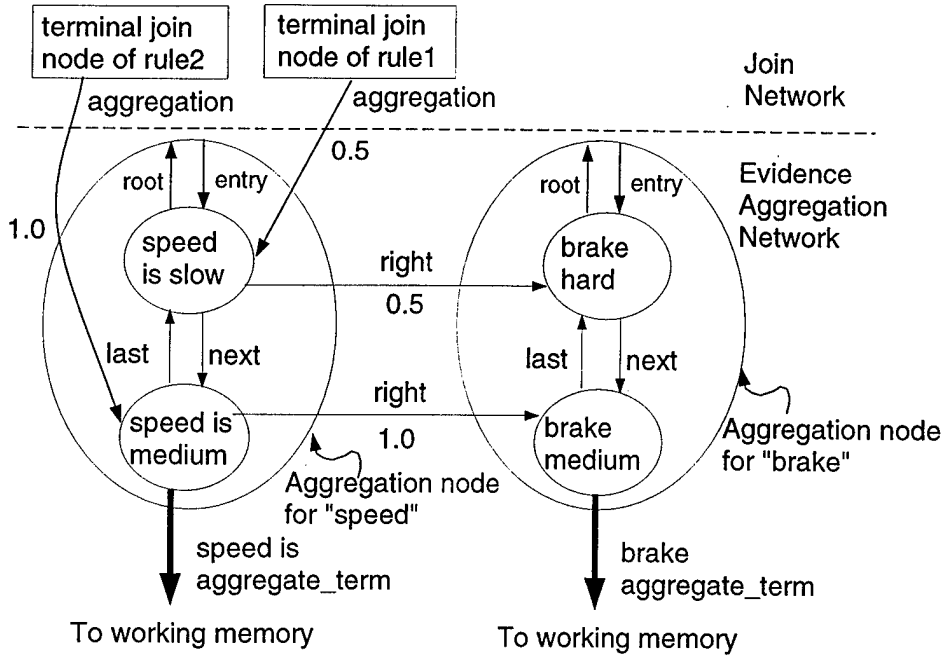

Figure 24: This example shows how fuzzy evidence is aggregated in the Evidence Aggregation Network.

rule1: IF (distance is near) THEN (speed is slow) (brake hard)
rule2: IF (distance is far) THEN (speed is medium)(brake medium)
rule3: IF (speed is slow) THEN (. . .)

The complete Evidence Aggregation Network for these rules is shown in Fig.24. The left node is for the linguistic variable *speed* and the right node for *brake*, the other linguistic variable appearing in the consequents of the rules. With the help of the *right* pointer attached to the first sub-node in the node for *speed*, the system knows that the assertions (speed is slow) and (brake hard) belong to the same consequent. Similarly, via the right pointer attached to the sub-node (speed is medium) the system knows that the assertions (speed is medium) and (brake medium) come from the same rule. Also, note that the two pointers used to glue the linked list of consequent items to an aggregation node are called the *root* and *entry* for reasons that are obvious from the figure.

To explain how evidence gets aggregated by the Evidence Aggregation Network, we will consider the same set of rules as above and assume that the fact in the working memory is (distance is far). As shown in Fig.24, 0.5 represents the degree-of-match between the fact (distance is far) and the rule antecedent (distance is near) while 1.0 is the degree-of-match for the antecedent (distance is far) and the same fact. After rule1 is enabled, the aggregation sub-node corresponding to (speed is slow) will receive the value 0.5 from the terminal join node of rule1 via the link *aggregation*. Subsequently, the aggregation sub-node representing (brake hard) will also receive the value 0.5 via the link *right*. Next, after rule2 is enabled, the aggregation sub-node corre-

sponding to (speed is medium) will receive the value 1.0 form the terminal join node of rule2 and transmit it to the aggregation sub-node for (brake medium) via the link *right*. Finally, the results of the aggregation for the linguistic variables *speed* and *brake* will be asserted into the working memory.

6 An Example to Illustrate the Workings of FuzzyShell

In what follows, we will use a simple example to show the workings of FuzzyShell. In particular, our example will illustrate how FuzzyShell addresses the first difficulty presented in the Introduction, namely, the difficulty of ensuring that the evidence from all the enabled rules containing the same consequent linguistic variable is aggregated before any other rules can be invoked. This example concerns a mobile robot that should either circumnavigate an object if the mission is obstacle avoidance, or that should approach the object if the mission is object recognition. In each case, the final inference can only be drawn after a two-level inference, meaning that the rules fired initially must enable other rules whose consequents constitute the final conclusions. For the sake of our explanation here, the following six rules will suffice. Note, however, that an actual system for mobile robot navigation will contain hundreds of rules and that many of the rules would be far more complicated than what we are able to show here.

Fact-1: (robot (m̂ission obstacle-avoidance)(ŝensor 2D-vision))
Fact-2: (2D-vision (p̂ixel-position-x med-left)(p̂ixel-position-y far))

(rule-name rule-1
    (robot (m̂ission ?x)(ŝensor 2D-vision))
    (2D-vision (p̂ixel-position-x left) (p̂ixel-position-y near))
⇒
    (goal (m̂ission ?x)(ôrientation NWW)))

(rule-name rule-2
    (robot (m̂ission ?x)(ŝensor 2D-vision))
    (2D-vision (p̂ixel-position-x med-left)(p̂ixel-position-y far))
⇒
    (goal (m̂ission ?x)(ôrientation NW)))

(rule-name rule-3
    (goal (m̂ission obstacle-avoidance)(ôrientation NW))
⇒
    (t̂urn-angle right-20)
    (d̂istance-to-travel medium))

(rule-name rule-4
    (goal (m̂ission obstacle-avoidance)(ôrientation NNW))

⇒
(t̂urn-angle right-30)
(d̂istance-to-travel long))

(rule-name rule-5
(goal (m̂ission object-recognition)(ôrientation NWW))
⇒
(t̂urn-angle left-40)
(d̂istance-to-travel short))

(rule-name rule-6
(goal (m̂ission object-recognition)(ôrientation NW))
⇒
(t̂urn-angle left-30)
(d̂istance-to-travel long))

The membership functions of the fuzzy terms for the linguistic variables *pixel-position-x, pixel-position-y, orientation, distance-to-travel* and *turn-angle* are shown in Fig.25. Assume that the depth-first strategy is selected for conflict resolution, meaning that a rule matched with the most recent facts will have a higher priority to be fired and the rules with the same priority will be randomly selected for inclusion in the agenda. After Fact-1 and Fact-2 have been asserted into the working memory to start the inference engine, rule-1 and rule-2 will be activated and placed in the agenda. Since both these rules contain the same linguistic variable in their consequent sides, the Evidence Aggregation Network will go ahead and compute the final membership function for this linguistic variable, taking into account the sup-min values propagated down by the Pattern and the Join Networks and the prior membership function for the variable.

Since rule-1 and rule-2 are matched with the same facts, they have the same priority to be fired. Assume rule-1 is fired first. The firing of this rule will cause the membership function of the *orientation* variable to be updated with the latest membership function computed for this variable by the Evidence Aggregation Network. Fig.26 illustrates the inference process resulting from the enabling of rule-1 and rule-2 and the firing of rule-1. It is important to note that the fuzzy evidence of the variable *orientation* was updated on the basis of the two rules, rule-1 and rule2, even though only rule-1 has been fired so far. Therefore, as far as fuzzy inference is concerned, we may think of rule-1 and rule-2 as being bundled together. But note that bundling together does not imply that rule-2 should be fired immediately after rule-1. In general, the consequent side of a rule (such as rule-2) will include action elements that are non-fuzzy; the inference process with regard to these action elements must proceed in the traditional manner.

Firing of rule-1 will cause the following fact to be asserted in the working memory:

Fact-3: (goal (m̂ission obstacle-avoidance)(ôrientation aggregate_term)

This new fact will now be encapsulated in a token and propagated through the Pattern Net-

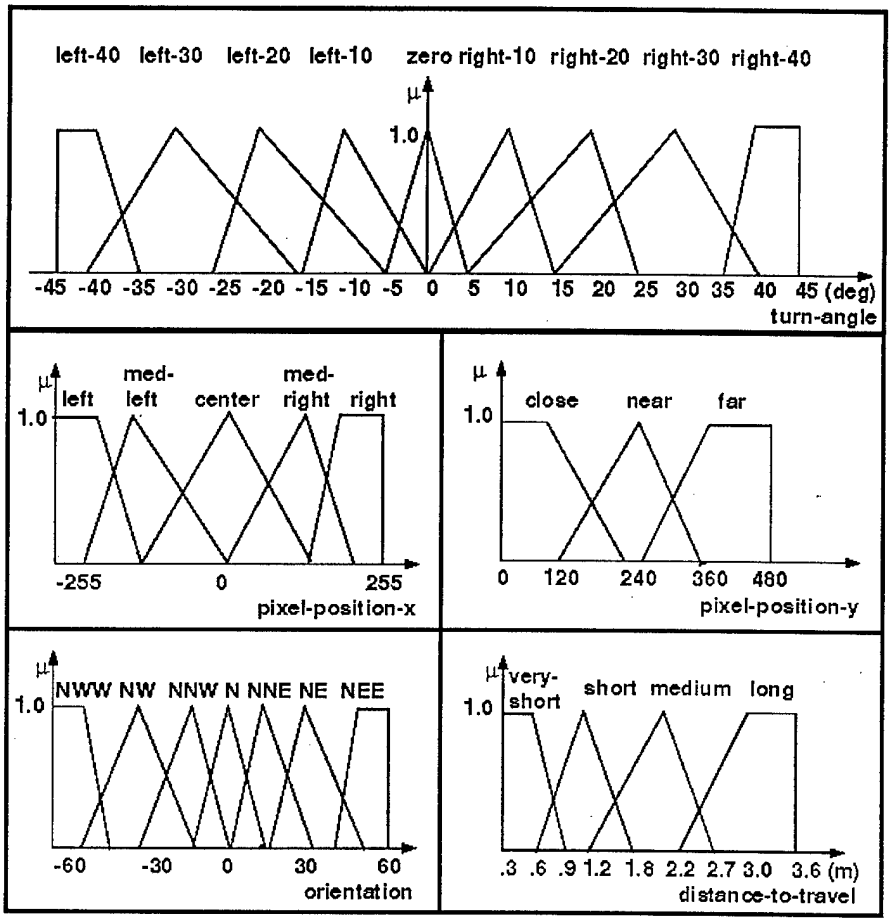

Figure 25: Shown here are the membership functions of the fuzzy terms for the linguistic variables *pixel-position-x*, *pixel-position-y*, *orientation*, *distance-to-travel*, and *turn-angle*.

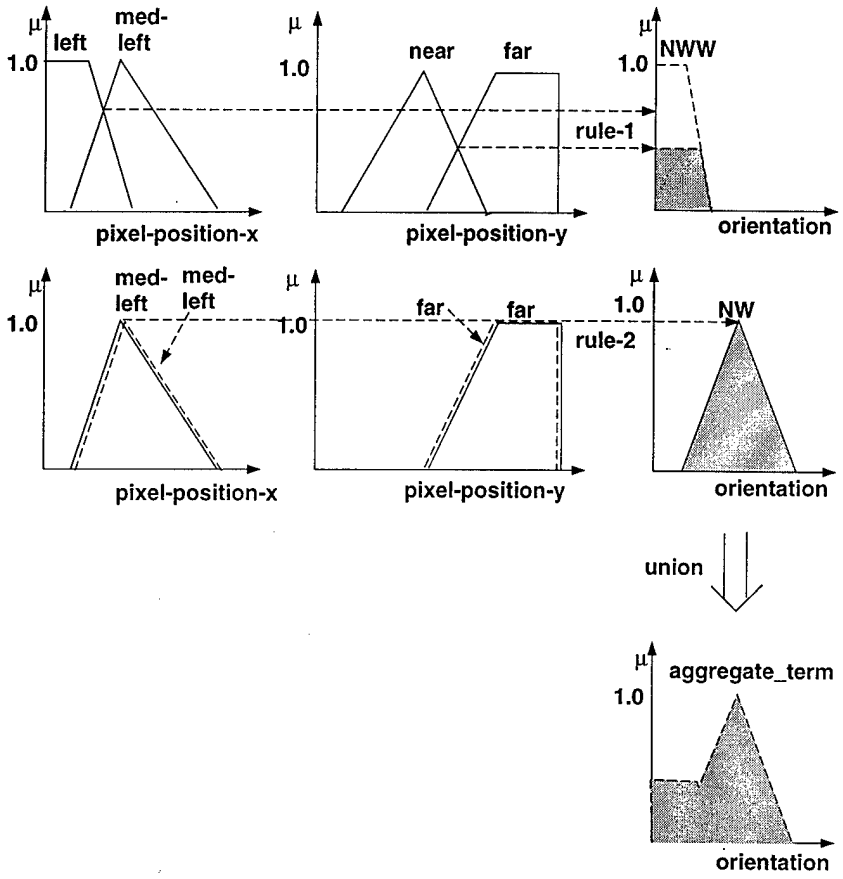

Figure 26: This figure illustrates how the membership function of *aggregate_term* for the linguistic variable *orientation* is aggregated in the first cycle of the inference engine.

work; as a result, the rules rule-3 and rule-4 will be enabled and placed in the agenda. At this time, the agenda will contain rule-3, rule-4 and rule-2. Since rule-3 and rule-4 are matched with the same fact, Fact-3, they have the same priority. In addition, their priority is higher than the priority of rule-2 since Fact-3 is the most recent fact. Recognizing that the enabled rule-3 and rule-4 contain the same linguistic variables in the consequents, the Evidence Aggregation Network will go ahead and figure out the updated membership function for the variables *turn-angle* and *distance-to-travel* taking into account, via the Pattern and the Join Networks, the fuzzy constraints generated by the condition-elements of the two rules. Assume rule-3 is fired first. As a result, the membership functions for the linguistic variables *turn-angle* and *distance-to-travel* will be updated to those computed by Evidence Aggregation Network, as pictorially illustrated in Fig.27.

Now the agenda will contain rule-4 and rule-2 for firing. Since the former has a higher priority (owing to the higher recency number of the fact that enabled this rule), it will be fired next. Since, for the fuzzy part of the inference, the rule was already accounted for when it was first placed in the agenda, the firing of this rule would cause changes only if the consequent side contained non-fuzzy elements. In our example, no further changes would take place as a result of the firing of rule-4. Finally, rule-2 will be fired. Again, for reasons identical to those just mentioned, this firing will not cause any changes to the currently known facts. [7]

If the purpose of a fuzzy expert system is to only reason, then the evidence shown in Fig.27 for *turn-angle* and *distance-to-travel* will be the conclusive evidence for the robot motion. However, the expert system usually must make its overall conclusions known to the world in terms of numerical values for the different linguistic variables. FuzzyShell provides a simple command to defuzzify the linguistic variables based on the "center of gravity" method.

7 Conclusion

While it is relatively easy to program up a small-scale fuzzy expert system, the same cannot be said when hundreds or thousands of rules are involved, as is common with large-scale expert systems. Every fuzzy reasoning system that allows chains of inference, meaning that a rule is allowed to trigger other rules, must correctly address the first of the difficulties mentioned in the Introduction. The fuzzy expert system shells that are currently available commercially or as free-ware on the internet satisfy only one of these requirements for multi-step fuzzy inference. As was mentioned before, the reason for this failure in the software systems currently available is clear: these systems are a result of a mostly cosmetic modification of the traditional non-fuzzy expert system shells. On the other hand, in the work reported here we have completely modified the guts of an inference engine – the Rete network – to make it work correctly for the fuzzy case. How this Rete network is modified is one of the main contribution of our work.

[7] Although we have shown the two rules, rule-5 and rule-6, those rules will not be fired since the current mission of the robot is to avoid the obstacle.

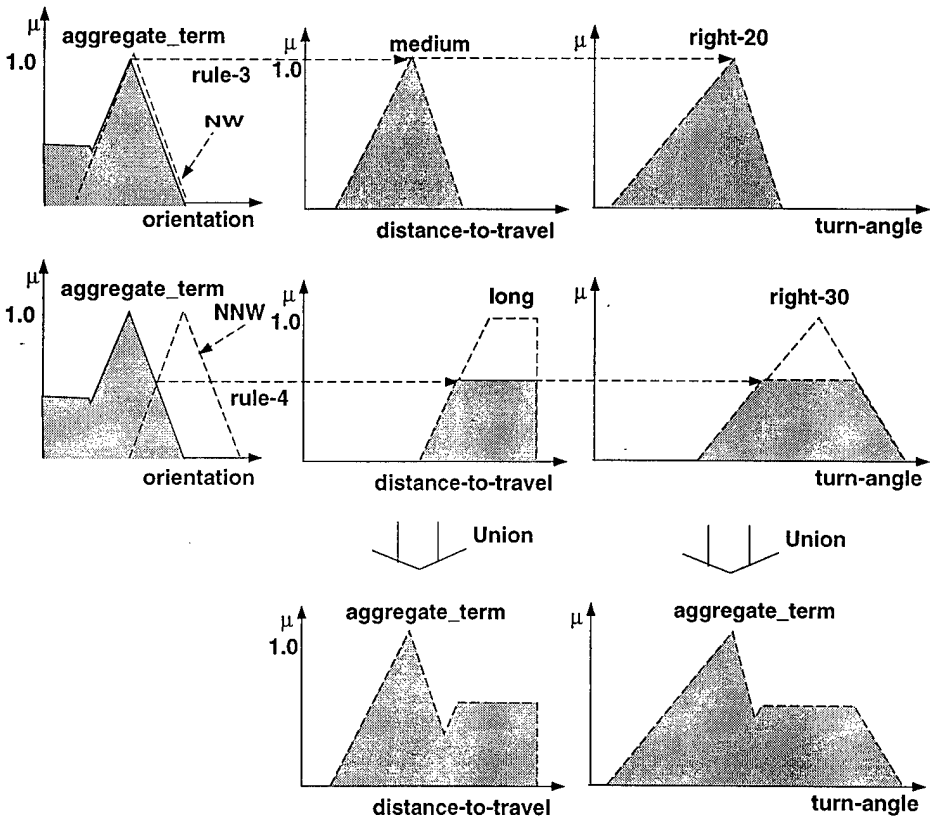

Figure 27: This figure illustrates how the membership functions of *aggregate_term* for the linguistic variables *turn-angle* and *distance-to-travel* are aggregated in the second cycle of the inference engine.

References

[1] K.P. Adlassnig, "Fuzzy set theory in medical diagnosis," *IEEE Transaction on Systems, Man and Cybernetics*, Vol. 16, pp. 260-265, 1986.

[2] K.P. Adlassing, G. Kolarz, "Representation and Semiautomatic Acquisition of Medical Knowledge in Cadiag-1 and Cadiag-2," *Computers and Biomedical Research*, Vol. 19, pp. 63-79, 1988.

[3] J.F. Baldwin, "Evidential support logic programming," *Fuzzy Sets and Systems,* Vol.24, pp. 1-26, 1987.

[4] T. Bilgic and I. B. Turksen, "Effective search methods for pattern matching inferencing using specific similarity measures," *1992 IEEE Int'l Conference on Fuzzy Systems*, pp. 161-167, 1992.

[5] P. Bonissone, S. Gans and S. Decker, "RUM: A layered architecture for reasoning with uncertainty," *10th Int'l Joint Conference on Artificial Intelligence*, pp. 891-898, 1987.

[6] L. Brownstone, R. Farrell, E. Kant, and N. Martin, Programming expert systems in OPS5, *Addison-Wesley Publishing Co.* , 1985.

[7] J. Buckley, W. Siler and D. Tucker "Fuzzy expert system," *Fuzzy Sets and Systems,* Vol.20, no. 1, pp. 1-16, 1986.

[8] J. Buckley and D. Tucker, "Second generation fuzzy expert system," *Fuzzy Sets and Systems*, Vol. 31, pp. 271-284, 1989.

[9] CLIPS User's Manual, version 5.1, Jan. 6th, 1992, Software Technology Branch, Lyndon B. Johnson Space Center.

[10] D. Driankov, H. Hellendoorn, and M. Reinfrank, An Introduction to Fuzzy Control, *Springer-Verlag*, 1993.

[11] C. Elkan, "The paradoxical success of fuzzy logic," *Proceedings of the 11th National Conference on Artificial Intelligence,* pp. 698-703, 1993.

[12] C. Elkan, "The paradoxical success of fuzzy logic," *IEEE Expert*, pp. 3-8, August, 1994.

[13] H. Farreny, H. Prade and E. Wyss, "Approximate reasoning in a rule-based expert system using possibility theory: a case study," *Proc. 10th World Computer Congress(IFIP)*, pp. 407-413, 1986.

[14] C. L. Forgy, "On the efficient implementation of production systems," *Ph.D. thesis, Department of Computer Science*, CMU, Feb., 1979.

[15] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *AI*, Vol. 19, pp. 17-37, 1982.

[16] C. L. Forgy, "The OPS83 user's manual," *Production Systems Technologies*, 1989.

[17] FuzzyCLIPS Version 6.02A User's Guide, National Research Council Canada, 1994.

[18] J. Giarratano and G. Riley, "Expert systems : principles and programming," *PWS Publishing Company*, 1994.

[19] P.L.K. Jones, "REVEAL: an expert systems support environment", in *Expert Systems: Principles and Case Studies,* R. Forsythe, Ed., first edition(Chapman & Hall, London, 1984).

[20] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller – Part 2," *IEEE Transaction on Systems, Man and Cybernetics* , Vol. 20, pp. 419-435, 1990.

[21] K. S. Leung, W. S. Wong and W. Lam, "Applications of a novel fuzzy expert system shell," *Expert Systems: The Int'l J. Knowledge Engineering*, Vol. 6, no. 1, pp. 2-10, 1989.

[22] E. H. Mamdani, "Application of fuzzy logic to approximate reasoning using linguistic synthesis", *IEEE Transaction on Computers*, Vol. 26, no. 12, pp. 1182-1191, 1977.

[23] M. Mizumoto and H. Zimmermann, "Comparison of fuzzy reasoning methods," *Fuzzy Sets and Systems*, Vol. 8, pp. 253-283, 1982.

[24] Allen Newell and Herbert A. Simon, Human Problem Solving, *Prentice-Hall*, 1972.

[25] Z.A. Sosnowski, "FLISP - a language for processing fuzzy data," *Fuzzy Sets and Systems,* Vol. 37, pp. 23-32, 1990.

[26] I.B. Turksen, Y. Tian and M. Berg, "A fuzzy expert system for a service centre of spare parts," I. J. of Expert Systems with Applications, Vol. 5, pp. 447-464, 1992.

[27] T. Whalen, B. Schott and F. Ganoe, "Fault diagnosis in fuzzy networks," Proc. in Int'l Conf. Cybernetics and Society, *IEEE Press*, 1982.

[28] R. R. Yager and L. A. Zadeh, "An introduction to fuzzy logic applications in intelligent systems," *Kluwer Academic Pub.* 1992.

[29] R. R. Yager and L. A. Zadeh, "Fuzzy sets, neural networks and soft computing," *Van Nostrand Reinhold*, 1994.

[30] L. A. Zadeh, "The role of fuzzy logic in the management of uncertainty in expert system," *Fuzzy Sets and Systems 11*, North-Holland, pp. 199-227, 1983.

[31] L. A. Zadeh, "Fuzzy logic," *IEEE Computer*, Vol. 21, no. 4, pp. 83-93, 1988.

[32] L. A. Zadeh, "The calculus of fuzzy if/then rules," *AI Expert*, March, pp. 23-27, 1992.