A Fuzzy Rule-based Decision Support System Environment

Thesis submitted
in partial fulfillment of the requirements
for the award of the degree of
Doctor of Philosophy
by

Narasimha Bolloju

School of Mathematics and Computer/Information Sciences
University of Hyderabad
Hyderabad - 500 046
INDIA

March 1991

Certificate

This is to certify that the thesis entitled "A Fuzzy Rule-based Decision Support System Environment" being submitted by Narasimha Bolloju in fulfillment of the partial requirements for the award of the degree of Doctor of Philosophy in Computer Science to the University of Hyderabad, is a record of bonafide work carried out by him under our supervision.

The results presented in this thesis have not been submitted to any other University or Institute for award of any degree or diploma.

Candidate

Narasimha Bolloju
CMC Ltd.
Secunderabad

Supervisors

Dr. Naveen Prakash
Indian Institute of Technology
Kanpur

Dr. A. S. Reddy
University of Hyderabad
Hyderabad

Dean
School of Mathematics and Computer/Information Sciences
University of Hyderabad
Hyderabad - 500134
INDIA

To
My Family

Acknowledgements

I would not have begun this research but for the encouragement and support of Naveen. He continually kept guiding me all through the research. If this thesis presents clearly my ideas and contributions, it is largely due to his critical comments and suggestions. My sincere thanks to Naveen for making all this possible.

I thank A S Reddy, my second supervisor, for his guidance in this research. I also thank him for his suggestions in improving the manuscript.

I also thank Murali Mohan, C Ravishankar and many other colleagues at CMC who have helped me at various stages of this research activity.

I thank CMC for providing me the facilities for this research.

Finally, I thank my wife Jyoti and daughters Sameera and Sneha for the support and bearing with my absence at home.

Table of Contents

A Fuzzy Rule-based Decision Support System Environment

Abstract

This thesis proposes a fuzzy rule-based decision support system environment aimed at handling semi-structured decision problems at higher organisational levels.

A three layered decision modelling formalism based on decision models, decision stages and decision rules is proposed. Decision models, at the top of the hierarchy, represent decomposed decision problems and specification of their composition using decision stages. Decision stages represent the smallest decision problems, and consist of a specification of probable alternatives and decision rules to be employed in constraining and ranking these probable alternatives. Lastly, decision rules, at the bottom of the hierarchy, represent the basic modelling knowledge as a high level declarative fuzzy rules.

Imprecision and uncertainty in decision rules and decision parameters is represented using the concepts of fuzzy sets, possibility distributions, and necessity-possibility measures.

The decision support environment presented in this thesis integrates a number of concepts from AI and ES into the generic DSS framework.

The knowledge system comprises of modelling knowledge and environmental knowledge components. The modelling knowledge component, based on the formalism mentioned above, represents complete domain specific knowledge. The environmental knowledge component comprises of procedural and factual knowledge and it is represented by Prolog-like rules with extensions to represent imprecision and uncertainty using fuzzy sets, possibility distributions, and necessity- possibility measures.

The language system provides a non-procedural command oriented language interface for model execution and information retrieval. The decision models can be activated for execution either by specifying the name of the decision model or by formulating new models. A fuzzy query language provides the facilities to retrieve information from the database and the knowledge base.

The feasibility of the proposed environment is demonstrated by an implementation in Prolog on a VAX 11/750. The effectiveness of this environment in solving semi-structured decision problems is illustrated with an example.

List of Symbols

N	Necessity **measure** definition
c	constant value
r_c	condition rule evaluation function
r_a	action rule evaluation function
r_v	term evaluation function
n	Necessity measure
a_i	Argument name
a_f	Aggregate function name
$\mathbf{I}$	Assert action predicate
$\mathbf{D}$	Retract action predicate
$\mathbf{R}$	Replace action predicate
$\mathcal{Q}$	Fuzzy quantifier
$\mathcal{Q}_m$	Modifier to fuzzy quantifier
$x, y, \ldots$	Variables
$\mathbf{p}, \mathbf{p}_1, \mathbf{p}_2, \ldots$	Predicate symbols

α	An alternative of a decision stage
β	A value from a given base domain *B*
γ	A fact (or a component of factual knowledge)
ε	Null action
ϕ	An action rule (also used as *null set)*
ψ	A condition rule
κ	A rank rule
λ	A rule (or a component of procedural knowledge)
μ	A decision model (also used as *membership function)*
π	Possibility measure
θ	Comparison operator
ρ	A decision rule
σ	A decision stage
τ	A term
ω	A probable alternative

A	Set of **alternatives** (feasible)
B	Base domain
Γ	Set of facts
Λ	Set of rules
M	Set of decision models
Π	Possibility measure definition
P	Set of decision rules
Σ	Set of decision stages
Ω	Set of probable alternatives

$\exists$	Existential quantifier	$\forall$	Universal quantifier
$\bot$	Certain false	$\top$	Certain truth
$\otimes$	Unit alternative	$\oplus$	**Exclusive-or**
$\cup$	Set Union	$\cap$	Set Intersection
$\wedge$	Conjunction (and)	$\vee$	Disjunction (or)
$\neg$	Negation (not)		

inf	**infimum** (aggregate function)
sup	supremum -do-
min	minimum -do-
max	maximum **-do-**
min	minimum (2 or 3 argument function)
max	maximum -do-

M_n	Decision model name
M_p	Decision model parameters
A_{M_n}	Set of decision models with name M_n
S_{M_n}	Set of applicable models for a given invocation of M_n

R_n	Decision rule name
R_p	Decision rule parameters
A_{R_n}	Set of decision rules with name R_n
S_{R_n}	Set of applicable rules for a given invocation of R

Chapter 1
Introduction

1.1 Decision Support Systems

The concepts of Decision Support Systems (DSS) were first introduced in early 70s under the term management decision systems. Initial advances in this area took, by and large, the form of specific DSS applications developed in various organisations. These systems were primarily designed and developed to meet specific organisational needs and were meant to support some decision making activity.

Decision problems can be characterised on two dimensions, viz. degree of structuredness and the organisational level addressed. The degree of structuredness of a decision problem can be expressed as:

a) structured,
b) semi-structured, or
c) unstructured.

Structured decision problems are well-defined with respect to the identification of goals and algorithms or procedures, and problems of this kind are amenable to analytical models. Proven techniques such as Management Information Systems (MIS)/ Operations Research (OR)/ Management Science (MS) can be applied to these. Further, decision making in structured problems can be automated to a large extent. Examples under this category include resource allocation,

location-allocation, and assignment problems at lower organisational levels. On the contrary, no algorithms or procedures exist for unstructured problem solving and they require a good amount of human intelligence. In this class of problems it is almost impossible to provide any worthwhile support in decision makings. The problems of interest, therefore, are the semi-structured problems which require a combination of formal and informal procedures for solving them. Since this class of problems cannot be automated completely, it is a good area for systems based on human-computer interaction in which the computer takes over the part of the problem that can be automated and the decision maker controls the remaining

Dutta (1987) characterises decision problems on the dimension of structuredness by identifying the differences in the level of

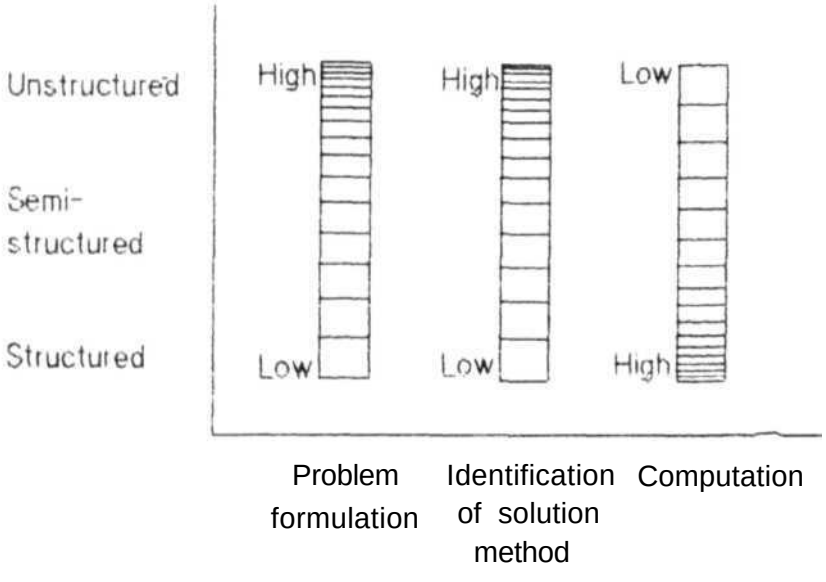


Fig. 1.1 Proportion of problem solving effort

understanding of the underlying reasoning process and the ease of externalising such a process. In an alternate approach, Dutta and Basu (1985) characterise decision problems on the dimension of structuredness based on the effort required in solving the problems as shown in Fig. 1.1.

On the organisational dimension, the decision problems can be classified under strategic, tactical and operational levels with strategic level at the top of the hierarchy. At the operational level, which is the lowest, majority of the problems are not open-ended and almost all the problem parameters are under the control of the decision maker. As we move higher in the organisational levels the problems become more open-ended, their parameters become increasingly difficult to quantify precisely and certainly, and a number of parameters go beyond the control of the decision maker. The decision models of such problems at these levels are, by and large, informal and the modelling knowledge is often uncertain.

In due course researchers and developers in the area of DSS began identifying characteristics which are common to DSS. One of the important characteristics is that the DSS are aimed at solving semi-structured decision problems and in doing so they use data and analysis models. These systems are interactive and meant for helping, not replacing, the decision makers. They are flexible and adaptable to accommodate changes in the environment and the decision making approach of the user. Further, they improve the effectiveness

in decision making rather than the efficiency and operate under the control of the decision maker.

For quite some time there was not a single accepted definition of DSS because of the breadth of application areas and kind of decision making support involved. However, today, the definition of DSS given by Sprague as "interactive computer based systems designed to help decision makers in solving ill-structured problems using data and models" is widely accepted (Sprague 1980).

In the next section we shall discuss some important frameworks of DSS and survey various related issues. Thereafter, we identify the requirements of DSS and the state-of-the-art in handling semi-structured decision problems at higher organisational levels, i.e. strategic and tactical levels. We shall, then, briefly describe the modelling methodology proposed in this thesis. The last section details the layout of the thesis.

1.2 DSS Frameworks and Related Issues

A number of research workers have proposed the general architecture/ framework of DSS (Sprague 1980; Bonczek *et al* 1984; Bosnian 1983; Teng *et al* 1988). All these proposals were centered around identifying the components of DSS and the interrelationships and interactions between these components. The components typically identified are a) database or knowledge base, b) model base, c) user interface, and d) problem- processing system,

Sprague's Framework:

Sprague proposed a framework of DSS and presented three perspectives of DSS from the points of view of the end-user, the builder and the tool-smith (Sprague 1982). Further, based on the above perspective,

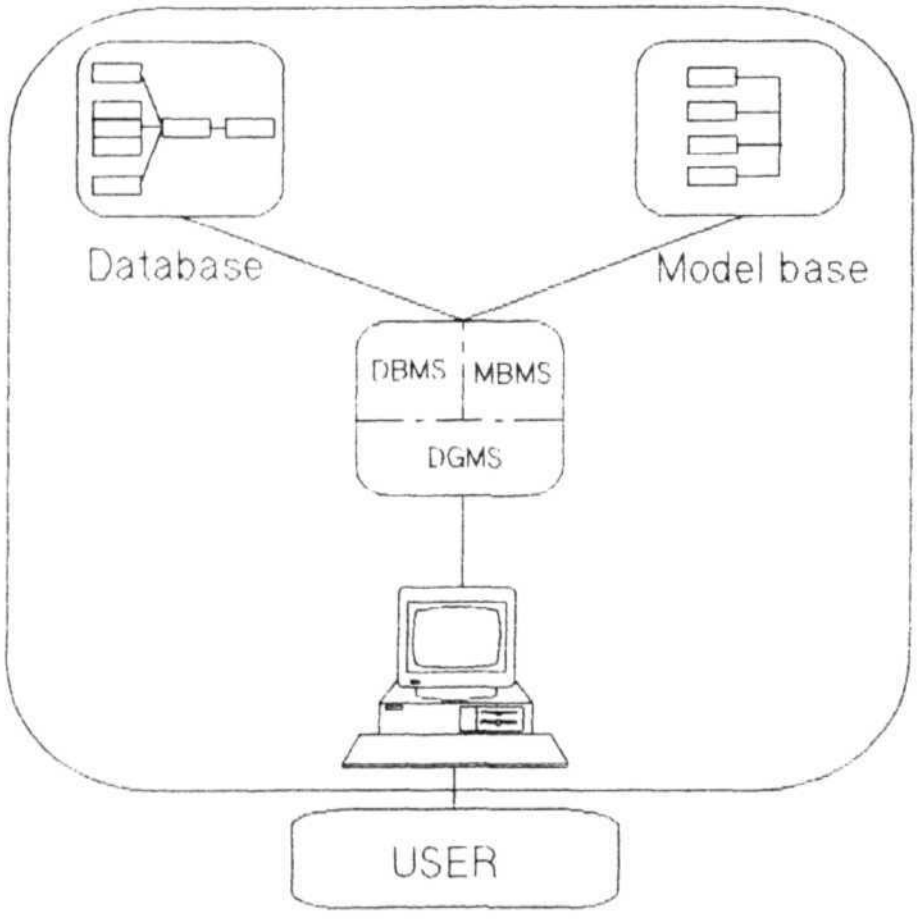


Fig. 1.2 Components of DSS

he identified three technology levels, specific DSS, DSS generator

and DSS tools for these three perspectives respectively. According to him a DSS, shown in Fig. 1.2, consists of the following three components:

a) Dialogue Generation and Management System (DGMS) that defines the user-interface using which the decision maker interacts with the DSS. and it supports functions such as accepting requests for model execution, data retrieval, etc.

b) Model Base Management System (MBMS) that provides facilities for creation of new models, cataloging and maintaining a wide range of models, interrelating various models, etc.

c) Data Base Management System (DBMS) that takes care of the data management functions such as data definition, creation, maintenance, retrievals/updates, etc. and facilities for representing data from external sources and extraction of relevant data from organisational databases.

Databases of DSS are comprised of data from external and internal sources to varying degrees compared to non-DSS databases. Also, there is more adhocism in the operations of non-DSS databases that in those normally performed on DSS databases. Model base is a collection of models that are of interest to the specific DSS application. It is possible to manipulate the models, in a manner similar to the manipulation of data in databases, and to integrate models to form complex models.

Framework of Bonczek et al

Bonczek *et al* (1984) proposed a generic architecture of DSS (Fig. 1.3) as consisting of the three following systems:

- Knowledge System,
- Language System, and
- Problem-processing System.

User requests for data retrieval and model execution are specified using the facilities offered by the language systems (LS). The

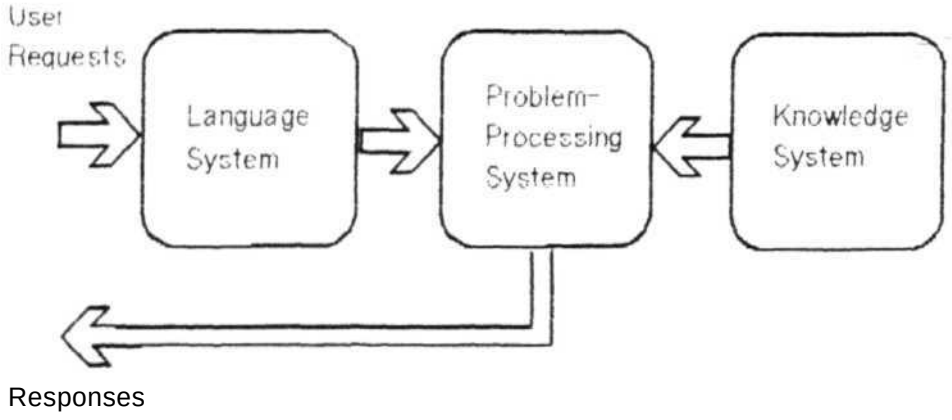


Fig. 1.3 DSS Framework of Bonczek *et al*

modelling knowledge that is to be used in solving decision problems and the environmental knowledge that is referred to by the modelling knowledge form the knowledge system (KS) The problem-processing system accepts the requests from the user specified in LS and uses the environmental knowledge and modelling knowledge to provide the results to the user.

Sol (1983) presented an integrated framework based on the proposals

made by Sprague (1980), Bonczck *et al* (1984) and (Bosman 1983). Teng *et al* (1988) present a unified architecture for intelligent DSS by integrating the concepts of expert systems and DSS.

In the rest of this section we shall orient the discussion around the framework proposed by Bonczek *et al.* This is because of our concern with decision problems at higher organisational levels. Such problems have many characteristics common with those found in problems being modelled as Expert Systems, particularly with respect to knowledge representation and problem solving approaches. Therefore, we believe that a discussion oriented around this architecture would be more appropriate. This viewpoint is supported by McLean and Sol (1986) who point out that it is easier to relate and compare DSS versus AI/ES using the framework of Bonczek *et al.*

1.2.1 Knowledge System

Knowledge system, as mentioned above, represents the environmental knowledge and the modelling knowledge in order to enable the problem-processing system to respond to the user requests issued through the language system. In the framework proposed by Sprague (1982) these two components were treated separately as data subsystem and model subsystem respectively.

In this thesis, *environmental knowledge* refers to the data subsystem consisting of factual and procedural knowledge. On the other hand, the modelling knowledge component refers to a specific class of decision problems which are defined over and above environmental

knowledge.

1.2.1.1 Environmental Knowledge

In this subsection we shall discuss the environmental knowledge aspect of the knowledge system. Requests for information retrieval from the LS are processed against the environmental knowledge. These request can be initiated either by the user or by the problem processing system during model execution.

The possible approaches to realise this component can be a collection of tiles of some application or a database. Majority of the developments in the representation and usage of environmental knowledge took place under the three following areas:

a) programming languages,
b) database management, and
c) artificial intelligence.

We shall briefly discuss the various stages of this development process and their limitations which induced further research.

The information systems prior to 70s were based on conventional file systems using different types of file organisations. Programming languages provided the data definition and manipulation capabilities as an integral part of them. Therefore, the programs were dependent on the organisation/layout of the data and needed a lot of maintenance along with the other inherent problems such as redundancy, inconsistencies, etc. Majority of the initial DSS have

been developed based on this approach using languages like FORTRAN, COBOL.

Database technology evolved to solve these problems and two distinct schools of research emerged under relational model of data and network model of data (Date 1981; Ullman 1984). A large number of corporate databases have been implemented and a number of specific DSS have been built based on these using high level data manipulation facilities offered by the database management systems (DBMS). It was generally argued that the user interfaces of relational databases such as SQL, QBE, QUEL, etc. are more user-friendly in comparison to those of network databases. However, query languages similar to the above do exist on network databases (Parimala *et al* 1985).

Independent developments in AI generated technology that can be used to tackle most of the above mentioned problems on smaller data/knowledge bases. As a result a number of knowledge representation schemes have been identified such as first order logic, production rules, semantic nets, frames, etc (Nilsson 1980; Cohen and Fiegenbaum 1982; Rich 1983; Jackson 1986). Knowledge representation based on first-order logic (FOL) (Grey 1984) is, in the opinion of this author, one of the simplest and earliest technique identified to represent knowledge and make inferences. The programming language Prolog (Cloksin and Mellish 1981) is based on first-order predicate calculus in which programs are defined as a set of axioms in FOL and the execution of a program amounts to

proving a theorem on these axioms. Some researchers argue that FOL offers an excellent framework for representation and manipulation of data because data bases, particularly relational, query languages and programming language Prolog are based on the FOL (Dutta and Basu 1984; Dutta 1987). Also, Coelho and Rodrigues (1983) favour FOL as a knowledge representation scheme in their proposal for a three-layered architecture for management environments.

Semantic nets is a graphical knowledge representation technique best suited for representation of generalisation- specialisation concepts as a graph consisting of isa arcs and part of arcs. A decision aiding application employing semantic nets for knowledge representation is reported in (Widmeyer and Lee 1986).

Frame based schemes of knowledge representation use a slot-filler approach to capture complex conceptual hierarchies. While presenting an application of a frame based language, SRL, for a job shop scheduling problem, Fox (1985) stresses on the representation and sharing of information across multiple applications.

Production systems based knowledge representation scheme captures a more informal and empirical knowledge using a set of if-then rules. A number of expert systems (MYCIN, PROSPECTOR, etc.) have been built based on the rule based knowledge representation scheme. Some of the advantages cited in favour of this scheme are a) they naturally represent the domain expert's knowledge, and b) the knowledge captured is modular and easier to maintain (Jackson 1986). A comparison of DSS and expert systems is presented in appendix C.

1.2.1.2 Modelling Knowledge

The unique aspect of DSS is the research and the concepts related to decision modelling. Initial decision models were based on one or more of the techniques from Operations Research, Statistics and Management Science. Most of these techniques cater either to structured decision problems or to decision problems having very little unstructuredness. The decision models based on these are quite formal and the decision makers are required to be knowledgeable with respect to their usage. Further, majority of the analytical techniques produce optimal solutions whereas in decision making it is practically impossible to provide optimal solutions and therefore, the decision makers usually require satisficing solutions. Keen (1986) points out that the optimisation science for decision making is too often unrealistic, impractical, artificial and unused.

Since decision makers, in practical situations, use informal models, a number of systems have been developed to provide decision modelling facilities using relatively more informal equation oriented techniques. Although such models represent semi-structured problems to some extent, they still do not correspond closely enough to the real world situation. Bosnian (1986) cautions that the equation based models can be dangerous as cognitive aids especially when applied by decision makers without much knowledge of model constraints and procedures. He, further, criticises that many DSS

modelling facilities are oriented towards the choice phase and rarely for modelling activities such as identification of alternatives, problem formulation, etc. McLean and Sol (1986) argue that the applicability of equation type models is limited to very few cases and suggest usage of process type models. On the correspondence of decision models with reality, Bosman and Sol (1985) point out that the generality of equation oriented models is high and their correspondence with reality is low. In process oriented informal models the converse is true.

Decision making becomes more and more complex as problems exhibit more unstructuredness. Holtzman and Breese (1986) put forward the reasons which can be attributed to the difficulties in decision making as:

- lack of clarity
- inadequate structural understanding
- dissatisfaction with the available alternatives
- inferential complexity
- combinatorial complexity, and
- inability to deal with uncertainty.

Subsequently they identify the support required in the areas of generating alternatives, construction of decision models, measuring uncertainty, etc.

Representation of modelling knowledge using production rules can be an effective tool for handling decision problems at higher organisational levels. This can be justified on the premise that

majority of the modelling knowledge is in the form of rule, guidelines, etc. and therefore, a rule based representational scheme would be more appropriate. The successful application of this methodology in a number of expert systems has demonstrated the utility and the associated benefits (Jackson 1986). A comprehensive comparison of expert systems and decision support systems is presented in appendix C.

Problem Decomposition and Model Management

The basic assumption made in problem decomposition in decision making is that judgment is improved when a complex, ill-defined problem is decomposed, analysed and solved by a set of smaller, more well-defined problems (Lagomasino and Sage 1985). It appears that one major approach in order to solve complex decision problems is the divide and conquer method. In order to support this kind of decomposition we require facilities for representation and manipulation of multiple decision models.

Bonczek *et al* (1983) and Blanning (1985,1986) advocate the application of relational (data model) theory for management of decision models. Blanning (1986) extends the definition of relation to include decision models as virtual relations as "a subset of Cartesian product of a set of domains corresponding to the input and output parameters of the model". Under this framework multiple decision models can be combined using relational join and a election operation produces results of model execution.

The effectiveness of first-order logic for model management has been illustrated by Dutta and Basu (1984, 1985). They argue that the inference mechanism can implicitly, as a byproduct of the proof procedure, handle the selection and execution of multiple models without the need for user control.

1.2.2 Language System

Language system is essentially the user interface of DSS which accepts the commands from the user to perform various functions. Two important functions of LS are:

a) knowledge base query (data retrieval), and
b) request for model execution.

The first function enables the user to specify information retrieval from KB using the query commands of LS. This function is extensively used for exploratory retrievals to analyse the modelling knowledge. These commands to query can be either procedural or non-procedural. While using procedural query languages the user explicitly specifies how the information is to be retrieved, step-by-step. On the other hand, the non-procedural query languages can be used by merely stating what is expected and the system determines bow the requested information is to be retrieved. Query languages such as SQL, QUEL (based on relational calculus) are examples of non-procedural languages (Date 1981; Ullman 1984). Query languages based on logic can, in addition to providing non-procedural commands, facilitate querying on derived/procedural

knowledge. Additionally, it is possible to specify recursive queries in such languages. Finally, another kind of query languages is the fuzzy query languages where results can be retrieved by specifying approximate conditions (see appendix E).

The other function of LS, mentioned above, is handling of user requests for model execution. Again, these requests can be procedural where the user explicitly identifies the models and the values of various parameters. On the other hand, the user can request model execution in terms of the requirement, and may not explicitly identify the models and their sequence of execution.

Among the other functions of LS, knowledge system maintenance is the most essential function. This function enables users to assert new knowledge and replace/retract existing knowledge.

1.2.3 Problem-Processing System

The problem-processing system (PPS) is the central component of a DSS. While the knowledge system and the language system are representational, the PPS is a dynamic software component (Bonczek *et al* 1983). This system accepts commands from the user specified in LS, retrieves the knowledge available in KS, uses this knowledge to process the user requests, and presents the derived results back to the user using the presentation knowledge. Bonczek *et al* (1984) identify the abilities of a PPS as:

- Information collection
- Problem recognition
- Model formulation
- Analysis
- Implementation

PPS uses its information collection ability to obtain the requests specified in LS and, also, to obtain the knowledge available in KS. The problem recognition ability recognises various aspects of the problem not explicitly specified in the user requests. Through the model formulation ability, PPS identifies the models and their sequence of execution required to solve the user problem. In order to achieve this PPS uses the model management functions discussed in the previous subsection. The analysis ability corresponds to inferring/interpreting the user requests against the knowledge present in KS. Finally, implementation ability is used to present the result to the user and/or reflect a choice in the knowledge base for further use.

1.3 Imprecision and Uncertainty

As **we** have discussed in section 1.2 the decision problems at higher organisational levels are pervaded by imprecision and uncertainty. It is, therefore, necessary to consider the methodologies available for representation and manipulation of this aspect.

Imprecision arises when a variable or an attribute that is supposed to take on a single value from a domain is defined with a set of

probable values. On the other hand, if the truth (or falsity) of a given value cannot be established then that value is known to be uncertain. It is possible that the values could be both imprecise and uncertain as shown in the table 1.1.

	Precise	Imprecise
Certain	20 25	about 25 between 20 and 24 young
Uncertain	may be 25 probably 30	may be about 25 probably young

Table 1.1 Imprecise and uncertain values

Imprecision and uncertainty can be present, in addition to the attribute values, in various components of modelling and environmental knowledge. Bolloju (1990a) discusses the presence of imprecision and uncertainty in objects, attributes, relationships and integrity constraints.

The impact of imprecision and uncertainty can be seen on all the three systems of the framework. Firstly, it is necessary to represent imprecision and uncertainty in various components of the knowledge system. Then, having represented in knowledge system, language system should facilitate imprecision and uncertainty in the user requests. Lastly, problem-processing system should be able to process the requests that are specified with imprecise and uncertain elements against the imprecise and uncertain knowledge in the knowledge system. A brief survey of fuzzy databases and fuzzy query

languages is presented in appendix E.

1.3.1 Imprecision

Fuzzy sets and the possibility distributions are two, more or less equivalent, approaches for the representation of imprecision. A fuzzy set A (Zadeh 1978) is defined by its membership function (μ) which maps the elements of a universe of discourse U to values between 0 and 1. Given a proposition of the form 'X is **A'**, the membership function μ_A restricts the possible values of X on U as:

$$\mu_A : X \rightarrow [0,1]$$

Fuzzy sets can be used in either conjunctive or disjunctive notion. In conjunctive notion all the elements of the fuzzy set are members with varying degrees of membership; and X is generally considered to be multivalued. On the other hand, in the disjunctive notion the membership value $\mu_A(u)$ is interpreted as the possibility of the **single-valued** X taking on the value u. Under this interpretation, possibility distribution is equivalent to fuzzy set as defined by;

$$\Pi_X(u) = \mu_A(u) \qquad \forall\, u \in U$$

1.3.2 Uncertainty

Some well-known approaches for the representation of uncertainty are listed below:

a) Bayesian theory

b) Degrees of belief and disbelief

c) Dempster-Shafer theory of evidence

d) Necessity-Possibility measures.

Bayesian Theory

Representation and propagation of uncertainty in the past has been based on the probability theory. Bayesian approach is built on the strong mathematical foundations of probability theory. Given an evidence *e* related to hypothesis ***h*, Bayes'** theorem states:

$$P(h \mid e) = \frac{P(e \mid h) \cdot P(h)}{P(e)}$$

where

P(*e*) and **P(*h*)** are the respective prior probabilities of evidence *e* and hypothesis *h*,

P(*e* | *h*) is the conditional probability of evidence ***e*** given the hypothesis *h*, and,

P(*h* | *e*) is the posterior probability of the **hypothesis** *h* given the evidence ***e***

In many practical applications, the estimation of the prior and conditional probabilities is a difficult task because of **the** large numbers of estimates involved. **Further,usage** of subjective estimates of probability can give rise to inconsistencies (Dutta 1987). It was suggested that through the assumption of conditional independence it is possible **to** reduce the number of probabilities to be estimated.

Degrees of Belief and Disbelief

A number of expert systems, such as **MYCIN,** employ the concept of *certainty factors* based on degrees of belief and disbelief. Under this approach a certainty factor (between -1 and 1) is associated with each hypothesis for a given evidence. The certainty factor, CF, for a given hypothesis *h* and evidence *e* is derived from:

$CF(h,e) = MB(h,e) - MD(h,e)$

where

MB(*h*,*e*) is the degree or measure of increased belief in *h* based on ***e***, and

MD(*h*,*e*) is the degree or measure of increased disbelief in *h* given *e.*

Evidence from conjuncted (disjuncted) propositions is combined using minimum (maximum) of the CFs of the individual propositions. Then, the confidence in hypothesis (or consequent) is estimated as a product of the certainty factor associated with the rule and the certainty factor derived from the combined evidence (or antecedents).

This approach makes departures from the axioms of probability theory. Rigorous estimation of probabilities are avoided through the use of certainty factors and it was generally felt that the experts are more comfortable in assigning certainty factors (Buchanan and Shortliffc **1984).**

Dempster-Shafer Theory of Evidence

The Dempster-Shafer theory of evidence is based on degrees of belief to represent uncertainty. In Baycsian theory, probability distributions are defined over the individual hypotheses. However, in Dempster-Shafer theory, the probability distributions are constructed over all *subsets* of hypotheses. That is, a probability estimate can be assigned to a given subset of hypotheses collectively, without dividing the estimate among the individual hypotheses. This property of Dempster-Shafer theory enables one to represent the ignorance regarding the dominance within the individual hypotheses of a subset and makes it an attractive technique for representation of uncertainty compared to Bayesian theory.

Under the Dempster-Shafer theory, the set of all relevant propositions is called the *frame of discernment 0.* All the elements of *6* are assumed to be mutually exclusive and exhaustive. Then, the theory introduces a belief function, *Bel,* which is defined using a *basic probability assignment* function, ***m***, as:

$$Bel\{h) = \sum_{e_i \text{ entails } h} m(e)$$

That is, belief in hypothesis *h* is the sum of the basic probability assignments of all the evidences *e.* entailing *h.* The theory provides mechanism to combine evidences using the basic probability assignments to derive further degrees of

belief.

Necessity-Possibility Measures

The representation of uncertainty using necessity-possibility measures is based on possibility theory and fuzzy logic (Zadeh 1985; Pradc 1985a, 1985b). In this approach a pair of measures is associated with propositions to indicate the uncertainty. And, the necessity measure of a proposition *p* corresponds to the impossibility that *p* is false. That is, given the possibility measure $\Pi(p)$ and $\Pi(\neg p)$ representing the possibility of *p* being true and false respectively, the necessity measure $N(p)$ is defined as:

$$N(p) = 1 - \Pi(\neg p)$$

The theory, further, identifies possible functions to combine these measures in conjunction and disjunction, and propagate uncertainty in implication.

Other Approaches

In addition to these four approaches discussed above, there have been a number of approaches for the representation and propagation of uncertainty. For example, Cohen (1985) proposes an approach termed *model of endorsements* for representation and propagation of uncertainty. He describes the abilities of the approach and compares it with Bayesian approach. Baldwin (1987a; 1987b) reports an approach to uncertainty based on the combination of evidence theory **and** fuzzy set theory in a logic programming style. This

approach provides a pair of supports to represent uncertainty and a calculi for the propagation of uncertainty. A comparison of four uncertainty calculi along with their respective computational complexities is reported in (Henkind and Harrison 1988).

1.4 Motivation

Analysing the discussion in the previous section, we arrive at the conclusion that DSS technology lags behind requirements in the following aspects:

a) decision modelling methodology to faithfully capture decision problems at higher organisational levels,

b) integrated decision support environments facilitating requisite tools and techniques in an unified framework, and

c) bridging the gap between theory and practice.

In the remaining part of this section we shall elaborate these aspects and identify the related state-of-the-art in DSS.

Decision Modelling Methodology: In order to model complex decision problems at higher organisational levels, the modelling methodology should offer the following facilities:

- problem decomposition
- informal rule based modelling
- representation and propagation of imprecision and uncertainty in modelling knowledge

The requirements and the associated benefits of decomposition of a

complex, ill-defined decision problem into many smaller, **well-defined** decision problems have been discussed earlier. Though the first-order logic is helpful in composing smaller decision models into a complex model through the inference mechanism (Dutta and Basu 1984), it just offers *one kind* of model composition. For a more general composition of decision problems we still require better methods such as conditional composition, iterative composition, etc.

Now let us consider the modelling of the smaller and relatively better-defined decision problems. Firstly, it was strongly felt that the equation oriented decision models are not applicable to the area of decision making under consideration. Hence, the obvious alternative is production rule based decision modelling. This approach is at a fairly higher level and it is easier to adapt to the changing requirements. However, the level of production rules used in expert systems needs to be reconsidered. We believe that majority of the decision problems require facilities, such as quantification and aggregation, which are not common in production rule systems.

Various approaches for representation and propagation of imprecision and uncertainty in environmental knowledge are direct applications from **AI/ES** technology. Unfortunately, this aspect in decision modelling is either avoided completely or partially handled.

Integrated **Decision Support** Environment: We can summarise, from the discussion in section 1.2, the various facilities required in

decision support as:

- integration of knowledge bases and databases
- support for representation and manipulation of data from both internal and external sources
- representation and propagation of imprecision and uncertainty
- non-procedural model formulation
- high level non-procedural approximate query languages
- domain independent problem-processing systems

The first three of the above refer to the knowledge system component. As many of the DSS applications are built on top of some large databases, it is mandatory to access, possibly transparently, the external databases along with the local knowledge bases. Further, it is possible that the data and/or the knowledge could contain imprecise and uncertain information. Moreover, the knowledge system should provide facilities to represent and manipulate data from external sources such as rumours, personal beliefs, etc.

The user interface provided by the language system should be non-procedural and high level in specification. More importantly, these facilities must be in tune or identical to those in model specification.

The last facility cited above refers to the PPS component. Domain independence in PPS is an important requirement, since many decision problems at higher organisational levels are ad hoc and non-repetitive in nature. This domain independence assists the

decision maker in effectively utilising the same environment for many decision problems.

Gap between Theory and Practice: Majority of the research, in solving semi-structured problems at higher organisational levels, is either in the form of proposals of frameworks/architectures (Bonczek *et al* 1984; Bosman 1983,1986; Bosman and Sol 1985; Sol 83; Sprague 1980) or simplified application implementations (Alley *et al* 1979; Badia and Martin-Clouaire 1989; Binaghi *et al* 1989; Dickinson and Ferrel 1985; Green-Hall 1985).

Considering the state-of-the-art, the spreadsheets or equivalent equation based DSS with relational database interfaces are the most common ones. The availability of better modelling environments and tools has not changed significantly from the initial frameworks of Sprague (1980) and Bonczek *et al* (1984). A recent book (Hopple 1988) on the state-of-the-art in DSS also confirms this belief. The major stress in practical systems appears to be oriented towards better presentation of results (e.g., graphics) only and not on modelling.

1.5 The Proposal

We propose a fuzzy rule based decision support system environment aimed at handling semi-structured decision problems at higher organisational levels. The thesis proposes a decision modelling formalism and demonstrates its effectiveness using an integrated decision support environment. We shall now broadly describe the

framework presented in this thesis.

Decision Modelling Formalism

A three layered decision modelling formalism based on decision models, decision stages and decision rules is proposed. Decision models, at the top of the hierarchy, represent the decomposed decision problems and specification of the composition of the decision problems built using decision stages. Decision stages represent the smallest decision problems, and consist of a specification of probable alternatives and the decision rules to be employed in constraining and ranking the alternatives. Lastly, decision rules, at the bottom of the hierarchy, represent the basic modelling knowledge (available in the form of rules, guidelines, etc.) as high level declarative fuzzy rules. Imprecision and uncertainty in decision rules and decision parameters is represented using concepts of fuzzy sets, possibility distributions, and necessity-possibility measures.

Integrated Decision Support Environment

The decision support environment presented in this thesis integrates a number of concepts from AI and ES into the generic DSS framework proposed by Bonczek *et al* (1984). This integration is domain independent and it is aimed at enhancing the effectiveness of decision makers in solving ad hoc and non-repetitive semi-structured decision problems.

The knowledge system comprises of modelling knowledge and environmental knowledge components. The modelling knowledge

component, based on the formalism mentioned above, represents complete domain specific knowledge. The environmental knowledge system component is referred by the modelling knowledge component and it is in the form of procedural and factual knowledge. Procedural knowledge of this component can be specified using Prolog-like rules with extensions to represent imprecision and uncertainty using fuzzy sets, possibility distributions, and necessity- possibility measures. A significant part of the factual knowledge can be in the form of an external database.

The language system provides a non-procedural command - oriented language interface for model execution and information retrieval. The decision models can be activated for execution either by a mere specification of the name of the decision model or by a composition or formulation of new models. Again, the decision parameters, to either of these, can be imprecise and uncertain. A fuzzy query language provides the facilities to retrieve information from the database and the knowledge base.

The PPS component is realised using a layered interpreter built on top of a fuzzy inference engine. It is possible to define heuristic knowledge to control the generation of alternatives. Further, it is possible to define the functions to be employed in the propagation of uncertainty during the inference.

The Gap between Theory and Practice:

The feasibility of the proposed environment is demonstrated using an implementation in Prolog on a VAX 11/750. The effectiveness of this

environment in solving semi-structured decision problems is presented by modelling a decision problem.

We feel that a framework based on the above is a step towards a generalised DSS environment catering to semi-structured decision problems at higher organizational levels. This thesis also demonstrates the application of various AI techniques employed in expert systems, knowledge representation, problem solving, and fuzzy logic and databases in decision support systems. Finally, we believe that such a framework would bridge the existing gap between research and practice in DSS that are meant for solving semi-structured decision problems at higher organisational levels.

1.6 Layout of the Thesis

An overview of the proposed DSS environment is presented in chapter 2. The architecture of this environment is illustrated and a description of various components of the system and their functions and abilities is presented along with the inter-relationships between these components. An example decision problem is also presented in detail.

Chapter 3 describes the knowledge system with its two knowledge components. Formal definitions of modelling knowledge and environmental knowledge are presented in this chapter. Subsequently, the proposed syntax of various components of the modelling knowledge is presented and illustrated with examples. Lastly, syntax and examples of the environmental knowledge **are**

presented in this chapter.

The language system of the DSS environment is presented in chapter 4. Model activation and formulation and information retrieval based on fuzzy constructs is presented formally. Later, some examples, based on the unified language system, are included to illustrate the user-interface to the proposed environment.

The operational semantics of interpretation at various knowledge levels is presented in the next chapter on PPS. Initially, the preliminary operations such as semantic unification, fuzzy arithmetic, etc. are defined in detail. Subsequently, the formal definition of semantics behind the interpretation of decision models, decision stages and decision rules are presented. Then, the processing of exploratory information retrieval functions is defined formally.

Chapter 6 describes a prototype implementation of the proposed DSS environment, Plan Aid. The advantages of Prolog for the implementation are discussed. Then, various algorithms of interpretation, semantic unification, etc. are presented using Prolog clauses.

The conclusions and scope for future extensions are discussed in the chapter 7. Also, some of the limitations of this proposed system and possible ways of tackling these are touched upon.

The syntax of the decision models, decision stages and decision rules is presented in Appendix A. In addition, the fuzzy query

language syntax is included in this appendix. A complete example session with the prototype implementation is presented in Appendix B. The next three appendices present discussion on the relevance of AI/ES technology under the titles of Comparison of DSS and ES, Fuzzy Expert System Shells and Fuzzy Databases and Query languages. Lastly, the appendix F presents definition of the triangular norms.

Chapter 2
An Overview

2.1 Introduction

We shall present an overview of the proposed DSS environment in this chapter. In section 2.2 we shall elaborate the characteristics of decision problems and the decision modelling process. Then, in section 2.3, we shall present the architecture of the proposed DSS environment briefly describing various components of the architecture. An example of decision problem is discussed in the section 2.4 with details of the related modelling knowledge.

2.2 DSS Environment

We have discussed that the decision making process is informal and decision makers employ the knowledge which is in the form of rules, guidelines, etc. in solving semi-structured decision problems. It is, therefore, convenient to express such knowledge in declarative rule based specification rather than algorithmic specification. A majority of these rules are often of imprecise and uncertain nature. We have also discussed that fuzzy sets and possibility-necessity measures can together capture both imprecision and uncertainty. Consequently, we have adopted a fuzzy rule based specification of modelling knowledge for knowledge representation in this thesis.

The proposed decision support system environment encapsulates the

complete set of functions that are required for decision making- That is,

a) a knowledge system with fuzzy rule based specification for modelling and environmental knowledge components,
b) a language system with facilities to activate and formulate decision models, and to perform exploratory information retrieval, and
c) a problem-processing system to process the user requests specified in language system against the knowledge in knowledge system.

Decision Models and Decision Modelling:

The term "decision model" refers to a representation of some real world decision problem. Decision modelling is a process of capturing the real world problem as a decision model. We have observed in the introductory chapter that the characteristics and complexity of this process differs with the organizational level and the structuredness of the decision problem concerned. Further, the modelling requirements and tools used in modelling considerably differ on these characteristics.

The process of decision modelling can be described as consisting of the following steps. Firstly, the decision problem will have to be identified and decomposed into smaller problems which can be tackled in a relatively easier way. Subsequently, for each of these subproblems a class of probable alternatives will have to identified

along with the constraints that are to be applied on these alternatives to select feasible alternatives. Further, the characteristics which can be used in comparing alternatives against each other will have to be defined for each subproblem.

Once a decision problem is modelled following the above process the decision maker can use it for analysing the effects of a particular decision by selecting an alternative and studying its consequences. The decision maker can also change the decision parameters and/or decision rules to perform analyses such as sensitivity analysis, what-if analysis, etc.

2.3 Architecture of the Proposed DSS Environment

The architecture of the proposed DSS environment resembles the one proposed by Bonczek *et al* (1984). The knowledge system is spread across two components, modelling knowledge and environmental knowledge. The language system offers facilities for model activation and formulation, and information retrieval. Lastly, the problem-processing system consists of a layered interpreters (decision model, stage, rule interpreters), fuzzy query processor and a fuzzy inference engine. The architecture is depicted in fig. 2.1. An informal discussion on this architecture and the decision modelling formalism is reported in (Bolloju 1989a).

2.3.1 Knowledge System

The knowledge system is divided into modelling knowledge and

environmental knowledge components. Modelling knowledge is specific to the decision problem domain and it is possible that it represents one or more decision problems of a given domain. On the other hand,

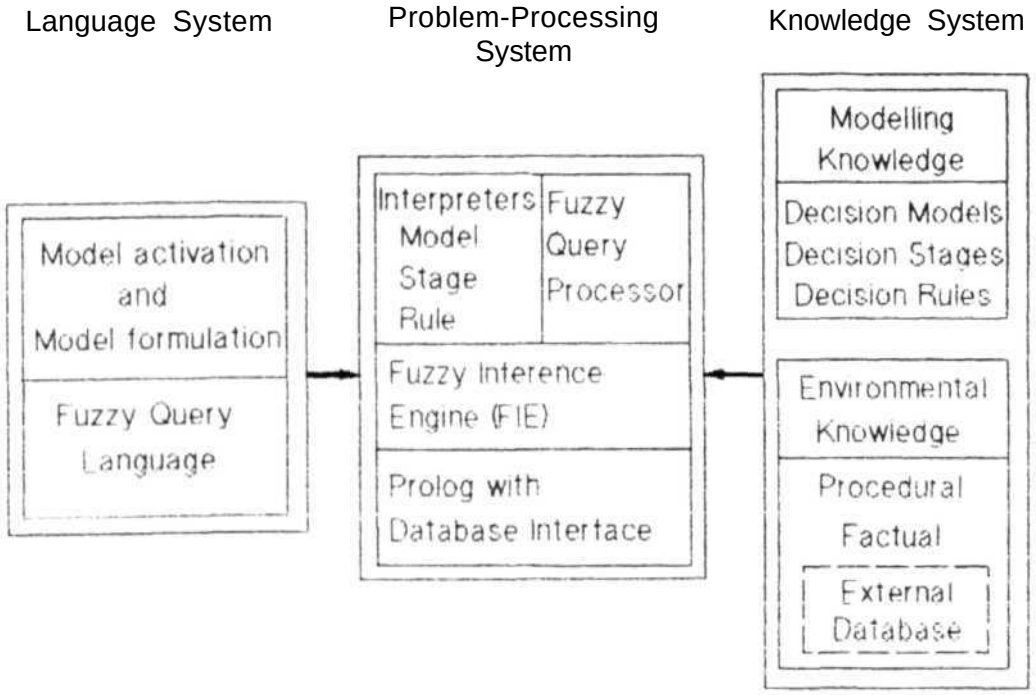


Fig. 2.1 Architecture of Proposed DSS Environment

environmental knowledge, though relevant to modelling knowledge, does not consist of any problem specific knowledge. It merely comprises of the knowledge specific to the universe of discourse. The properties of both these knowledge components is available in

the form of meta knowledge that includes information such as linguistic constant definitions, fuzzy relational operator definitions, etc.

Modelling Knowledge:

Modelling knowledge is structured as a hierarchy of three levels namely, decision models, decision stages and decision rules. Decision models, at the top of the hierarchy, represent the decomposed decision problems and specification of the composition of the decision problems using decision stages. Decision stages represent the smallest decision problems, and consist of a specification of probable alternatives and the decision rules to be employed in constraining and ranking the alternatives. The decision rules attached to the specification of decision stages are designated as:

a) precondition,
b) action,
c) postcondition, and
d) ranking rules.

A precondition rule constrains the set of probable alternatives before the updates to the knowledge base are affected by action rule. This set of constrained alternatives are further pruned by postcondition rule to yield the set of feasible alternatives. A rank value is associated with each alternative in this set as defined by rank rule.

Decision rules, at the bottom of the hierarchy, represent the basic

modelling knowledge (available in the form of rules, guidelines, etc.) as high level declarative fuzzy rules. It is possible to chain the decision models and the decision rules recursively to define more complex decision models and decision rules. Imprecision and uncertainty in decision rules and decision parameters is represented using concepts of fuzzy sets, possibility distributions, and necessity-possibility measures.

Environmental Knowledge:

Environmental knowledge is the basis of one or more DSS applications and comprises of procedural and factual knowledge. It is possible that part of the factual knowledge is available in the form of an existing database. Procedural knowledge of this component can be specified using Prolog-like rules with extensions to represent imprecision and uncertainty using fuzzy sets, possibility distributions, and necessity- possibility measures.

2.3.2 Language System

The user interface of the DSS environment is defined by the language system. We can identify two types users of this environment as model builders and decision makers. Model builders are responsible for creating majority of the modelling and environmental knowledge components. Decision makers make use of this knowledge to generate alternatives and analyse the consequences. Decision makers may alter the modelling knowledge component provided by the model builders. It is, however, possible that the role of model builder

is assumed by a decision maker.

The language system provides a non-procedural command oriented language interface for model execution and information retrieval. The decision models can be activated for execution either by a mere specification of the name of the decision model or by a composition or formulation of new models. Again, the decision parameters, to either of these specifications, can be imprecise. A fuzzy query language provides the facilities to retrieve information from the database and the knowledge base. The language constructs do not distinguish between the procedural or factual knowledge available in the environmental knowledge base and their specification is identical to that of the decision rules.

Model Activation and Formulation:

Model execution is one of the two important features of any DSS. Model execution refers to a) direct activation of a predefined decision model along with its parameters and b) formulation of a decision model, interactively, using predefined decision stages. At this point the user can exercise any of the three following options:

a) commit an alternative for that stage and system proceeds to the next defined stage

b) request for another alternative for that stage

c) skip the current stage in order to select a different alternative at a previous stage

A complete solution to the decision problem is found when all the decision stages involved have some alternative committed.

Fuzzy Query Language:

Exploratory information retrieval from the knowledge base can be performed using fuzzy query language. This is a non-procedural high level query language with extensions to access imprecise and/or uncertain knowledge. Further, the inexact conditions, such as *approximately equal,* for information retrieval can be specified. In addition, the database access is made transparent, i.e., the user need not distinguish between knowledge base access and database access. As a matter of fact, a single query can refer to both the knowledge base and the database.

In the condition part of the queries aggregate functions, existential and universal quantifiers, and fuzzy quantifiers can be specified. These functions and quantifiers can also be nested to specify more complex conditions.

The procedural knowledge which is in the form of rules can also be retrieved using the constructs available in this language interface which amounts to querying on derived knowledge. Furthermore, updates to the knowledge base can be made using the three update predicates provided in the query language. However, these updates will only be valid for the current session.

2.3.3 Problem-Processing System

The **PPS** component is realised using a layered interpreter built on top of a fuzzy inference engine. It is possible to define heuristic knowledge to control the generation of alternatives. Further, it is possible to define the functions to be employed in the propagation of uncertainty during the inference. In addition, this system provides an interface to external databases for transparent access of data to be used by the modelling knowledge component.

Model, Stage, Rule Interpreters:

Decision rule interpreter, built on top of the fuzzy inference engine, provides the higher level constructs such as quantification, aggregation, etc. The decision stage interpreter resides above the rule interpreter and performs functions such as generation of alternatives using user-defined heuristics, if any, validation of the alternatives against the pre- and post-conditions, etc. The top level interpreter offers the direct user interface by interpreting the decision models. Further, this layer provides the control to the user at each of the decision stages of the model by presenting the currently available alternatives along with ranking values.

Fuzzy Query Processor:

Fuzzy query processor is built using the decision rule interpreter and it is at the same level as that of decision stage interpreter. In addition to basic non-procedural retrieval commands with fuzzy

extensions, this processor is also capable of interpreting and executing the update requests. The updates made through this are only temporary and are discarded at the end of the session. Further, features such as fuzzy arithmetic can be effectively used to perform computations on the retrieved imprecise/precise values.

Fuzzy Inference Engine:

The problem-processing system is built on top of a Prolog interpreter in a layered architecture. Immediately residing above the Prolog is a fuzzy inference engine which facilitates the necessary extensions to Prolog such as semantic unification (or fuzzy pattern matching), fuzzy comparison, inference using uncertain rules, etc.

2.4 A Prototype Implementation

A prototype of the proposed DSS environment, named Plan Aid, is implemented in C-Prolog on VAX-11/750. A C-Prolog interpreter has been extended to provide transparent access (Bolloju and Kamath 1989) to databases of Admin (Naveen *et al* 1983). At the core of PlanAid is a fuzzy inference engine that is capable of performing inference under imprecision and uncertainty. The problem-processing system is built around this core using layers of interpreters corresponding to the decision models, decision stages and decision rules. The implementation provides the complete set of facilities required for assistance in decision making activity.

2.5 An Example

In this section we shall present an example of project selection and assignment to illustrate the proposed decision modelling formalism. This decision problem involves (see Fig. 2.2), as the first part, a decision regarding a selection of some projects, from a given set of proposed projects, meeting a set of imprecise and uncertain rules. Subsequently, in the second part an assignment of project managers is to be made, for the selected projects. The latter part of the problem also could contain imprecise and/or uncertain rules in the assignment of project managers.

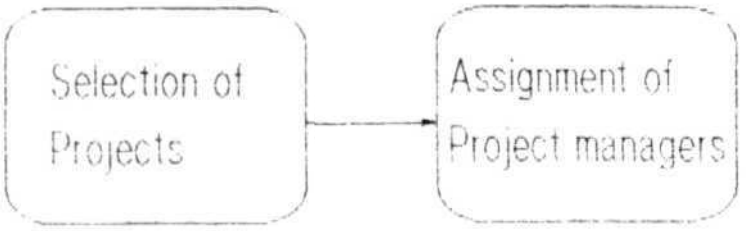


Fig. 2.2 An example decision model

This decision problem can be thought of as a decomposed decision model consisting of two decision models corresponding to the two parts of the problem. We shall map the first decision model to a decision stage and the latter shall be constructed as an iteration of a decision stage which assigns project managers to individual projects.

Decision parameters to the decision models such as maximum available

budget is *about 500000* or average duration of selected projects is *around 15* months, can be specified. Such parameters, in turn, could be used as parameters to the sub-problems as required. Further, these parameters could be either fuzzy or vague (as shown above) or crisp or exact due to

a) the parameters being external to the system, or
b) the decision maker's inability to exactly quantify the parameters.

We shall now elaborate the decision models for selection of projects and assignment of a project leader to a given project. The latter can be used to assign the project managers to all the selected projects in the first part of the problem.

An alternative for the first decision model or stage is a subset of proposed or available projects satisfying a given set of rules such as a) the total cost of the projects does not exceed available budget, b) the selected projects constitute a good mix of projects, etc. Any such subset of proposed projects satisfying the rules is an alternative of the decision stage that can be considered.

The alternatives for the second decision stage are of a different type. Here, an assignment (single) of a project manager to a given project is an alternative. For such a decision stage a single object has to be selected from a given set of objects. That is, a manager (single object) is an alternative from the list of probable managers (set of objects) It can be noticed that the type of alternatives for the two decision stages under consideration are

conceptually different. In the first one the type of alternatives is a **s**ubset of a given set of objects and whereas in the latter it is an element of a given set of objects.

The two decision stages of the example are depicted below:

1: decision stage

select projects:

select a set of projects from proposed projects

satisfying

a.l) budgetary constraints and

a.2) a good project mix and

a.3) caters to the market.

2: decision stage

assign a project manager:

assign a project manager from probable managers

satisfying

a) he/she can handle the project,

b) evaluate the load on the project manager

c) he/she is not overloaded and

d) rank assignments based on possibility of success

The decision stages, as shown above, consist of references to rules apart from the specification of type of alternatives. These decision rules represent the basic modelling knowledge that is used in the specification of a given decision stage. The rules may be classified as *precondition, action, postcondition* and *ranking* rules. In the above examples, identifiers *a,b,c* and *d* are used to denote the above four classes respectively. It is possible to define the decision rules as shown below for the first decision stage:

if the total cost is *about* maximum available budget
and
there are not more than 2 *expensive* projects in the selected set of projects
then that set of projects meets the cost constraint.

if the set of selected projects consists of at least
one turn-key, one developmental, one research project
then that set of selected projects is a good mix.

if the set of selected projects consists of at least
one turn-key, one research project
then that set of selected projects is *possibly* a good mix.

In fact, decision rules can be chained to define more complex rules as shown below for the second decision stage:

if the probable manager has past experience in the area of the project
then he/she can *definitely* handle the project.

if the probable manager is qualified and
has requisite aptitude
then he/she can *possibly* handle the project.

if the probable manager has past experience in other areas
then he/she can *quite possibly* handle the project.

if the probable manager has handled successfully a similar project
then the likely success is *excellent.*

if the probable manager is handling a similar project
then the likely success is *good.*

One can observe from the above set of rules the following characteristics:

a) the rules are in the form of production rules
b) the rules refer to the factual knowledge (e.g., past record of handling projects) or procedural knowledge (e.g., handling similar projects)
c) a good number of the rules specified are inexact or imprecise (e.g., *appropriate aptitude, good experience)*
d) the consequent of the rules is not always certain (e.g., *certainly, possible, likely)*

The environmental knowledge upon which this model can operate could be consisting of:

a) factual knowledge such as proposed projects
b) procedural knowledge such as probable managers
c) factual knowledge as part of a network database consisting of information regarding past history, current status of projects being handled by the probable project managers
d) derived knowledge (view on database) such as probable project managers as employees who have handles at least one project or who are currently handling at least one project.

In addition, the environmental knowledge base could comprise of definition of linguistic terms such as good, appropriate, etc.

2.6 Summary

An overview of the proposed DSS environment is presented in this chapter with a brief description of various components in the architecture. The example presented above are be modelled using the proposed decision modelling formalism in the next chapter. In chapter 6 we shall discuss the implementation details of the prototype. A complete session with a prototype implementation, Plan Aid, is presented in the appendix B.

Chapter 3
Knowledge System

3.1 Introduction

In chapter 1 we have described the knowledge system as consisting of modelling and environmental knowledge components. The knowledge specific to decision problems is termed as the modelling knowledge and this knowledge refers to the environmental knowledge for deriving necessary information. The environmental knowledge is composed of procedural and factual knowledge. It is possible to build a number of specific DSS applications on a given environmental knowledge. Parts of the factual knowledge could be available in the form of an external database. In this chapter, we shall present the formal definitions of these two components of the knowledge system. We shall also propose a syntax to realise this formalism and illustrate it with examples.

3.2 Modelling Knowledge

Decision modelling, as observed earlier, is the process of capturing a real world decision problem. The knowledge of the decision maker in solving the decision problem can be termed as modelling knowledge. We believe that this knowledge can be expressed at different levels of abstraction (see Fig. 3.1). At the top level we can think of decision models as consisting of decision models. These

models could in turn be, further, defined using other, more specific, models or decision stages. Each of the decision stages can be visualised as a selection of feasible alternatives from a specified set of probable alternatives. These decision stages are defined using a set of constraints/rules, possibly imprecise and/or uncertain in order to specify the mapping from the set of probable alternatives to the set of feasible alternatives. The various levels in the modelling knowledge and the interrelationships are shown in Fig 3.1.

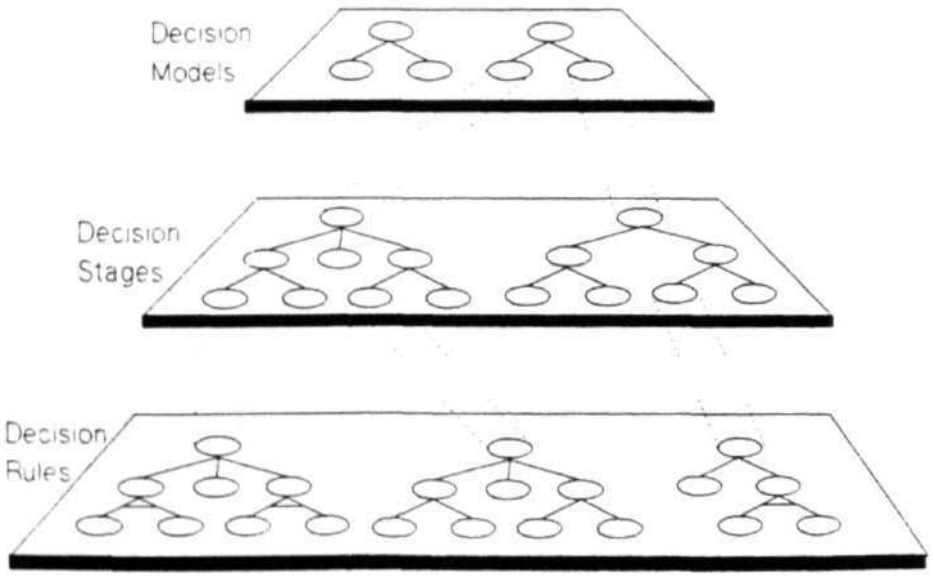


Fig. 3.1 Levels in Modelling Knowledge

In this section we shall, first, present the decision modelling formalism (Bolloju 1991a) proposed in this thesis. Thereafter, we shall discuss in detail how this formalism can be realised using the proposed specification language.

3.2.1 The Definitions

Definition 3.1: *Modelling knowledge* is a collection of decision models, decision stages and decision rules, and it is defined by the following 3-tuple:

$<M,\Sigma,P>$

where

M is a set of decision models, $\{\mu_1,\mu_2,\ldots,\mu_m\}$,

Σ is a set of decision stages, $\{\sigma_1,\sigma_2,\ldots,\sigma_s\}$, and

P is a set of decision rules, $\{\rho_1,\rho_2,\ldots,\rho_r\}$.

Definition 3.2: A *Decision model,* μ, is defined as a 2-tuple

$<\mu_h,\mu_b>$

where

μ_h is a model head which consists of an identifier and a list of parameters to the model, and

μ_b is a model body which is composed of decision models and decision stages as defined below.

Definition 3.3: A decision stage head is a *decision model body.* If μ_{b1} and μ_{b2} are decision model bodies then the following are also decision model bodies.

μ_{b1} , μ_{b2}

μ_{b1} ; μ_{b2}

ψ ? μ_{b2}

ψ ? μ_{b1} : μ_{b2}

$\forall x \in \chi \quad \mu_{b1}$

[ψ : μ_{b1}]

(μ_{b1})

The composition operator Y **defines** a sequence of decision models, i.e., decision model μ_{b1} precedes μ_{b2}. On the other hand, the operator ';' defines two models that are independent of each other and they can be solved in any order. The next composition defines a conditional model invocation indicating that if the predicate ψ is evaluated to true then μ_{b1} is to be invoked. The next composition defines invocation of μ_{b1} if the predicate ψ is evaluated to true; otherwise the decision model μ_{b2} is to be invoked. The next composition operator defines an iteration of invocation of model μ_{b1} for all x belonging to % (see Defn. 3.8). The composition [ψ : μ_{b1}], also, defines an iteration of μ_{b1} while the predicate ψ is evaluated to true. Lastly, a decision model body can be parenthesised to appropriately group the part of the composition.

Definition 3.4: A *Decision stage, a* , is defined as a **6-tuple**

$$< \sigma_h, \omega, \psi_1, \phi, \psi_2, \kappa >$$

where

σ_h is *stage head* which consists of an identifier and a list of parameters to the decision stage,

ω is a predicate expression (Defn. 3.5) identifying a *set of probable alternatives* of the decision stage

ψ_1 is a *pre-condition* predicate,

ϕ is an *action* predicate,

ψ_2 is a *post-condition* predicate, and

κ is a *ranking* predicate or term.

A decision stage σ maps a set of probable alternatives defined by ω to a set of feasible alternatives using the predicate ψ as a precondition and the predicate ψ as a postcondition after the updates to the knowledge base defined by ϕ are performed. All these predicates are evaluated with respect to the alternatives defined by ω.

The *pre-condition* and *post-condition* predicates are defined according to the definition of condition rule body (Defh. 3.7). Similarly, the *action* predicate is defined as action rule body (Defn. 3.9). Lastly, the *ranking* predicate or term is the rank rule body as defined in Defn. 3.10.

Definition 3.5: A predicate expression ω, in its simplest form, is a predicate symbol or name belonging to environmental knowledge. If ω_1 and ω are predicate expressions then the following are also predicate expressions:

ω_1^*

$\omega_1 \times \omega_2$

A predicate expression ω defines the set of probable alternatives identified by a given decision stage. The probable alternatives are the set of n-ary tuples (derived or ground facts) defined by the predicate symbol of the procedural or factual knowledge component. The expression co defines the set of probable alternatives as a power set, i.e., all subsets of n-ary tuples defined by ω. The expression co x co defines the cartesian product of the sets of **m-ary** and n-ary tuples identified by the expressions co and co .

Definition 3.6: A *Decision rule, p* , is defined as a **3-tuple**

$$<\rho_h, \rho_b, \rho_c>$$

where

ρ_h is the decision rule head or the consequent which consists of an identifier and a set of arguments,

ρ_b is the decision rule body or the antecedent which can be either a condition rule body, or an action rule body or a rank rule body,

ρ_c is the certainty of the decision rule defined as a pair of values $<n,n>$

where

$\pi = \Pi(\rho_h|\rho_b)$ is the possibility that the consequent is true whenever the antecedent is found true,

$n = N(\rho_h|\rho_b)$ or

$1 - \Pi(\neg\rho_h|\rho_b)$ is the impossibility that the opposite of the consequent is

true whenever the antecedent is found true.

Decision rules arc classified into condition, action or ranking categories according to the type of the rule **body**. Decision rules of a given category can be chained to define more complex decision rules. The certainty measure can only be associated with the decision rules of condition category.

Definition 3.7: A *Condition rule* ***body***, ψ, is defined as:

$p_1(a_1, a_2, \ldots, a_n)$ or

$p_2(\tau_1, \tau_2)$ or

$p_3(x_1, x_2, \ldots, x_m)$

where

p_1	is a decision rule identifier,
$a_{i,i \in [1,n]}$	are the arguments of the decision rule identified by p ,
p_2	is a relational operator, fuzzy or crisp,
τ_1 and τ_2	are terms which are operands of p_2,
p_3	is a predicate name belonging to environmental knowledge identifying a relationship, and
$x_{i,i \in [1,m]}$	are the variables in relationship P_3

Further, if ψ and ψ are condition rule bodies then the following are also condition rule bodies:

$\psi_1 \wedge \psi_2$

$\psi_1 \vee \psi_2$

$\neg \psi_1$

(ψ_1)

$\forall x \in \chi \ \psi_1$

$\exists x \in \chi \ \psi_1$

$\mathcal{Q} x \in \chi \ \psi_1$

$\mathcal{Q}_m \ \mathcal{Q} x \in \chi \ \psi_1$

where

$\wedge, \vee, \neg$ **are** conjunction, disjunction and negation operators respectively,

$\forall, \exists$ are universal and existential quantifiers,

$\mathcal{Q}$ stands for a fuzzy quantifier such as *most, few,* etc. and

$\mathcal{Q}_m$ stands for a modifier to the fuzzy quantifier $\mathcal{Q}$.

The variable x associated with various forms of quantified rule bodies is defined over the n-ary tuples identified by the range expression χ. **For** example, V x G χ ψ. defines universally quantified rule body with x as a quantified variable ranging over the n-ary tuples identified by χ.

Definition 3.8: A *range* *expression* χ is defined as:

$\mathbf{p}(x) \wedge \mathbf{p}_1(y_1) \wedge \mathbf{p}_2(y_2) \wedge \ldots \wedge \mathbf{p}_n(y_n) \wedge \psi$

where

P	is a predicate symbol or name belonging to environmental knowledge identifying a component of factual or procedural knowledge with x as the range **variable,**
$\mathbf{p}_{i,i\in[1,n]}$	are predicate symbols of environmental knowledge with variables $\mathbf{y}_{i,i\in[1,n]}$ ranging over $\mathbf{p}_i$, and
ψ	is a condition rule body with x as a bound variable in ψ.

A range expression as described above defines a set of n-ary tuples of predicate p satisfying the condition rule ψ. In the proposed syntax of range expression, we shall use the following equivalence:

$$\mathbf{p}_1(y_1) \wedge \mathbf{p}_2(y_2) \wedge \ldots \wedge \mathbf{p}_n(y_n) \wedge \mathbf{p}_r(y_1,y_2,\ldots,y_n) \equiv \mathbf{p}_r(y_1,y_2,\ldots,y_n)$$

That is, the variables in relation p implicitly define the respective ranges. This equivalence, as we shall see in the example 4.4 in chapter 4, makes the specification simpler.

Definition 3.9: An *Action rule body*, ϕ, is defined as consisting of either assert $\mathcal{I}$ or retract $\mathcal{D}$ or replace $\mathcal{R}$ action predicate as shown below:

$$\mathbf{I}(\mathbf{p}(\tau_1,\tau_2,\ldots,\tau_n))$$

$$\mathbf{D}(\mathbf{x})$$

$$\mathbf{R}(\mathbf{x}/\mathbf{y},(a_1/\tau_1,a_2/\tau_2,\ldots,a_n/\tau_n))$$

Further, if ϕ_1 ϕ_2 are action rule bodies then the following

are also action rule bodies.

ϕ_1 , ϕ_2

$[\psi : \phi_1]$

$\forall x \in \chi \ \phi_1$

$\psi \ ? \ \phi_1 : \phi_2$

$\psi \ ? \ \phi_1$

The assert action predicate (I) defines the assertion of fact p with values of the terms $T_1, T_2, \ldots, T_D$ as the arguments. Then, the retract action predicate (D) defines the retraction of the fact identified by the variable x. Finally, the replace action predicate (R) is equivalent to the combination of retraction of the fact identified by the variable x and assertion of a new fact identified by the variable y with values of the terms $T_1, T_2, \ldots, T_D$ as the argument values for $a_1, a_2, \ldots, a_D$ respectively.

The composition operator Y defines the actions to be performed in sequence. The next operator *[y/ :* ϕ_1*]* defines the iteration of ϕ while the condition ψ is true (strictly). The V x $\in$ % ϕ define* iteration of ϕ with the variable x ranging over χ. The composition ψ ? ϕ *:* ϕ defines the action ϕ if the condition ψ is true and ϕ otherwise. The last composition is similar to the above with null action for the case of condition ψ is false.

Definition 3.10: Any term r or any condition rule body ψ is a *rank rule body.*

Alternatives axe ranked based on the absolute value of the term T if

a **term** is used as a rank rule body. On the other hand, condition rule body evaluates the rank value based on the degree of certainty of the inference (see Defn. **5.4)**.

Definition 3.11: A *term x* can be any one of the following:

c

$a_i(x)$

$f(\tau_1, \tau_2, \ldots, \tau_n)$

$a_f(\tau, \chi)$

$a_f(x, \chi)$

where

C is a constant (Defn. 3.16),

$a_i(x)$ is the > value of the **n-tuple** identified by the variable x,

f is a function with x ,x ,...,T as the arguments,

a_f is an aggregate function on τ or x over the range expression χ.

Constant values can be either fuzzy sets or possibility distributions indicating imprecise and/or uncertain values or precise and certain values. The argument values represented by a.(x) also have the same structural definition. The symbol T represents functions such as '+', '-', etc. on the terms τ ,τ ,...,T . Lastly, **aggregate** functions such as sum, max, etc. of term T or count of variable x defined over the set of tuples identified by % are represented by the last two categories of the definition of term.

3.2.2 Syntax and Examples

In this section we shall describe how the formalism presented in the previous section is realised in the proposed syntax.

Syntax of decision models:

DecisionModel = *DecisionModelHead* <- *DecisionModelBody* .

DecisionModelHead = *DecisionModelName* [(*Arguments*)] .

Arguments = *Argument* [, *Arguments*] .

DecisionModelBody =
DecisionModelComp1 [-> *DecisionModelBody*] .

DecisionModelComp1 =
DecisionModelComp2 [|| *DecisionModelComp1*] .

DecisionModelComp2 =
(if *CondRuleBody*
then *DecisionModelBody* [else *DecisionModelBody*])
| (foreach *Variable* of *RangeExpression*
do *DecisionModelBody*)
| (while *CondRuleBody* do *DecisionModelBody*)
| *DecisionStageHead*
| ((*DecisionModelBody*)) .

Example 3.1:

```
sel_proj_and_assign_leader(Budget) <-
        select_a_project(PJ,Budget) ->
        assign_project_leader(L,PJ).
```

The above example illustrates a decision model *sel_proj_and_assign_leader* as composed of the decision stages *select_a_project* and *assign_project_leader* to be solved in sequence. The first decision stage selects a project among the set of proposed projects and the second decision stage assigns an employee from probable managers to the selected project. In other words, the

above decision model generates project and project leader pairs according to the respective decision stage specifications.

The decision model arguments can be either variables or constants. The variables in argument list can be classified as either input or output variables. In the above example *Budget* is an input variable, and the variable *PJ* acts as an output variable to the decision model ***select_a_project*** and as an input variable to the decision model ***assign_pr oject_leader***. This feature can be used to pass any information across decision models and decision stages.

The constant argument values are generally used for the specification of multiple decision models with the same identifier to a given decision problem. Example 3.2 illustrates the usage of constant value as the first argument to indicate the applicability of a decision model to a given situation, i.e., selection of a decision model based on the type of a project.

Example 3.2:

```
sel_proj_and_assign_leader(research,Budget) <-  ...
sel_proj_and_assign_leader(developmental,Budget) <-  ...
sel_proj_and_assign_leader(turnkey,Budget) <-  ...
```

The decision model presented below illustrates a different model composition comprising . a decision stage ***select _projects*** and an iteration of the decision stage ***assign_project_leader*** to assign a manager to each of the projects selected.

Example 3.3:

```
sel_projs_and_assign_mgrs(Max_budget) <-
        select_projects(P,Max_budget) ->
        foreach PJ of P
          do assign_project_leader(L,PJ).
```

Syntax of Decision Stages:

The syntax of decision stages is presented below:

DecisionStage = *DecisionStageHead* <- *DecisionStageBody* .

DecisionStageHead = *DecisionStageName* [(*Arguments*)] .

DecisionStageBody = select *AltVariable* of *AltSpecification*
[using *StageSpec1*] .

AltVariable = *Variable* | subset *Variable* .

AltSpecification = *PredicateName* .

StageSpec1 = (precond *CondRuleBody* [, *StageSpec2*])
| *StageSpec2* .

StageSpec2 = (action *ActionRuleBody* [, *StageSpec3*])
| *StageSpec3* .

StageSpec3 = (postcond *CondRuleBody* [, *StageSpec4*])
| *StageSpec4* .

StageSpec4 = [rank *RankRuleBody*] .

Example 3.4:

```
select_a_project(P,Budget) <-
        select P of proposed_project
            using
                precond ((promising(P) or
                          expertise_available(P)) and
                          (P.cost =< Budget)),
                rank    100/P.duration+2*P.cost.
```

```
select_projects(Max_budget) <-
        select subset P of proposed_project
            using
                precond (meets_cost_constr(P,Max_budget) and
                         good_mix_of_proj(P) and
                         market_driven(P)),
                rank    sum(PJ.cost:in(PJ,P)).

assign_project_leader(L,P) <-
        select L of probable_manager
            using
                precond   can_handle(L,P.area,P.type),
                action    add_project_load(L,P.type,NewL),
                postcond  not over_loaded(NewL),
                rank      poss_success(L,P).
```

The first decision stage above specifies that the alternatives are elements from the set of **proposed** projects satisfying

a) the rule ***promising_project*** or

b) the rule *expertise* ***_available*** and the project cost is within the budget.

Subsequently, the selected projects should be ranked using the expression specified as a rank term.

The decision stage, ***select_projects***, selects a subset of projects from the proposed projects using the rules associated with the precondition. That is, each subset of projects selected should

a) meet the cost constraint, and

b) have a good mix of projects, and

c) consist of market driven projects.

The last decision stage, ***assign_pr oject_leader***, assigns a probable manager (L) for the project (P) such that L ***can_handle*** the project P and L is *not overloaded* after the computation of the additional load

on L by **the** action rule ***add_project_load***. The assignment will, then, be **ranked** by the possibility of success of L leading P.

It is possible to define more than one decision stage with the same **identifier** to cater to different cases of specification. That is, the precondition, action, postcondition and rank rules associated with the decision stage specification can vary across the different cases. The example 3.5 below illustrates this type of specification with separate decision stages for projects with the available budget specified as low, medium and high budget values.

Example 3.5:

select_projects(~ low_budget,Max_budget) <- ...

select_projects(~ medium_budget,Max_budget) <- ...

select_projects(~ high_budget,Max_budget) <- ...

Syntax of Decision Rules:

The modelling knowledge at the lowest level of the modelling knowledge hierarchy proposed in this thesis is represented by decision **rules.Decision** rules can be chained to define more complex rules. The syntax of decision rules is presented below:

DecisionRule = ***DecisionRuleHead*** <- *DecisionRuleBody* .

DecisionRuleHead — ***DecisionRuleName*** [(*Arguments*)] .

DecisionRuleBody = (*CondRuUBody* [@ ***NecPosMeasure***])
| *ActionRuleBody*
| ***RankRuleBody*** .

NecPosMeasure = [***Necessity*** *Degree* , *PossibilityDegree*] .

CondRuUBody = *CondRuleDisj* [**and** *CondRuUBody*) .

```
CondRuleDisj = CondRulePred [ or CondRuleDisj ] .

CondRulePred = PredicateName(Arguments)
             | ( Expression RelationalOperator Expression )
             | ( Quantifier(Variable:RangeExpression,
                            CondRuleBody) )
             | ( CondRuleBody )
             | ( not ( CondRuleBody ) ) .

PredicateName = FactName
              | RuleName
              | ObjectName
              | RelationshipName
              | in .

RelationalOperator = PredefinedRelOpr
                   | UserDefinedRelOpr .

PredefinedRelOpr = = | \= | > | >= | < | =< .

Quantifier = exists | all
           | ( [Modifier] FuzzyQuantifier ) .

Expression = Term [( + | - ) Expression ] .

Term = Factor [ ( * | / ) Term ] .

Factor = Constant
       | Variable.AttrName
       | FunctionName(Arguments)
       | AggregateFunctionName(Variable:RangeExpression)
       | AggregateFunctionName(Term:RangeExpression)
       | ( Expression ) .

RangeExpression = PredicateName(Variable)
                  [ and ( CondRuleBody | RangeExpression ) ]

ActionRuleBody = ActionRuleComp [ and ActionRuleBody ] .

ActionRuleComp = ActionRuleSimp
               | ( foreach Variable of RangeExpression
                   do ActionRuleComp )
               | ( if CondRuleBody
                   then ActionRuleComp
                   [ else ActionRuleComp ] ) .

ActionRuleSimp =
    insert(GroundPredName(AttrValueList))
  | delete(Variable)
  | replace(Variable1 by Variable2, (AttrValueAssignList) .
```

GroundPredName = *FactName*

| *ObjectName* .

AttrValueList = *AttrName* : *Term* [, *AttrValueList*] .

AttrValueAssignList = *AttrName* := *Term* [, *AttrValueAssignList*]

RankRuleBody = *CondRuleBody* \ *Term* .

Example 3.6:

```
% 1

% set of projects, P, meets the cost constraint if the total
% cost of the projects is below the available budget and
% there are not more than two expensive projects in the set P
meets_cost_constr(P,Max_budget) <-
        (sum(PJ.cost:in(PJ,P)) =< Max_budget) and
        (count(PJ:(in(PJ,P) and PJ.cost = ~expensive)) =< 2).

% 2

% the probable manager P can certainly handle a project
% of a given area and type if he/she has past experience
% in that area
can_handle(P,Area,Type) <-
        has_past_experience(P,Area).

% the probable manager P can possibly handle a project
% if he/she has lead other projects
can_handle(P,Area,Type) <-
        has_lead_other_projects(P) @ [0.9,1].

% the probable manager P can more or less handle a project
% of a given type if he/she has the requisite aptitude and
% is qualified for the project type
can_handle(P,Area,Type) <-
        has_requisite_aptitude(P) and
        qualified(P,Type) @ [0.8,1].

% rules defining the aptitude in terms of leadership aptitude
has_requisite_aptitude(P) <-
        P. leader ship_apt = ~ excellent.

has_requisite_aptitudc(P) <-
        P.leadership_apt = ~above_average @ [0.8,1].

% rules defining qualified in terms of qualifications and
% age
qualified(P,research) <- P.qual = phd.
qualified(P,research) <- P.qual = mtech @ [0.7,1].
qualified(P,developmental) <- P.qual = mtech.
```

```
qualified(P,developmental) <-
        P.qual = btech and P.age > 30 @ [0.7,1].
qualified(P,turnkey) <- P.age > 32.

% if P has lead a project of a given area in the past
% then P certainly has the past experience
has_past_experience(P,Area) <-
        exists(PJ:project(PJ),(PJ.area = Area          and
                               team_of(PJ,R)           and
                               R.role = leader         and
                               participated_as(E,R) and
                               E.no = P.no)).

% if P has participated in a project of a given area
% in the past as a deputy leader then P possibly has the
% past experience
has_past_experience(P,Area) <-
        exists(PJ:project(PJ),(PJ.area = Area          and
                               team_of(PJ,R)           and
                               R.role = deputy_leader and
                               participated_as(E,R)    and
                               E.no = P.no)) @ [0.8,1].

% P has lead at least one project in the past
has_lead_other_projects(P) <-
        exists(R:role(R), (R.role = leader        and
                           participated_as(E,R) and
                           E.no = P.no)).

% 3

% update the total load on M using the following action rules
add_project_load(M,research,N) <-
        replace(M by N, total_load := M. total_load + ~ heavy).

add_project_load(M,turnkey,N) <-
        replace(M by N, totaLjoad := M.total_load+ ~ medium).

add_project_load(M ,developmental ,N) <-
        replace(M by N, total_load := M.total_load + ~ below_heavy).

% 4

% rule for estimating the overload on probable manager
over_loaded(N) <-
        if N.experience = ~fairly_senior
           then N.total_load > 120
        else if N.experience = ~senior
           then N.total_load > 140
           else N.total_load > 130.
```

```
% 5

% possibility M leading P successfully is defined as
% the past record of M or likely success of M leading P and
% M having the knowledge of project area.
poss_success(M,P) <-
        past_success(M) or
        (likely_success(M,P.type)     and
         has_knowledge(M,P.area)).

% the past record is good if all the projects in which M
% participated as leader or deputy leader have been
% successfully completed
past_success(M) <-
        all(J:(project(J) and team_of(J,R) and
                (R.role = leader or R.role = deputy_leader) and
                participated_as(E,R) and E.no = M.no),
            J.status - success).

likely_success(M,research) <-
        M.qual = phd and M.experience > = 5.
likely_success(M,research) <-
        M.qual = mtech and M.experience > = 9 @ [0.9,1].
likely_success(M,developmental) <-
        M.qual = mtech and M.experience >= ~senior.
likely_success(M,turnkey) <-
        M. experience = ~ fairly_senior.

has_knowledge(M,Area) <-
        exists(P:(project(P) and P.area = Area),
               (team_of(P,R) and participated_as(E,R) and
                E.no = M.no)).
```

The example above depicts the following rules:

a) *meets_cost_constr*,

b) *can_handle*,

c) *add_project_load*,

d) *over_loaded*, and

c) *poss_success*.

The first one illustrates an elegant use of aggregate functions in production rule specification. The second example defines a complex set of chained rules and the hierarchy of these rules is shown in

Fig. 3.2 as an AND-OR tree. An action rule to update the total load on the project manager is illustrated by the third rule. Then, the fourth rule is used as a postcondition to verify that the project manager is not over loaded. The last example, defines the ranking of alternative assignments of managers to projects.

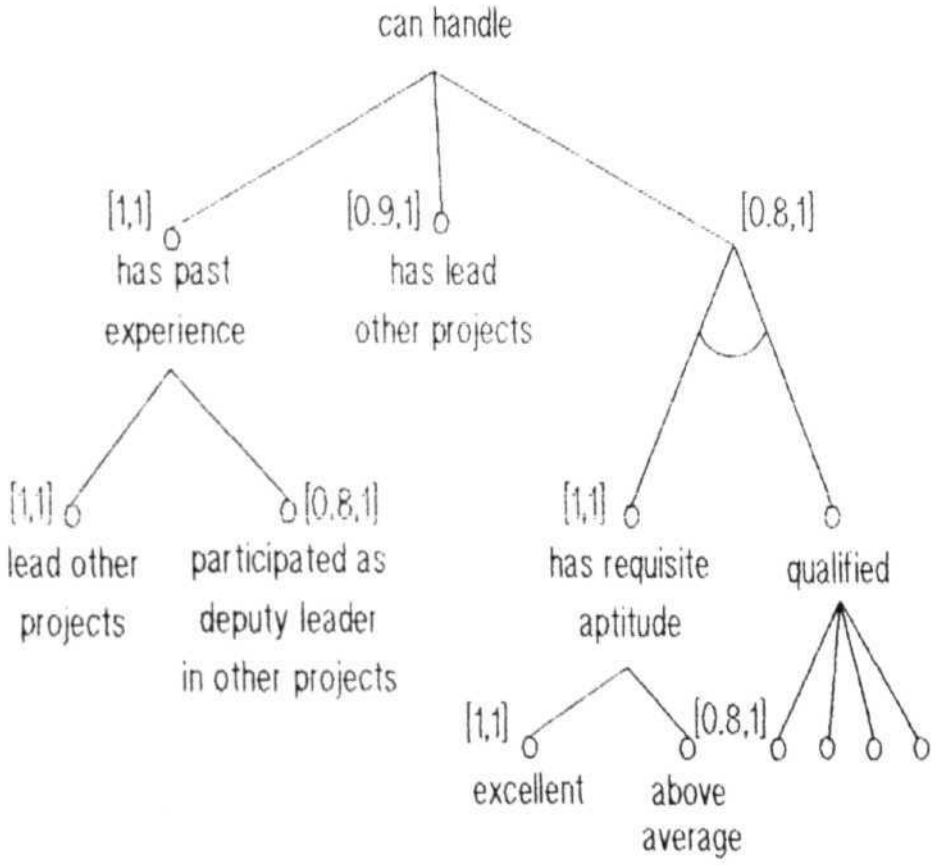


Fig 3.2 Hierarchy of rules associated with rule ***can_handle***

3.3 Environmental Knowledge

As described earlier, environmental knowledge is the basis for one or more DSS applications. This knowledge component can be divided into procedural and factual knowledge components. The procedural knowledge component provides information that is derivable from other procedural and factual knowledge component. It is possible that parts of the factual knowledge could be available in the form of an external database. Fig. 3.3 depicts the structural aspects of environmental knowledge. In this section we shall present the definitions related to the environmental knowledge. Subsequently, we present the proposed syntax for the representation of environmental knowledge along with examples.

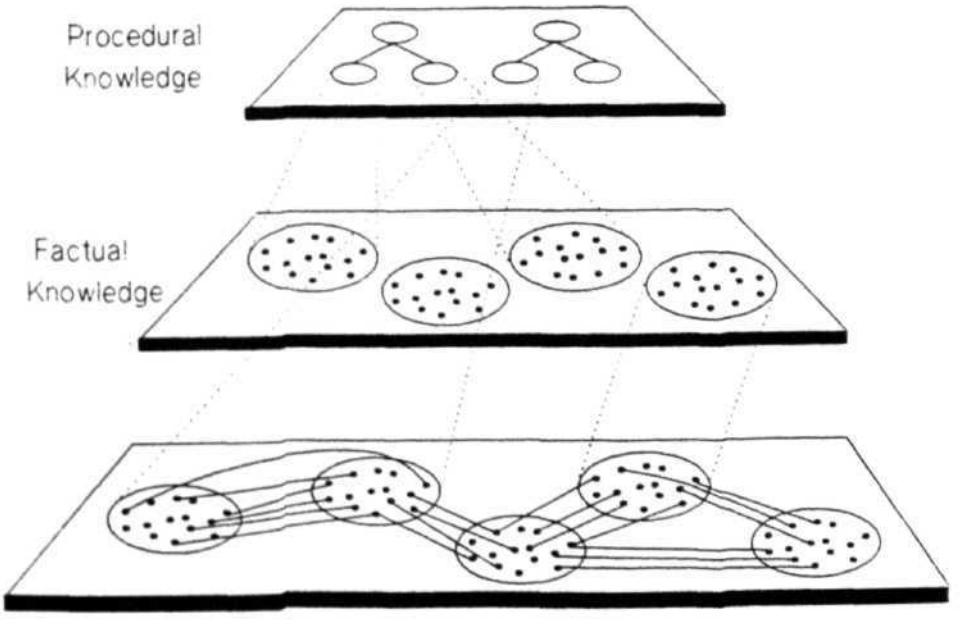


Fig 3.3 Levels in Environmental Knowledge

3.3.1 The Definitions

Definition 3.12: Environmental **knowledge** is a collection of procedural and factual knowledge and it is defined as a 2-tuple:

$<\Lambda,\Gamma>$

where

Λ is a set of rules, $\{\lambda_1,\lambda_2,\ldots,\lambda_r\}$, and

Γ is a set of facts, $\{\gamma_1,\gamma_2,\ldots,\gamma_g\}$.

That is, the rules $\lambda_1,\lambda_2,\ldots,\lambda_r$ constitute the procedural knowledge component and the facts $\gamma_1,\gamma_2,\ldots,\gamma_g$ constitute the factual knowledge.

Definition 3.13: A *rule*, λ , is defined as a 3-tuple

$<\lambda_h, \lambda_b, \lambda_c>$

where

λ_h is the rule head or the consequent which consists of an identifier and a set of arguments,

λ_b is the rule body or the antecedent as defined below,

λ_c is the certainty of the rule defined as a pair of values $<n,\pi>$

where

$\pi = \Pi(\lambda_h|\lambda_b)$ is the possibility that the consequent is true whenever the antecedent is found true,

$n = N(\lambda_h | \lambda_b)$ or

$1 - \Pi(\neg\lambda_h | \lambda_b)$ is the impossibility that the opposite of the **consequent is** true whenever the antecedent is found true.

Although the rules of procedural knowledge component and the decision rules have similar structure, they differ in the definition of their respective rule bodies.

Definition 3.14: A *rule body,* X_b, is defined as:

$\mathbf{p(a_1, a_2, \ldots, a_n)}$

where

P is a predicate symbol or **identifier** of some component of environmental knowledge, or a crisp or fuzzy relational operator, and

$\mathbf{a_1, a_2, \ldots, a_n}$ are arguments which are constants or variables representing the corresponding argument of p (see Defn. 3.16).

Further, if $\boldsymbol{\lambda}$ and X are rule bodies then the following are also rule bodies:

$\lambda_1 \wedge \lambda_2$

$\lambda_1 \vee \lambda_2$

$\neg \lambda_1$

(λ_1)

where $\wedge$, $\vee$, and $\neg$ are conjunction, disjunction and negation

operators. And, the rule bodies can be parenthesised to appropriately group the parts of the specification. The negation is defined by the principle of *negation by failure.* That is, knowledge base assumes the *Closed World Assumption* **(CWA)** and negation of a goal G is defined as success if the rule corresponding to G fails certainly.

It can be noticed from the above definition that the level of specification of these rules is *lower* compared to that of decision rules. The rule body definition does not permit the quantification and use of terms with aggregate functions etc.

Definition 3.15: A *fact* γ is defined as a 2-tuple as:

$$\gamma = <\gamma_t, \gamma_c>$$

where

γ_t is an n-ary tuple of a ground predicate **p** of environmental knowledge and has a structure $\mathbf{p}(a_1, a_2, \ldots, a_n)$, and

γ_c is the certainty of the n-ary tuple defined as a necessity-possibility measure.

Definition 3.16: A *constant value* a_i is defined as a fuzzy subset, C, on a base domain $B = \{\beta_1, \beta_2, \ldots, \beta_\rho\}$ with the membership function:

$$\mu_C : \beta \quad [0,1] \quad \text{such that} \quad \exists \beta \ (\mu_C(\beta) = 1).$$

Apart from the discrete possibility distribution of the fuzzy subset the following special cases may be considered:

i) **fuzzy subset C represents a *precise* constant value, C_p,**

if $\mu_C(C_p) = 1$ **and**

$$\mu_C(\beta) = 0 \ \forall \ \beta \in B \wedge \beta \neq C_p.$$

ii) **fuzzy subset C represents an unknown value**

if $\mu_C(\beta) = 1 \quad \forall \ \beta \in B$

iii) **fuzzy subset C represents a continuous trapezoidal possibility distribution C_f defined by the 4-tuple $<\beta_a,\beta_b,\beta_c,\beta_d>$ as:**

$$\beta_a \leq \beta_b \leq \beta_c \leq \beta_d, \text{and}$$

$$\mu_C(\beta) = 0 \qquad \forall \ \beta, \ \beta < \beta_a$$

$$\mu_C(\beta) = \frac{\beta - \beta_a}{\beta_b - \beta_a} \qquad \forall \ \beta, \ \beta_a \leq \beta < \beta_b$$

$$\mu_C(\beta) = 1 \qquad \forall \ \beta, \ \beta_b \leq \beta \leq \beta_c$$

$$\mu_C(\beta) = \frac{\beta_d - \beta}{\beta_d - \beta_c} \qquad \forall \ \beta, \ \beta_c < \beta \leq \beta_d$$

$$\mu_C(\beta) = 0 \qquad \forall \ \beta, \ \beta > \beta_d$$

iv) **fuzzy subset C represents a crisp possibility distribution C_c defined by the 2-tuple $<\beta_l,\beta_u>$ as:**

$$\beta_l \leq \beta_u, \qquad \text{and}$$

$$\mu_C(\beta) = 0 \qquad \forall \ \beta, \ \beta < \beta_l$$

$$\mu_C(\beta) = 1 \qquad \forall \ \beta, \ \beta_l \leq \beta \leq \beta_u$$

$$\mu_C(\beta) = 0 \qquad \forall \ \beta, \ \beta > \beta_u$$

3.3.2 Syntax **and** Examples

The syntax proposed **in** this thesis to realise the above definitions of environmental knowledge is based on that of Prolog. However, some extensions have been made to accommodate the

necessity-possibility measures and removing the restrictions of the positional arguments. The latter facility results in improved readability in addition to removing the need to for confirming to a prespecified order of arguments.

In order to realise the high level specification of modelling knowledge using the environmental knowledge it is necessary to define the structural description (or templates) of various components of environmental knowledge. The following syntax includes the syntax for these templates also.

Syntax of Environmental Knowledge:

EnvironmentalKnowledge = {*Rule*} {*Fact*} {*MetaDefinitions*}

Rule = *RuleHead* :- *RuleBody* [@ *NecPosMeasure*] . .

RuleHead = *RuleName* [(*Arguments*)] .

RuleBody = *RuleDisj* [, *RuleBody*] .

RuleDisj = *RulePred* [; *RuleDisj*] .

RulePred = *PredicateName*(*AttrIdentValues*)
| *PrologPredicate*(*Arguments*)
| (not (*RuleBody*)) .

AttrIdentValues = [*AttrName* :] (*Constant* | *Variable*)
[, *AttrIdentValues*] .

Fact = *FactName*(*AttrIdentValues*) [@ *NecPosMeasure*] . .

MetaDefinitions = {*FactDef*} {*RuleDef*} {*LinguisticConstDef*} {*FuzzyRelationDef*} {*HeuristicDef*} {*DatabaseDef*} .

FactDef = *FactName* fact_def (*Attrnames*). .

RuleDef = *RuleName* rule_def (*Attrnames*). .

LinguisticConstDef = *LingConstName* lc_def
(*ImpreciseConstant* | *LingConstSpec*) .

LingConstSpec = *LingConstName* [(and | or) *LingConstSpec*] .

FuzzyRelationDef = *FuzzyRelationName* fr_def(*LeftOperand*,
RightOperand,
OperProcedure). .

HeuristicDef = *DecisionStage*
heu_def(heu_var:*Variable*,
heu_val:*Term*,
subset_size:[*Integer1*,*Integer2*],
domain_size:*Integer*). .

DatabaseDef = db_def(schema:*SchemaName*,
subschema:*SubschemaName*). .

Example 3.7:

```
about_4                lc_def   fposs([2,4,4,6]).
two_to_three           lc_def   cposs([2,3]).
around_4               lc_def   fset([2^0.5,3^0.8,4^1,
                                      5^0.8,6^0.5]).
either_3_or_5_or_6     lc_def   cset([3,5,6]).
```

Fig. 3.4 depicts these linguistic constants in a graphical form.

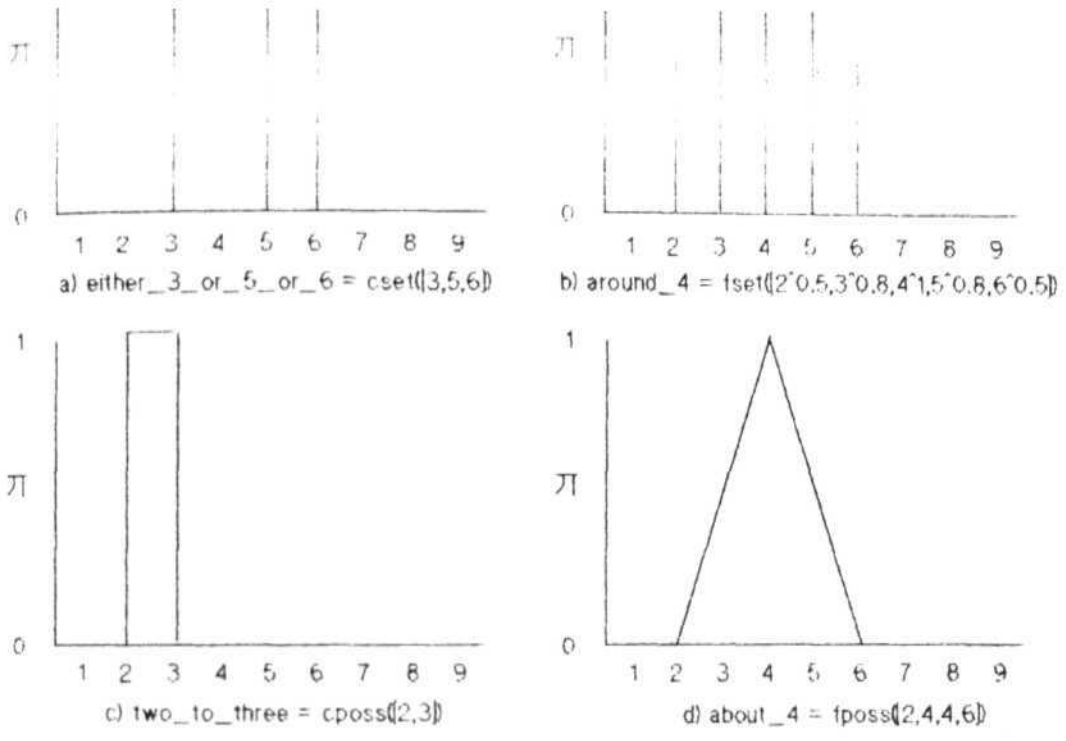


Fig. 3.4 Examples of imprecise values

Example 3.8:

```
prop_proj    fact_def  ( pid,          % project id
                         title,        % project title
                         duration,     % in years
                         manpower,     % number of persons
                         type,         % research/dev/turnkey
                         area,         % broad area
                         complexity).  % on scale [0,100]
```

A schema defining the structure of a fact type is shown above can be used to represent a set of projects such as:

```
prop_proj( p1,dist_dbms,cposs([36,48]), ~about_5, development al,
           database, ~quite_complcx).
prop_proj( p2,dss_generator, ~about_2_years,3,research,
           decision_support, ~moderately_complex).
prop_proj( p3,tp_package,36, ~about_4,developmental,
           operating_systems, ~ complex).
prop_proj( p4,prolog_compiler, ~ above_2_years, ~ about_5,
           research,programming_languages, ~ complex).
prop_proj( p5,gims, 12,15,turnkey,
           games_management, ~not_so_complex).
prop_proj( p6,expsys_shell, ~ about_18_months,3,research,
           artificial_intelligence,~ complex).
```

It is also possible to associate the argument names to the facts as shown below for the project dist_dbms:

```
prop_proj( pid:p1,
           title:dist_dbms,
           manpower: ~about_5,
           duration:cposs([36,48]),
           type: developmental,
           complexity: ~quite_complex,
           area :database).
```

From the above it can be noticed that the order of arguments is immaterial when the argument names are associated with the values.

The example shown below illustrates the use of necessity - possibility measures to indicate the uncertainty in the facts.

```
promising_area(a1 ,four_GL).
promising_area(a2,database) @ [0.9,1].
promising_area(a3,networks).
promising_area(a4,programming_languages) @ [0.8,1].
promising_area(a5,operating_systems) @ [0.7,1].
promising_area(a6,artificial_intelligence).
promising_area(a7,decision_support).
promising_area(a8,games_management) @ [0.6,0.9].
```

Example 3.9:

An example of **rule_def** and three rules for the computation of project cost **are** shown below:

```
proposed_project rule_def  (  pid,title,duration,type,area,
                              complexity,cost).

% rules for computation of project cost in millions of Rs.

proposed_project(Pid,Title,Duration,research,Area,Complexity,Cost)
     :-prop_proj(Pid,Title,Duration,Manpower,
                 research,Area,Complexity),
       Cost is Manpower*Duration*0.025.

proposed_project(Pid,Titlc,Duration,turnkey,Area,Complexity,Cost)
     :- prop_proj(Pid,Title,Duration,Manpower,
                  turnkey, Area,Complexity),
       Cost is Manpower*Duration*0.020.

proposed_project(Pid,Title,Duration,developmental,
                 Area,Complcxity,Cost)
     - prop_proj (Pid ,Title,Duration ,Manpower,
                  developmental, Area,Complexity),
       Cost is Manpower*Duration*0.022.
```

Project cost is computed as a function of manpower required and the duration of the project by the three following rules of environmental knowledge component. These rules correspond to *research, turnkey* and *developmental* project types. It is also possible to use the argument names as defined in rule_def instead of the positional arguments.

The procedural knowledge can be represented by chaining the rules such as the one shown above.

Example 3.10:

A database declaration is shown below:

```
db_def(schema:personnel, subschema :projemp).
```

The following object and relationship definitions which are defined under the subschema **projemp** of the schema **personnel,** will **be**

automatically included as a result of the above definition:

```
employee        obj_def     (no,name,qual,experience,age,leadership_apt).
project         obj_def     (pid,title,area,status).
role            obj_def     (rid,role).

participated_as     rel_def     (employee,role).
team_of             rel_def     (project,role).
```

3.4 Summary

The two components of the knowledge system are defined in this chapter. The modelling knowledge is defined formally as a hierarchy of decision models, decision stages and decision rules. The specification of these components is non-procedural and higher level and a syntax closely related to the definitions is presented with illustrative examples.

Environmental knowledge, the other knowledge system component, is the basis for one or more specific DSS applications. This component comprises of procedural and factual knowledge. After the definition of various aspects of this knowledge component a syntax based on Prolog has been presented for the representation. It is possible that part of the factual knowledge is available in the form of an external database. Therefore, the objects and relationships between the objects are integrated into the definition of factual knowledge.

The semantics of various operations are presented in chapter 5. In the chapter 4 the user interface of the proposed DSS environment is described. A complete example of the modelling knowledge and environmental knowledge is presented in appendix B.

Chapter 4
Language System

4.1 Introduction

Language System defines the user-interface to the DSS environment. The primary functions of this system are model execution and exploratory information retrieval. In this thesis we shall refer to the former function as model activation and model formulation functions. The latter function is realised using a fuzzy query language capable of retrieving and updating the environmental knowledge.

This thesis proposes a unified specification language to both knowledge system and language system. That is, constructs for model formulation and fuzzy query language closely resemble the decision model composition and decision rule structure respectively.

In the section 4.2 we shall define the model activation and formulation functions and later in section 4.3 we present the definitions related to the information retrieval. Lastly, in section 4.4 we shall summarise this presentation made in this chapter.

4.2 Mode) Activation and Formulation

Model activation, as described earlier, refers to the activation or execution of a predefined decision model. The activation of

decision model generates the possible alternatives according to the constituent specification.

Definition 4.1: Activation $\mathbb{M}_{\mu}$ of a decision model, μ, is defined as the two tuple:

$$<\mathbb{M}_n, \mathbb{M}_p>$$

where

$\mathbb{M}_n$ is the model identifier of μ,

$\mathbb{M}_p$ is a set of parameters $<v_1, v_2, \ldots v_n>$ applicable for a given activation.

Decision models are formulated using the predefined decision models according to the model composition rules.

Definition 4.2: Every decision model activation is a model formulation. Further, if $\mathbb{M}$ and $\mathbb{M}$ are valid model formulations then the following are also model formulations:

$\mathbb{M}_1 , \mathbb{M}_2$

$\mathbb{M}_1 ; \mathbb{M}_2$

$\psi \ ? \ \mathbb{M}_2$

$\psi \ ? \ \mathbb{M}_1 : \mathbb{M}_2$

$\forall \ x \in \chi \ \ \mathbb{M}_1$

$[\psi : \mathbb{M}_1]$

$(\mathbb{M}_1)$

it can be observed that this definition is similar to that of model body (Defn. 3.3). That is, the compositional rules are identical and therefore it is possible to formulate any decision model that is

definable.

Syntax:

The command syntax for the activation and formulation decision models is shown below:

ActivateCommand = activate *DecisionModelBody* . .

Example 4.1:

The command to activate a predefined decision model ***select_a_project*** with budget as Rs. 3 million is shown below:

activate **select_a_project(P,3).**

The argument P is a variable that gets bound to the selected project. The responses to activate commands are presented in the appendix B.

Example 4.2:

The usage of an imprecise model parameter is illustrated below:

activate **select_projects(fposs([10,12,14,16])).**

The budget parameter specifies the maximum budget available as *approximately between 12 and 14 million.*

Example 4.3:

A formulation of decision model using two decision stages ***select_a_project*** and ***assign_project_leader*** is shown below:

activate **(select_a_project(P,10) -> assign_project_leader(P)).**

The response to this command would be similar to the activation of a decision model defined with this composition as the decision model

body.

4.3 Information Retrieval

Retrieval of information from the environmental knowledge base for exploratory purposes is based on query specification according to the definitions of decision rules. A detailed discussion on the approximate information retrieval is presented in (Bolloju 1990b, 1991b). We shall **define,** in this section, the four basic forms of queries and illustrate the proposed syntax with some examples.

Definition 4.3: A query is defined as a **3-tuple**

$$<\Omega_\tau, \Omega_\chi, \Omega_c>$$

where

Ω_τ is a set of terms $\{\tau_1, \tau_2, \ldots, \tau_n\}$ whose values are to be retrieved,

Ω_χ is a range expression with all the variables in Ω_τ are existentially quantified, and

Ω_c is a threshold necessity-possibility measure.

Syntax:

The syntax of the retrieval query command is shown below:

RetQuery = get *TermList* [where *RangeExpression* [@ *Threshold*]] .

TermList = *Term* [, *TermList*] .

TermList in this production identifies the terms to be evaluated and then displayed. ***RangeExpression*** essentially **defines** the set of tuples of environmental knowledge base using which *TermList* can be evaluated. Finally, *Threshold* limits the selected components to a

degree of certainty and it is expressed as a necessity-possibility measure. The syntax of *RangeExpression* is listed under that of decision rules (see section 3.1.2).

Example 4.4: We shall now present a number of examples to illustrate the retrieval commands:

1) get **P.title** where prop_proj(P).

The above command retrieves the titles of all proposed projects (prop_proj is part of the factual knowledge).

2) get P.title,P.cost where proposed_project(P).

This command illustrates the use of derived knowledge. The second term in the query *cost* is derived using the procedural knowledge associated with the rule *proposed_project*.

3) get P.title,P.duration where

prop_proj(P) and **P.duration** = **~ about_2_years.**

This command shows an imprecise constant (**~ *about_2_years***) being used in the condition.

4) get P.title,P.duration where

prop_proj(P) and **P.duration = ~ about_2_years**
® [0.5,1].

The threshold necessity possibility measure limits the selected tuples.

5) get **count(P:(proposed_project(P)** and

P. cost **= ~ expensive)).**

This example has an aggregate function as its term and it indicates the count of *expensive* projects in the proposed projects.

6) get **E.all** where **employee(E).**

7) **get** E.name,P.title,R.role where
participated_as(E,R) and
team_of(P,R).

8) get E.name, count(R:(participated_as(E,R) and
R.role = leader))
where employee(E).

Transparent access to the database is illustrated in the commands 6,7 and 8. Also, in the commands 7 and 8 the ranges of variables are implicitly **defined** (see **Defn.** 3.8). Accordingly, the command 7 is equivalent to the **following**

get E.name,P.title,R.role where
employee(E) and
project(P) and
role(R) and
participated_as(E,R) and
team_of(P,R).

9) get E.name where employee(E) and
exists(R:participated_as(E,R),R.role=leader).

10) **get** E.name where **employee(E)** and

all(R:participated_as(E,R),R.role=leader).

Commands 9 and 10 depict the use of existential and universal quantification respectively.

11) get **P.name** where **probable_manager(P).**

12) get **P.name** where **probable_manager(P)** and
has_past_experience(P,database).

13) **get L.name,P.name** where **probable_manager(L)** and
proposed_project(P) and
can_handle(L,P.area,P.type) @ [0.5,1].

Commands **11,** 12 and 13 use decision rules as part of the range expression. The last command is almost as good as a decision model for selecting project-manager assignments.

Definition 4.4: Assertion of zero, one or more new n-ary facts is defined by the 4-tuple:

$$<\Box_p, \Box_\tau, \Box_\chi, \Box_c>$$

where

$\Box_p$ is a ground n-ary predicate identifier,

$\Box_\tau$ is a set of terms {T ,T ,...,T } defining the values of the respective arguments {a ,a ,...,a } of $\Box$

$\Box_\chi$ is a range expression with all variables in $\Box$ are existentially quantified, and

$\Box_c$ is a threshold necessity-possibility measure.

Syntax:

InsQuery — ins *FactName(AttrValueList)*
[where *RangeExpression* [@ *Threshold*]] . .

RangeExpression identifies the tuples that are to be used to evaluate the terms in 1 . *FactName* is a predicate identifier of factual knowledge component under which the new fact will be asserted.

Example 4.5: Some example ins commands are shown below:

1) ins **prop_proj(pid:p7,**
title:gks,
duration: **~ about_2_years,**
manpower: 2,
type: developmental,
area :**graphics,**
complexity: **~ complex).**

The above command asserts a ***prop_proj*** with the argument values as indicated by the attribute identifiers.

2) ins db_experts(P.namc) where probable_manager(P) and
can_handle(P,database,research) @ [0.5,1].

This command asserts a new set of facts ***db_experts*** that is extracted using the rule ***probable_manager*** and decision rule ***can_handle.***

3) ins proj_part(P.name,
count(R:(participated_as(P,R) and
R.role = leader)),
count(R:(participated_as(P,R) and
R.role = **deputy_leader)))**
where employee(P).

The above command illustrates assertion of factual knowledge that is Aggregated from the external database.

Definition 4.5: Retraction of zero, one or more existing n-ary facts can be defined by the 3-tuple

$<\mathbb{D}_v, \mathbb{D}_\chi, \mathbb{D}_c>$

where

$\mathbb{D}_v$ is an existentially quantified variable identifying the fact to be retracted,

$\mathbb{D}_\chi$ is a range expression with variable $\mathbb{D}$ existentially quantified, and

$\mathbb{D}_c$ is a threshold necessity-possibility measure.

Syntax:

DelQuery = del *Variable*
where ***RangeExpression*** [@ *Threshold*] .

Example 4.6:

1) del P where prop_proj(P) and **P.area = games_management.**

The above command retracts the project with the title ***games_management*** from the set of ***prop_proj.***

2) del P where prop_proj(P) and **P.duration** > 30 **@** **[0.5,1].**

This command retracts all the tuples of ***prop_proj*** with duration (fuzzily) greater than 30 months.

Definition 4.6: Replacing of zero, one or more n-ary facts with new argument values is defined by the 4-tuple:

$<\mathbb{R}_v, \mathbb{R}_\tau, \mathbb{R}_\chi, \mathbb{R}_c>$

where

- $\mathbb{R}_v$ is a variable representing the n-ary fact to be replaced,
- $\mathbb{R}_\tau$ is a set of terms $\{\tau_1, \tau_2, \ldots, \tau_n\}$ defining the values of the respective arguments $\{a_1, a_2, \ldots, a_n\}$ of the predicate **identified** by the variable R ,
- $\mathbb{R}_\psi$ is a range expression with variable $\mathbb{R}$ existentially quantified, and
- $\mathbb{R}_c$ is a threshold necessity-possibility measure.

Syntax:

RepQuery - **rep** *Variable* with ***(AttrValueAssignList)***
where *RangeExpression [@ Threshold]*

The syntax of ***AttrValueAssignList*** is presented along with that of decision rules. All the n-ary tuples identified by the *RangeExpression* will be replaced with the new attribute values as defined by the *AttrValueAssignList.*

Example 4.7:

rep P with **(P.duration: = P.duration*** 1.2) where **prop_proj(P).**

This example illustrates an increment (fuzzy) of the project duration by 20% to all ***prop_proj.***

4.4 Summary

We have presented the definitions related to the two most important functions of the language system in this chapter. Activation of **predefined** decision models is merely by the specification of the model identifier along with its parameter values. Model formulation is according to the definition of decision model **body,** discussed in chapter 3. This formulation is non-procedural and is supported by high level constructs. We have also considered the basic commands of retrieval and update. Again, the query language is non-procedural and facilitates retrieval of derived knowledge also. We have, however, not considered the other miscellaneous functions such as knowledge system maintenance.

This query language can be easily proved as relationally complete since all the basic algebraic operations can be expressed using the four forms of the commands presented. The proof is analogous to that described in (Ullman 1984, pp. 193-194). Additionally, it has the extensions to query imprecise and uncertain information using possibly imprecise queries.

The proposed syntax of the language system has been presented with illustrative examples. We, **however,** have not listed the response

from PPS to the example commands. Appendix B presents an example session using the commands presented in the examples of this chapter using the prototype implementation.

Chapter 5
Problem-Processing System

5.1 Introduction

Problem-Processing System (PPS) is concerned with all the processing abilities related to model activation and information retrieval. PPS defines the processing operations required for a given model activation or information retrieval request. These operations can be broken down into simpler operations according to the layers of modelling knowledge. In the next section we shall define some preliminary operations before describing the operations involved in model activation. In section 5.4 we shall present the definitions for the operations related to information retrieval.

5.2 The Preliminaries

5.2.1 Semantic Unification

The unification described in logic programming (e.g., Rich 1983, pp. 154-157) is not applicable under imprecise and/or uncertain argument values. For example, while answering the query:

retrieve titles of proposed projects with
duration between 24 and 36 months

using

```
?- prop_proj(title:N,duration:cposs([24,36])),
```

the normal unification algorithm fails since it is meant for precise

and certain atomic values. It is, therefore, necessary to unity the imprecise and/or uncertain values appropriately. We can identify two issues related to the unification of such values:

a) concept of pattern and datum,

b) uncertain result of unification.

The former can be illustrated using an example of unifying values:

A = cposs([5,8]) and B = cposs([6,7]).

Intuitively, we can infer that B can be certainly unified with A since B is included in A. That is, for example, if one is looking for projects with duration *between 5 and 8* a project with a duration of *between 6 and 7* gets qualified certainly. On the other hand, a project with a duration of *between 5 and 8* can not, at least as certainly, get qualified when looking for projects *between 6 and* 7. Consequently, unifying A with B is different from unifying B with A and we need to distinguish between the pattern and the datum.

Issue (b) is concerned with the result of unification. We can definitely infer that the pattern cposs([10,13]) cannot be unified with the datum of cposs([5,8]), and, the pattern cposs([6,7]) can be certainly unified with the datum cposs([5,8]). However, we cannot be certain whether the pattern cposs([7,10]) can be unified with the datum cposs([5,8]). It follows from the above that we cannot certainly, always, unify two imprecise and/or uncertain values. Consequently, it is logical to use a necessity-possibility pair to represent the uncertain result of an unification. The following definitions are presented as fuzzy pattern matching in (Prade

1985a, 1985b) and as semantic unification (Baldwin 1987a).

Given the universe of discourse U with pattern *P* and datum *D* defined on U, the possibility (77) of *P* matching *D* is:

$$\Pi(P:D) = \sup_{u \in U} \min (\pi_P(u), \pi_D(u))$$

And, the necessity (*N*) of *P* matching *D* is:

$$N(P:D) = \inf_{u \in U} \max (\pi_P(u), 1-\pi_D(u))$$

$$\text{or } 1 - \sup_{u \in U} \min (1-\pi_P(u), \pi_D(u))$$

The definitions presented above are for fuzzy sets and possibility distributions representing fuzzy sets. Tables 5.1 and 5.2 depict the definitions of necessity-possibility (NP) measures applicable for various combinations of imprecise values. In fact, these definitions are, essentially, adaptation of the above definition to the specific cases.

The symbols listed below are used in tables 5.1 and 5.2:

μ_D, μ_P are the membership functions of datum and pattern,

π_D, π_P are the possibility distribution of datum and pattern,

c_D, c_P are *cores* of datum and pattern,
where core c = {x | $\pi(x) = 1$},

s_D, s_P are *supports* of datum and pattern,
where support s = {x | $\pi(x) > 0$},

l_D, u_D are lower and upper bound of D, and

ϕ represents a null set.

N Pattern P	Datum D		
	atomic	cset	fset
atomic	1 if D = P	1 if {P} = D	$\inf_{x \in D} max(x = P, 1-\mu_D(x))$
cset	1 if D ∈ P	1 if D ⊆ P	$\inf_{x \in D} max(x \in P, 1-\mu_D(x))$
fset	$\mu_P(D)$	$\min_{x \in D} \mu_P(x)$	$\inf_{x \in D} max(\mu_P(x), 1-\mu_D(x))$
cposs	$\pi_P(D)$	1 if $D \subseteq \{x \mid \pi_P(x) = 1\}$	$\inf_{x \in D} max(\pi_P(x), 1-\mu_D(x))$
fposs	$\pi_P(D)$	$\min_{x \in D} \pi_P(x)$	$\inf_{x \in D} max(\pi_P(x), 1-\mu_D(x))$

Table 5.1 (a) Adaptation of Necessity measure

N	Datum D	
Pattern P	cposs	fposs
atomic	1 if $\mathbb{L}_D = \mathbb{U}_D = P$	$\inf_{x \in S_D} max(x=P, 1-\pi_D(x))$
cset	1 if $S_D \subseteq P$	$\inf_{x \in S_D} max(x \in P, 1-\pi_D(x))$
fset	$\min_{x \in \mathbb{C}_D} \mu_P(x)$	$\inf_{x \in S_D} max(\mu_P(x), 1-\pi_D(x))$
cposs	1 if $\mathbb{C}_D \subseteq \mathbb{C}_P$	$\inf_{x \in S_D} max(\pi_P(x), 1-\pi_D(x))$
fposs	$\inf_{x \in \mathbb{C}_D} max(\pi_P(x), 1-\pi_D(x))$	$\inf_{x \in S_D} max(\pi_P(x), 1-\pi_D(x))$

Table 5.1 (b) Adaptation of Necessity measure

Π	Datum D		
Pattern P	atomic	cset	fset
atomic	1 if D = P	1 if $P \in D$	$\mu_D(P))$
cset	1 if $D \in P$	1 if $P \cap D \neq \phi$	$\max_{x \in P}(\mu_D(x))$
fset	$\mu_P(D)$	$\max_{x \in D} \mu_P(x)$	$\sup_{x \in D} min(\mu_P(x), \mu_D(x))$
cposs	$\pi_P(D)$	1 if $\mathbb{C}_P \cap D \neq \phi$	$\max_{x \in \mathbb{C}_P \cap D}(\mu_D(x))$
fposs	$\pi_P(D)$	$\max_{x \in D} \pi_P(x)$	$\sup_{x \in D} min(\pi_P(x), \mu_D(x))$

Table 5.2 **(a)** Adaptation of Possibility Measure

Pattern P	Datum: cposs	Datum: fposs
atomic	$\pi_D(P)$	$\pi_D(P))$
cset	1 if $\mathbb{C}_D \cap P \neq \phi$	$\max_{x \in P}(\pi_D(x))$
fset	$\sup_{x \in \mathbb{C}_D \cap P}(\mu_P(x))$	$\sup_{x \in S_D \cap P} min(\mu_P(x), \pi_D(x))$
cposs	1 if $\mathbb{C}_D \cap \mathbb{C}_P \neq \phi$	$\sup_{x \in S_D \cap S_P} min(\pi_P(x), \pi_D(x))$
fposs	$\sup_{x \in S_D \cap S_P} min(\pi_P(x), \pi_D(x))$	$\sup_{x \in S_D \cap S_P} min(\pi_P(x), \pi_D(x))$

Table 5.2 (b) Adaptation of Possibility Measure

Table 5.3 below depicts some examples of semantic unification.

Pattern	Datum	Result
fposs([1,2,3,4])	A	[1,1]
cposs([4,10])	*cposs([5,8])*	[1,1]
cposs([5,8])	*cposs([4,10])*	[0,1]
cposs([3,7])	*fposs([2,4,6,8])*	[0.5,1]
fposs([3,5,6,9])	*cset([2,4,6])*	[0,1]
6	*fposs([3,5,5,7])*	[0,0.5]
cposs([4,6])	*fset([3^0.4,4^0.5,5^1,6^0.5])*	[0.6,1]
~about_33y	*~about_32y*	[0,1]
~very_good	*~excellent*	[0,0.75]

Table 5.3 Examples of semantic unification

5.2.2 Combination of NP-measures

In the previous subsection we described the uncertain result of unification with respect to a single pattern and a single datum. In

this subsection, we shall discuss the possible methods for combining two or more uncertain unification results. Later, we shall also present the result of unification under negation.

The various methods employed in evaluating a combined truth value under conjunction and disjunction are directly related to the concept of t-norms and co-t-norms respectively (Prade 1985a). The definitions of t-norms and co-t-norms are presented in appendix F. Among these various alternatives the min and max norms are extensively used in the combination of NP-measurcs. Under this approach if $[n_1,\pi_1]$ and $[n_2,\pi_2]$ are the two respective uncertain unification results of conditions C1 and C2 then

a) the result of C1 **and** C2 can be expressed as $[n,\pi]$ where

$$n = min(n_1,n_2) \quad \text{and} \quad \pi = min(\pi_1,\pi_2)$$

b) the result of C1 **or** C2 can be expressed as $[n,\pi]$ where

$$n = max(n_1,n_2) \quad \text{and} \quad \pi = max(\pi_1,\pi_2)$$

Lastly, if $[n,\pi]$ is the unification result of condition C then the result of $\neg$ C can be expressed as $[\bar{n},\bar{\pi}]$ where

$$\bar{n} = 1 - \pi \qquad \text{and} \qquad \bar{\pi} = 1 - n$$

5.2.3 Applicability

For a given decision model it is possible to have more than one decision model definition (sec Definition 3.2) with the same decision model identifier. That is, a decision model can have more than one definition each catering to a specific case. Similarly, decision stages, decision rules and rules representing procedural

knowledge of environmental knowledge can have more than one definition. Let us consider the applicability of decision models for a given decision model activation. We can, on similar lines, apply this concept of applicability to the other knowledge components.

The set of decision models of relevance to an activation $<M_n, M_p>$ can be defined as:

$$S_{M_n} = \{ \mu_i \mid \mu_i \in M \wedge \mu_i^{id} = M_n \}$$

where

M is the set of decision models, and

μ_i^{id} is the identifier of decision model μ_i.

The applicability of a given decision model μ during the model activation $<M_n, M_p>$ can be determined by semantically unifying the parameters M_p with the respective arguments of the model head of μ_i. It is possible to define a threshold (NP_μ) on the result of such unification to ascertain whether a given decision model is applicable or not. This threshold NP-measure is common to all the activation of any decision model. The set of applicable decision models can, thus, be defined as:

$$A_{M_n} = \{ \mu_i \mid \mu_i \in S_{M_n} \wedge NP_{\mu_i \mid M_n} \geq NP_\mu \}$$

The necessity-possibility measure $NP_{\mu_i \mid M_n}$ can be defined as:

$$NP_{\mu_i \mid M_p} = \left[\min_{j \in [1,k]} (n_j), \min_{j \in [1,k]} (\pi_j) \right]$$

where

k is the number of parameters in H ,

π_j is the necessity measure unifying j^{th} argument of M with the respective argument of the head of the decision model μ_i, and

π_j is the possibility measure unifying j^{th} argument of M with the respective argument of the head of the decision model μ .

The semantics of the comparison operator $\geq$ are described in section 5.2.6.

5.2.4 Fuzzy Comparison

We have discussed the equality (semantic unification or fuzzy pattern matching) in the section 5.2.1. The inequality can be treated as the negation on the equality. The results of other forms of comparison, 0, can be defined using the following:

$$\Pi(P\theta D) = \sup_{\substack{u\theta v \\ u\in U, v\in V}} \min(\pi_P(u), \pi_D(v))$$

$$N(P\theta D) = 1 - \Pi(P\bar{\theta}D)$$

Some examples of the application of this definitions are shown below:

Comparison	Result
fposs([10,20,30,40]) > fposs([60,70,80,90])	[0,0]
fposs([10,20,30,40]) > fposs([10,20,30,40])	[0,1]
fposs([20,30,40,50]) > fposs([10,20,30,40])	[0,1]
fposs([30,40,50,60]) > fposs([10,20,30,40])	[0.5,1]
fposs([40,50,60,70]) > fposs([10,20,30,40])	[1,1]
~excellent > ~very_good	[0.25,1]
~excellent < ~very_good	[0.25,0.75]

Table 5.4 Examples of fuzzy comparison

5.2.5 Fuzzy Arithmetic

Fuzzy arithmetic subsumes ordinary arithmetic and the four basic operations can be defined using:

$$\pi_{A*B}(w) = \sup_{\substack{w=u*v \\ u\in U,\, v\in V}} \min(\pi_A(u), \pi_B(v))$$

The above, essentially, produces results as possibility distributions. However, in order to preserve the property of unimodal possibility distribution for the representation of imprecision, the arguments with the discrete possibility distribution **(cset, fset)** are approximated to trapezoidal possibility distribution (cposs, fposs) as shown below:

Discrete possibility distribution Π_d is approximated to continuous possibility distribution with bounds β_l and β_u:

if $\pi_d(\beta) = 0$ or $1 \quad \forall\, \beta \in B$

where

B is the base domain,

$\beta_l = \min_{\beta\in B \wedge \pi_d(\beta)=1} (\beta)$ and

$\beta_u = \max_{\beta\in B \wedge \pi_d(\beta)=1} (\beta)$

otherwise to trapezoidal possibility distribution defined by $<\beta_a,\beta_b,\beta_c,\beta_d>$ with:

$$\beta_a = \beta \in B \wedge \min_{\pi_d(\beta)>0} (\beta) - 1$$

$$\beta_b = \beta \in B \wedge \min_{\pi_d(\beta)=1} (\beta)$$

$$\beta_c = \beta \in B \wedge \max_{\pi_d(\beta)=1} (\beta) \quad \text{and}$$

$$\beta_d = \beta \in B \wedge \max_{\pi_d(\beta)>0} (\beta) + 1$$

Fig. 5.1 illustrates the basic operations of add, subtract, multiply and divide using trapezoidal possibility distributions. Some examples of fuzzy arithmetic are shown below:

fposs([2,3,4,5]) + fposs([6,7,8,9]) = fposs([8,10,12,14])

fposs([20,25,30,40]) - fposs([5,10,12,15]) = **fposs([5,13,20,35])**

fposs([2,3,4,5]) * fposs([3,4,5,6]) = **fposs([6,12,20,30])**

fposs([10,20,30,40]) / fposs([2,3,4,5]) = fposs([2,5,10,20])

rposs([2,3,4,5]) + cset([3,5]) = **fposs([5,6,9,10])**

cset([3,5]) + cset([5,6,7]) = cposs([8,12])

fset([2^0.5,4^1,5^0.5]) + 5 = **fset([7^0.5,9^1,10^0.5])**

(~excellent + **~very_good)/2** = **fposs([60,80,90,95])**

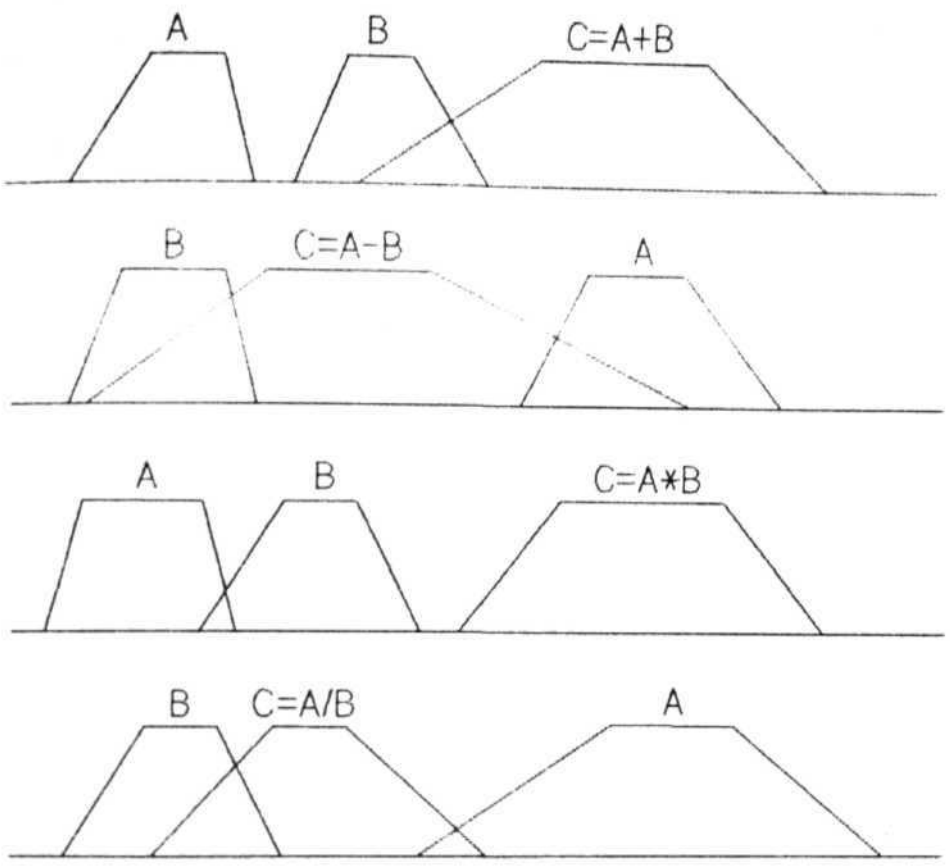


Fig. 5.1 Fuzzy arithmetic

5.2.6 Comparison of Necessity-Possibility Measures

We have used the comparison of two NP-measures in the definition of applicability of decision models (section 5.2.3). The comparison of two NP-measures cannot always certainly yield a truth value. In the following comparison:

$$[n_1,\pi_1] \geq [n_2,\pi_2]$$

if $(n_1 \geq n_2) \wedge (\pi_1 \geq \pi_2)$ then the comparison yields a certain truth value (T). On the other hand, if $(n_1 < n_2) \wedge (\pi_1 < \pi_2)$ then the comparison yields a certain false value ($\perp$).

The result of comparison is uncertain if

$$(n_1 \geq n_2) \oplus (\pi_1 \geq \pi_2)$$ where $\oplus$ stands of exclusive-or.

We designate such a comparison as uncertain and neither of the operands *dominates* the other. We represent this non-dominating case with a necessity-possibility measure [0,1]. The certain true and the certain false cases are represented by necessity-possibility measures [1,1] and [0,0] respectively.

5.3 Model Activation and Formulation

Model activation refers to the activation of a predefined decision model. Model formulation, on the other hand, refers to the composition of a new decision model using the predefined decision models **and** decision stages for activation. Since the formulation is according to **the** compositional operators (see Definition 4.2) we can deduce that **the** activation of a formulation is equivalent to that of a decision model with the body as the formulation. Consequently, in the remaining part of this subsection it is sufficient to consider only model activation.

For a given decision model identifier it is possible that there can **be more than one** decision model defined. The applicability of the available decision models needs to be established using the

parameters supplied for the activation. We can, then, select the most applicable decision model for activation based on applicability (see section 5.2.3).

In the following subsection we shall present the **definitions** related to decision model activation. In the subsequent subsections, we shall define the alternatives of a decision stage and the operational semantics of decision rules.

5.3.1 Alternatives of a Decision Model

Definition 5.1: We can define the set of decision alternatives A_μ for an activated decision model μ as:

i) $A_\mu = A_\sigma$ if μ is a decision stage σ, i.e., the set of alternatives of the decision model μ are same as that of the decision stage σ.

ii) $A_{\mu 1, \mu 2} = A_{\mu 1} X A_{\mu 2}$ where $A_{\mu 1,\mu 2}$ is the cross product of the sets of alternatives of decision models μ_1 and μ_2.

iii) $A_{\mu 1 ; \mu 2} = A_{\mu 1} X A_{\mu 2}$ where $A_{\mu 1;\mu 2}$ is the cross product of the sets of alternatives of decision models μ_1 and μ_2. However, the order of evaluation of $A_{\mu 1}$ and $A_{\mu 2}$ is not important.

iv) $A_{\psi ? \mu 1} = A_{\mu 1}$ if $r_c(\psi) = T$

$= \otimes$ otherwise

where

$\otimes$ is a *unit alternative* defined by

$$A \times \otimes = A$$

$$\otimes \times A = A$$

$$\otimes \times \otimes = \otimes$$

r_c is a result function (see section 5.2.6) and

$\top$ is a certain truth value defined as [1,1].

v) $A_{\psi?\mu_1:\mu_2} = A_{\mu_1}$ if $r_c(\psi) = \top$

$= A_{\mu_2}$ otherwise

vi) $A_{[\psi?\mu_1]} = A_{\mu_1} \times A_{[\psi?\mu_1]}$ if $r_c(\psi) = \top$

$= \otimes$ otherwise

vii) $A_{\forall x \in \chi\ \mu_1} = A^1_{\mu_1} \times A^2_{\mu_1} \times \ldots \times A^n_{\mu_1}$

where

$A^i_{\mu_1}$ is the set of alternatives of A_{μ_1}

for $x = x_i$, $x_i \in \chi$

The above definition shows the alternatives to decision model activation under various forms of model composition. This subsumes the **definition** of formulated decision models since the formulations are made using the same compositional operators.

5.3.2 Alternatives of a Decision Stage

The identification of applicable decision stages during a given model activation or formulation is similar to that of decision models. In this subsection, we shall define the alternatives to an applicable decision stage during a specific model activation.

Definition 5.2: The set of probable alternatives Ω_σ to a decision stage σ is defined as:

i) $\{ t \mid \omega(t) \}$ if predicate expression ω is a predicate symbol belonging to the environmental knowledge.

ii) $\{ s \mid s \subseteq \{t \mid \omega(t)\} \}$ if predicate expression is of the form ω^* where ω is a predicate symbol.

iii) $\{ <t_1,t_2> \mid \omega(t_1) \wedge \omega(t_2) \}$ if the predicate expression is of the form $\omega_1 \times \omega_2$.

Definition 5.3: Given a set of probable alternatives Ω to a decision stage σ, the set of feasible alternatives A can be defined as:

$$A_\sigma = \{ \alpha \mid \alpha \in \Omega_\sigma \wedge \psi_1(\alpha) \wedge \phi(\alpha) \wedge \psi_2(\alpha) \}$$

That is, the subset of the probable alternatives that is satisfying the precondition ψ, action ϕ, and postcondition w is termed as the set of feasible alternatives. Further, with each alternative or $\in$ A a rank value is associated as:

$r_c(\kappa)$ if κ is a condition rule

$r_v(\kappa)$ if κ is a term.

5.3.3 Evaluation of Decision Rules

The identification of applicable decision rules during a decision

model activation (using the decision stage specification) is similar to that of decision models. In this section we shall present **the** method of evaluation of decision rules of the three categories.

Condition Rules:

Definition **5.4:** The result of evaluation of a given condition rule ψ is **defined** by the function $\mathbf{r}$ as shown below:

i) If **the** condition rule refers to another rule as:

$<\mathbb{R}_n,\mathbb{R}_p>$

where

$\mathbb{R}_n$ is the decision rule identifier and

$\mathbb{R}_p$ is the set of parameters of the decision rule

then, the result $r_c = [n,\pi]$ defined as:

$$n = \max_{\rho \in A_{\mathbb{R}_n}} min\ (n_\rho \mid <\mathbb{R}_n,\mathbb{R}_p>,\ n_{\rho_b},n_c) \quad \text{and}$$

$$\pi = \max_{\rho \in A_{\mathbb{R}_n}} min\ (\pi_\rho \mid <\mathbb{R}_n,\mathbb{R}_p>,\ \pi_{\rho_b},\pi_c)$$

where

$A_{\mathbb{R}_n}$ is the set of applicable decision rules,

$[n_\rho \mid <\mathbb{R}_n,\mathbb{R}_p>,\pi_\rho \mid <\mathbb{R}_n,\mathbb{R}_p>]$ is the applicability of decision rule ρ,

$[n_{\rho_b}, \pi_{\rho_b}]$ is the result of the decision rule body ρ_b of the decision rule ρ, and

$[n_c, \pi_c]$ is the certainty (ρ_c) associated with the decision rule

ii) $r_c(p_2(\tau_1,\tau_2)) = r_c(p_2(r_v(\tau_1),r_v(\tau_2)))$

if the condition of the form $p_2(\tau_1,\tau_2)$ (see definition 3.7)

where r_v is term evaluation function (Definition 5.7)

iii) $r_c(p_3(y_1,y_2,\ldots,y_n)) = \gamma_c$

if the condition refers to a relationship p_3 with variables $y_1,y_2,\ldots,y_n$.

where γ_c is the certainty associated with the tuple $\gamma_t = p_3(x_1,x_2,\ldots,x_n)$ with substitution x_i/y_i.

Further, if $r_c(\psi_1) = [n_1,\pi_1]$ and $r_c(\psi_2) = [n_2,\pi_2]$ are the evaluation results of ψ_1 and ψ_2 respectively then the results of other forms of condition rule is defined as:

iv) $r_c(\psi_1 \wedge \psi_2) = [min\ (n_1,n_2), min\ (\pi_1,\pi_2)]$

v) $r_c(\psi_1 \vee \psi_2) = [max\ (n_1,n_2), max\ (\pi_1,\pi_2)]$

vi) $r_c(\neg\ \psi_1) = [1-\pi_1, 1-n_1]$

vii) $r_c((\psi_1)) = r_c(\psi_1)$

viii) $r_c(\forall\ x \in \chi\ \psi_1) = [\min_{x_i \in \chi}(n_1^i), \min_{x_i \in \chi}(\pi_1^i)]$

where $[n_1^i,\pi_1^i]$ is the result of ψ_1 for $x = x_i$.

ix) $r_c(\exists\ x \in \chi\ \psi_1) = [\max_{x_i \in \chi}(n_1^i), \max_{x_i \in \chi}(\pi_1^i)]$

where $[n_1^i, \pi_1^i]$ is the result of ψ_1 for $x = x_i$

x) $r_c(\mathcal{Q}\ x \in \chi\ \psi_1) = r_{\mathcal{Q}}(\Sigma\ n_1^i, \Sigma\ \pi_1^i, |\ \chi\ |)$

where $[n_1^i, \pi_1^i]$ is the result of ψ_1 for $x = x_i$

$r_{\mathcal{Q}}$ is a function associated with the fuzzy quantifier $\mathcal{Q}$ which maps the sums of necessity and possibility measures using the cardinality of χ to an NP-measure.

xi) $r_c(\mathcal{Q}_m\ \mathcal{Q}\ x \in \chi\ \psi_1) = r_{\mathcal{Q}_m}(r_{\mathcal{Q}}(\ldots))$

where $r_{\mathcal{Q}_m}$ maps the result of $r_{\mathcal{Q}}$ to an NP-measure.

This definition covers all forms of condition rule as defined in chapter 3 (see Definition 3.7).

Definition 5.5: A range expression χ identifies a set of tuples as:

$$\{\ x\ |\ p(x) \wedge p_1(y_1) \wedge p_2(y_2) \wedge \ldots \wedge p_n(y_n) \wedge \psi(x)\ )\ \}$$

A result NP-measure is associated with each $x_i \in \chi$ which is defined as:

$$\left[\ min\ (\ n_p, n_i, \min_{j \in [1,n]}(\ n_{p_j})),\quad min\ (\ \pi_p, \pi_i, \min_{j \in [1,n]}(\ \pi_{p_j}))\ \right]$$

where

$r_c(p(x)) = [n_p, \pi_p]$,

$r_c(p_j(y_j)) = [n_{p_j}, \pi_{p_j}]$ and

$r_c(\psi(x_i)) = [n_i, \pi_i]$

Action Rule:

Definition 5.6: The result of an action rule is defined by the function r_a which maps the factual knowledge components of environmental knowledge from state *s* to state *t* as defined below:

i) $r_a(\mathbf{I}(\mathbf{p}(\tau_1,\tau_2,\ldots,\tau_n))) = \Gamma \cup \mathbf{p}(r_v(\tau_1),r_v(\tau_2),\ldots,r_v(\tau_n))$

where Γ is the factual knowledge, and

$v(\tau_i)$ is the value of the term τ_i

ii) $r_a(\mathbf{D}(x)) = \Gamma - \mathbf{p}(a_1(x),a_2(x),\ldots,a_n(x))$

where $a_i(x)$ is the i^{th} argument value of the n-ary tuple identified by the variable x

iii) $r_a(\mathbf{R}(x/y,(a_1/\tau_1,a_2/\tau_2,\ldots a_n/\tau_n))) =$

$(\Gamma - \mathbf{p}(a_1(x),a_2(x),\ldots,a_n(x)))$

$\cup \quad \mathbf{p}(r_v(\tau_1),r_v(\tau_2),\ldots,r_v(\tau_n))$

Further, if $r_a(\phi_1)$ and $r_a(\phi_2)$ are results of the action rule bodies ϕ_1 and ϕ_2 then the results of other forms of action rule bodies can be defined as:

iv) $r_a(\phi_1,\phi_2) = r_a(\phi_1),\ r_a(\phi_2)$

v) $r_a([\psi\ ,\phi_1]) = r_a(\phi_1),\ r_a([\psi\ ,\phi_1])$ if $r_c(\psi) = T$

$= \varepsilon$ otherwise

where ε represents a null action

vi) $r_a(\psi\ ?\ \phi_1:\phi_2) = r_a(\phi_1)$ if $r_c(\psi) = T$

$= r_a(\phi_2)$ otherwise

vii) $r_a(\psi\ ?\ \phi_1) = r_a(\phi_1)$ if $r_c(\psi) = T$

$= \varepsilon$ otherwise

viii) $r_a(\forall x \in \chi\ \phi_1) = r_a(\phi_1^1), r_a(\phi_1^2),\ldots,r_a(\phi_1^n)$

where $r_a(\phi_1^i)$ is the result of ϕ_1 for $x = x_i$

ix) $r_a((\phi_1)) = r_a(\phi_1)$

Rank Rule:

Rank rules arc of the form of either condition rule or term. We have defined the result function r_c for condition rule. We shall now define the result function for the second case of the rank rule.

Definition 5.7: The result of evaluation of a term τ is defined by the function r_v as shown below:

i) $r_v(c) = c$ if τ is a constant c,

ii) $r_v(a_i(x)) = a_i(x)$ if τ is the i^{th} argument value of the variable x,

iii) $r_v(f(\tau_1,\tau_2,\ldots,\tau_n)) = f(r_v(\tau_1),r_v(\tau_2),\ldots,r_v(\tau_n))$ if τ is defined as a function f,

iv) $r_v(a_f(\tau',\chi)) = \underset{x\in\chi}{a_f}(r_v(\tau'))$ if τ is defined as an aggregate function a_f on term τ'.

5.3.4 Inferencing on Environmental Knowledge

Inferencing on environmental knowledge during the decision rule interpretation closely follows the semantics described earlier for decision rules. Firstly, while proving a given goal the set of applicable rule is identified by semantically unifying the arguments of the goal with the arguments of the rules with the same

identifier. Then, the antecedent of the rule is evaluated using the t-norms and co-t-norms for combining the resulting NP-measures of the subgoals. Lastly, the result of the consequent is arrived at using the t-norm associated with the implication.

The subgoals are recursively evaluated to terminate either at a predicate of factual knowledge or a system predicate such as >, > =, etc. We have considered the evaluation of these in section 5.2.

5.4 Information Retrieval

Information retrieval for exploratory purposes is the second important function of the language system. We have discussed the definitions and syntax of the four forms of query commands in the chapter 4. The processing of the query commands is entirely based on the processing of the range expression and decision rules. In this section we shall present the operational semantics of the query functions.

5.4.1 Retrieval Query

The set of n-ary tuples selected by the query $<\Omega, \Omega, \Omega>$ is defined as:

$$\left\{ (<r_v(\tau_1), r_v(\tau_2), \ldots, r_v(\tau_n)>, <n,\pi>) \mid \Omega_\chi \wedge <n,\pi> \geq \Omega_c \right\}$$

where

$r_v(\tau_i)$ is the value of term $\tau_i \in \Omega_\tau$,

$<n,\pi>$ is the NP-measure associated with the tuple $<r_v(\tau_1), r_v(\tau_2), \ldots, r_v(\tau_n)>$.

5.4.2 Insert Query

The result of insert query $<\mathbb{I}_p,\mathbb{I}_\tau,\mathbb{I}_\chi,\mathbb{I}_c>$ is defined as:

$$\Gamma \cup \left\{ (p(r_v(\tau_1),r_v(\tau_2),\ldots,r_v(\tau_n)),<n,\pi>) \mid \mathbb{I}_\chi \wedge <n,\pi> \geq \mathbb{I}_c \right\}$$

where

p	is the predicate symbol identified by $\mathbb{I}_p$,
$r_v(\tau_i)$	is the value of term $\tau_i \in \mathbb{I}_\tau$,
$<n,\pi>$	is the NP-measure associated with the tuple $<r_v(\tau_1),r_v(\tau_2),\ldots,r_v(\tau_n)>$.

5.4.3 Delete Query

The result of delete query $<\mathbb{D}_v,\mathbb{D}_\chi,\mathbb{D}_c>$ is defined as:

$$\Gamma - \left\{ (p(a_1(x),a_2(x),\ldots,a_n(x)),<n',\pi'>) \mid \mathbb{D}_\chi \wedge <n,\pi> \gtrsim \mathbb{D}_c \right\}$$

where

$a_i(x)$	is the value of i^{th} argument of variable x identified by $\mathbb{D}_v$,
$<n',\pi'>$	is the NP-measure associated with the tuple idertified by the variable x, and
$<n,\pi>$	is the result of evaluation of $\mathbb{D}_\chi$.

5.4.4 Replace Query

The result of replace query $<\mathbb{R}_v,\mathbb{R}_\tau,\mathbb{R}_\chi,\mathbb{R}_c>$ is defined as:

$$\Gamma - \left\{ (p(a_1(x),a_2(x),\ldots,a_n(x)), <n',\pi'>) \mid \mathbb{R}_\chi \wedge <n,\pi> \geq \mathbb{R}_c \right\}$$

$$\cup \left\{ (p(r_v(\tau_1),r_v(\tau_2),\ldots,r_v(\tau_n)), <n,\pi>) \mid \mathbb{R}_\chi \wedge <n,\pi> \geq \mathbb{R}_c \right\}$$

where

p	is the predicate symbol identified by $\mathbb{R}_p$,
$r_v(\tau_i)$	is the value of term $\tau_i \in \mathbb{R}_\tau$,
$a_i(x)$	is the value of i^{th} argument of variable x identified by $\mathbb{R}_v$,
$<n',\pi'>$	is the NP-measure associated with the tuple identified by the variable x, and
$<n,\pi>$	is the result of evaluation of $\mathbb{R}_\chi$.

5.5 Summary

We have presented the semantics of various of operations of Problem-processing System in this chapter. After defining the preliminary operations such as semantic unification, fuzzy arithmetic, applicability, etc., the operational semantics of the two important functions of language system have been defined. It can be observed that the implementation of prototype system described in chapter 6 closely follows these definitions.

Chapter 6
PlanAid: A Prototype Implementation

6.1 Introduction

In this chapter, we shall present the implementation details of a prototype, PlanAid, based on the proposed decision modelling formalism and the decision support system environment. This prototype has been developed in C-Prolog on VAX-11/750 computer system. All the important features proposed in this thesis have been implemented with a simple command-oriented user interface. However, lesser importance has been given to the implementation of non-key features. Firstly, the efficiency of the implementation has not been considered as a major criteria. Then, the reporting of errors is very much limited. This prototype interfaces to the databases of Admin (Naveen *et al* 1983), a network data model DBMS, and the implementation of this part is reported in (Bolloju and Kamath 1989b).

We have chosen Prolog as the implementation language for the prototype, for the following reasons:

a) rule-based,

b) user definable ops,

c) met a-programming, and

d) rapid prototyping.

Since Prolog and the proposed decision modelling formalism are rule-based, the problems that arise due to possible impedance

mismatch are minimised. It is quite natural to extend and build various layers of interpreters based on the inference mechanism provided by Prolog. The **user-definable** operators (ops), a feature of Prolog, is extremely **useful** in realising the proposed syntax. We shall elaborate this aspect in a later section of this chapter. Then, the power of meta-programming makes the implementation of generic procedures with clauses (or procedures) as arguments very simple and elegant. Lastly, because of the high level nature of the language it is possible to develop the system rapidly.

6.2 Knowledge System

As discussed earlier in chapters 1 and 3 the knowledge system is only a representational system. The implementation of this system is, therefore, only the representational aspects of modelling and environmental knowledge components.

The op declaration of Prolog facilitates structures such as ***f(a,b)*** to be expressed as ***a*** *f b* with / being a functor. Using this facility it is possible to declare prefix, infix and postfix functor definitions. **The** following set of ops has been defined to enable the representation of valid syntactic sequences according to the proposed syntax as Prolog structures.

```
% op precedence declarations
:-op(1200,xfx,'<-').
:-op(1150,xfx,@).
:-op(1140,fx,[select,foreach,if]).
:-op(1135,fx,subset).
:-op(1135,xfx,then).
:-op(1133,xfx,else).
:-op(1130,xfx,[using,do]).
:-op(1129,xfx,[of,in]).
:-op(1100,xfy,[or,'||']).
:-op(1000,xfy,[and,'->']).
:-op(990,fx,[precond,action,postcond,rank]).
:-op(800,xfx,with).
:-op(700,xfx,[:=,by]).
:-op(200,xfx,':').
:-op(100,xfx,'.').
```

% user defined fuzzy relational operators are at level 700

Let us consider a decision stage ***select_a_project*** to illustrate the syntactic representation using the above listed declarations. The **specification** of the decision stage is shown below using the proposed syntax:

```
select_a_project(P,Budget) <-
        select P of proposed_project
            using
                precond ((promising(P) or
                            expertise_available(P)) and
                        ( P.cost =< Budget)),
                rank    100/P.duration+2*P.cost.
```

This is a Prolog structure with the above listed functors and it is same as the following structure:

```
<- (select_a_project(',')(P,Budget)),
    select(using(of(P,
                    proposed_project),
                 ',')(precond (and (or (    promising(P),
                                            expertise_available(P)),
                                       =< (.'(P,cost),Budget)),
                                  ),
                       rank ('+'('/'(100,'.'(P.duration)),
                                  ('*'(2,'.'(P,cost)))
                             )
                    )
                )
           )
   )
```

The declaration of ops, thus, provides syntactic convenience while reading **and** writing Prolog terms using the built-in Prolog predicates **read** and **write.** The predicate **read** rejects the invalid syntactic sequences by flagging an error message *syntax error.* The implementation of parser is, therefore, trivial and effective.

Various components of the knowledge system are represented as Prolog predicates with **the** following structures:

Modelling Knowledge:

```
f_model( Model_id,          % decision model identifier/name
         Parameters,
         Model_body).       % model body is a Prolog structure

f_stage( Stage_id,          % decision stage identifier/name
         Parameters,
         Alt_var,           % variable identifying the current
                            % alternative
         Alt_type,          % either subsetof or elementof
         P_Predicate,       % predicate that identifies the set
                            % of probable alternatives
         Precond,           % precondition (Prolog structure)
         Action,            % action          "
         Postcond,          % postcondition    "
         Rank).             % rank rule or term  "
```

```
f_drule( DRule_id,          % decision rule identifier/name
         DParameters,
         DRule_body,        % decision rule body (Prolog stru.)
         DRule_NP).         % necessity-possibility measures
```

Environmental Knowledge:

```
f_rule( Rule_id,            % rule identifier/name
        Arguments,
        RuTe_body,          % rule body as Prolog structure
        Rule_NP).           % necessity-possibility measures

f_fact( Fact_id,            % fact identifier/name
        Arguments,
        Fact_NP).           % necessity-possibility measures
```

The correspondence of the arguments of the predicate **f_stage** to the example decision stage (presented above) is shown below:

```
f_stage( select_a_project,                  % Stage_id,
         [P,Budget],                        % Parameters,
         P,                                 % Alt_var,
         element_of,                        % Alt_type,
         proposed_project,                  % P_Predicate,
         ((promising(P) or                  % precondition
           expertise_available(P))
           and (P.cost =< Budget)),
         true,                              % Action,
         true,                              % Postcondition
         (100/P.duration +                  % Rank term
         2*P.cost))
```

PPS transforms the components of KS into the predicate instances listed above. The template definitions such as fact_def, rule_def, lc_def, etc. are also read by the predicate read using the following op declaration:

```
:- op(1150,xfx,[fact_def,rule_def,lc_def,fr_def]).
```

The definitions of database objects and relationships are identified by the predicate **db_def** and they are represented using the

following:

dbobj(**Object_id,**
Attributes).

dbrel(**Relationship_id,**
Object_id1,
Object_id2).

In this implementation **Object_id** 1 and **Object _id2** correspond to one-side and many-side of the **one-to-many** relationship. A more elaborate discussion of these can be found in (Bolloju and **Kamath** 1989b).

6.3 **Language** System

Since the language system is also a representational system like the knowledge system, the implementation of this system is similar to that of the latter. The set of ops listed in the previous section are applicable to this system as well. In addition, the following list of operator definitions are included for the implementation of fuzzy query language command input:

```
:-op(1150,fx,[activate,list,edit,dss_domain]).
:-op(1140,fx,[get,ins,del,rep]).          % query commands
:-op(1135,xfx,where).
:-op(800,xfx,with).
:-op(700,xfx,[:=,by]).
```

It can be noticed that the commands query and help are not declared above. These commands do not have any arguments and they directly invoke the corresponding procedure.

PPS reads the command sequences entered by the user using the built-in Prolog predicate **read** and the interprets the command

appropriately using the knowledge available in knowledge system. The implementation of this interpretation is elaborated in section 6.4.

Let us consider a query to retrieve title and duration of proposed projects with duration about 2 years as shown below to illustrate the usage of the above op declarations.

```
get P.title, P.duration where
        prop_proj(P) and P.duration = ~about_2_years.
```

This is a valid Prolog structure according to the ops declared above and it is same as the following structure:

```
get(where(',','.'(P,title), '.'(P,duration)),
        and(prop_proj(P),
            '='('.'(P,duration), ~(about_2_years))
        )
    )
)
```

6.4 Problem-Processing System

PPS, contrary to the other two systems, is a computational system and it processes the requests issued using *US* against the knowledge available in KS in order to generate results for the user requests. In Pl an Aid, PPS has been implemented as a number of layers as shown in Fig. 6.1.

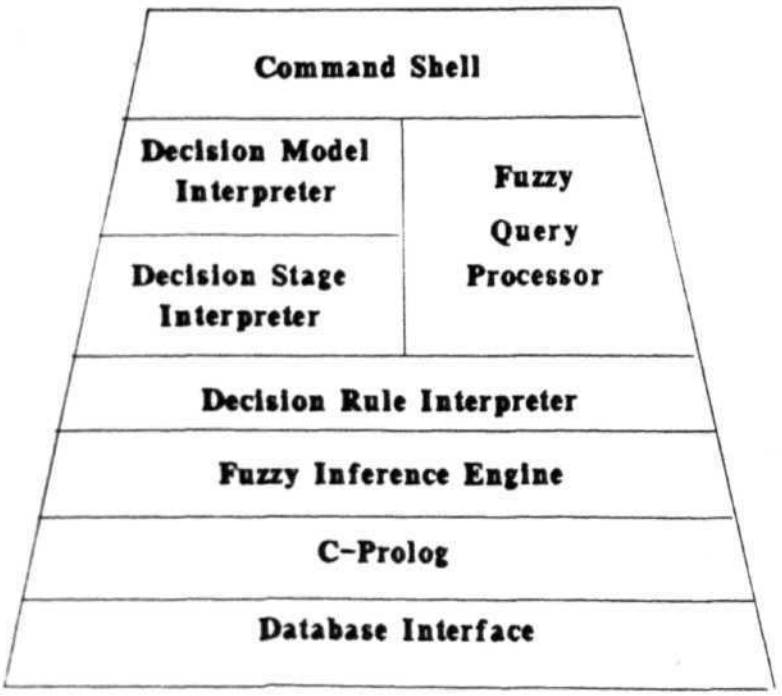


Fig. 6.1 Implementation layers of PPS

In the remaining part of this section we shall present the implementation details using Prolog clauses instead of the presentation of algorithms. We shall, however, suppress the details that are not directly relevant to the discussion.

6.4.1 Command Shell

The top most layer of PlanAid is a command shell which reads the user commands and executes the commands. The following set of clauses depict the implementation of this layer:

```
% PlanAid Shell

pashell :-  pa_init,                          % initialisations
            repeat,
                write('pa> '),
                read(Com),
                exec(Com),
                fail.                         % force backtracking

exec(Com)  :- valid(Com), call(Com).
exec(Com)  :- not valid(Com), mcssageC? Invalid command').
```

6.4.2 Decision Model Interpreter

Model activation and formulation are illustrated by the clauses listed below. The clause activate handles both the functions and uses the clause interpret_model to generate the alternatives according the decision stage definitions using the decision stage interpreter **(interpret_stage)**. Further, the clauses perform *undo* operation on any updates made in case the current alternative is *skipped*.

```
activate(Model_act)
 :-    Model_act =.. [Mdl_id|Params],
       f_model(Mdl_id,Args.Defn),              % predefined model
       sem_unif_args(Params,Args,NPargs),
       m_thresbold(NPargs),
       interpret_model(Defn).

activate(Model_form)
 :-    interpret_model(Model_form).

interpret_model((M1 -> M2))
       interpret_model(M1),
       interpret_model(M2).
```

```
interpret_model(M)
 :-    interpret_stage(M),
       inquire(Action),          % next alt |skip stage | commit
       ( Action = commit
         ;(Action = nextalt, fail)
         ;(Action = skipstg,
           undo_updates(M),
           clean_stg(M), fail)).

interpret_model(M)
 :-    clean_stg(M), fail.       % retract all housekeeping info
```

6.4.3 Decision Stage Interpreter

Decision stage interpreter **(interpret_stage)** is responsible for the generation and reporting of alternatives according to the decision stage specification. The **variable** Alt represents an alternative of the decision stage and has one of the two following structures depending upon the type of alternatives defined by the decision stage.

element type: **[Predicate_id,Argument_list]**

subset type : **[Predicate_id,Tuple_key_list]**

Argument values for the tuples identified by the members of **Tuple_key_list** are assigned by the predicate

in(V,S)

where V has the structure of an element type of alternative, and

S is **the** Alt variable of subset type.

A simplified version of the decision stage interpreter is shown below:

```
% decision stage interpreter

interpret _stage(St_head)
        St_head =.. [St_id|Params],          % split id and params
        f_stage(St_id,Args,Alt,Type,Pred,Pre,Act,Post,Rank),
                                             % get stage definition
        sem_unif_args(Params,Args,NPargs),
        s_thresbold(NPargs),                 % check applicability

        % evaluate the probable set of alternatives and
        % order this set using the heuristics
        heuristic_order(St_id,Gen_alt_goal,Alt,Defn,Ord_ids),

        select_an_alternative(St_id,Vars,Ord_ids,Rank_val),

        interpret_rule(Pre, NP_pre), pre_threshold(NP_pre),
        interpret_rule(Act,_),
        interpret_rule(Post,NP_post), post_threshold(NP_post),

        compute_rank(Rank,Rank_val),
        report_alternative(St_id,Pred,Defn,Alt,Rank_val).
```

The procedure **heuristic_order** orders the set of probable alternatives according to the following rules:

a) if an external heuristic is provided by the decision maker (beu_def) then that heuristic is used to order the tuples identified by the predicate expression (Gen_alt__goal)

b) if no external heuristic is available
   then the possibility of rank rule being used as an heuristic is analysed as shown below:
   i) if the rank rule is not present
      then there is no heuristic available
   ii) if the rank rule is a term and
      the term refers to the variables representing the unupdated facts and
      the alternatives are of element type
      then term is used as an heuristic
   iii) if the rank rule is a condition body and
      it refers to unupdated facts and
      the alternative are of element type
      then condition body is used as an heuristic.

In all other cases there is no heuristic available and the order that is used by the **select_an_alternative** is the order in which **Gen_alt_goal** generates the probable alternatives.

The rule interpreter **(interpret_rule)** is invoked through max_np to evaluate precondition, action and postcondition rules. Then, the procedure **compute_rank** evaluates the rank value according to the rank rule. Lastly, the procedure report_alternative displays the current alternative and the control gets transferred to the model interpreter for subsequent inquiry to commit or skip or display the next alternative for the current decision stage.

6.4.4 Fuzzy Query Processor

The query commands are interpreted and the results are displayed using the procedure execrq. This procedure is built around the range expression evaluater **eval_range_expr**. The goal **eval_range_expr** instantiates all the variables used in the **first** part of the query commands. For example, for a get query the variables whose argument values are to be output are instantiated. The aggregate functions, however, are evaluated using the predicate **eval** (see section 6.4.8) which in turn, recursively, uses **eval_range_expr**.

```
query :-    nl,
            repeat,
                nl, write('pa/q > '),
                rcad(Q),
                execrq(Q).
execrq(end_of_file) :- exit.          % exit on ^Z
```

```
% get command
execrq((get A where C @ NP))
 :-  execr(C,A,NP), fail.

execrq((get A where C))
 :-  default_np(NP), execr(C,A,NP), fail.

execrq((get A))
 :-  print_attr_values(A), nl, fail.
     % aggr functions are part of print_attr_values

    execr(Q,Vars,NP)
     :- nl, eval_range_expr(Q,Cnp), q_threshold(Cnp,NP),
        print_attr_values(Vars),
        print_np(Cnp),
        fail.

% ins command
execrq((ins F where C @ NP))
     insfact(F,C,NP).

cxecrq((ins F where C))
 :-  default_np(NP), insfact(F,C,NP).

execrq((ins F))
 :-  F =.. [Fid|Args],
     eval_insvals(Args,Vals),
     Fact =.. [Fid|Vals],
     asserta(Fact), fail.

    insfact(F,C,NP)
     :- eval_range_expr(C,Cnp), q_thresbold(Cnp,NP),
        execrq((ins F)).

    eval_insvals([],[]).
    eval_insvals([A|RA],[_|RV])
     :- var(A), eval_insvals(RA,RV).
    eval_insvals([A|RA],[V|RV])
     :- not var(A), eval(A,V), eval_insvals(RA,RV).

% del command

execrq((del V where C @ NP))
 :-  delfact(V,C,NP).
cxecrq((del V where C))
 :-  default_np(NP), delfact(V,C,NP).

    delfact(V,C,NP)
     :- eval_range_expr(C,Cnp), q_threshold(Cnp,NP),
        V = [Id,Args], Fact =.. [Id|Args],
        retract (Fact), fail.
```

```
% rep command
execrq((rep V with Substs where C @ NP))
 :-    repfact(V,Substs,C,NP).

execrq((rep V with Substs where C))
 :-    default_np(NP),
       repfact(V,Substs,C,NP).

       repfact([Fid,Oldvals],Substs,C,[N,P])
       :- eval_range_expr(C,Cnp), q_threshold(Cnp,NP),
          get_defn(Fid,Defn),
          compvals(Oldvals,Newvals,Substs,Defn),
          Old =.. [Fid|Oldvals], New =.. [Fid|Newvals],
          rctract(Old), asserta(New), fail.
```

6.4.5 Decision Rule Interpreter

The decision stage interpreter performs inference on fuzzy decision rules using the procedures **interpret_rule** and eval_range_expr. The result of inference through the backward chain of rules is returned as a necessity-possibility measure. It can be noticed in the first clause of the procedure **interpret_rule** that the second operand of the conjunction A **and** B is not processed when the first operand fails certainly. A similar approach can be seen in the clause corresponding to the existentially quantified rules. The clauses for rules that effect updates to the environmental knowledge base save information necessary to rollback upon backtracking. In the following list of clauses only some representative clauses are shown completely, while the others are just shown with the head of the clause.

```
interpret_rule((A and B),NP)
 :- interpret_rule(A,NP1),
    and_chk_interpret_rule(NP1,B,NP).

and_chk_interpret_rule([0,0],_,[0,0]).
and_chk_interpret_rule(NP1,B,NP) :-
    interpret_rule(B,NP2),
    and_comb(NP1,NP2,NP).
```

```
interpret_rule((A or B),NP)  :- ...

interpret_rule((not A),[N,P]):- ...

interpret_rule(exists(V:RG,Cond),NP) :-
        asserta(exists_np(0,0)),
        eval_range_expr(RG,[1,1]),
        max_np(Cond,[Cn,Cp]),
        retract(exists_np(Ln,Lp)),
        (([Cn,Cp] == [1,1], NP ~= [1,1]);
         ([Cn,Cp] \= [1,1],
          max(Ln,Cn,N), max(Lp,Cp,P),
          asserta(exists_np(N,P)), fail)).

interpret_rule(exists(_:_,_),[N,P])
    :-  retract(exists_np(N,P)).

interpret_rule(all(V:RG,Cond),NP) :- ...

interpret_rule((if Cond then T_part else E_part),NP)
        interpret_rule(Cond,NPcond),
        ( (NPcond == [1,1],!,
           interpret_rule(T_part,NP))
         ; interpret_rule(E_part,NP) ).

interpret_rule((if Cond then T_part),NP) :- ...

interpret_rule(insert(V),[1,1]) :-
        V =.. [Asrt_label|Arg_terms],
        eval_args(Arg_terms,Arg_vals),
        Ins_fact =.. [Asrt_label|Arg_vals],
        asserta(Ins_fact), % assert new insert as a fact
        current_stage(St_id), % save info to backtrack
        Rollback_info =.. [St_id,insert,Ins_fact,_],
        asserta(Rollback_info).

eval_args([],[]).
eval_args([Term|R_terms],[Val|R_vals]) :-
        eval(Term,Val), eval_args(R_terms,R_vals).

interpret_rule(insert(_),_) :-  % used on backtracking to undo
        current_stage(St_id),
        Rollback_info =.. [St_id,insert,Ins_fact,_],
        retract(Rollback_info),
        retract(Ins_fact), !, fail.

interpret_rule(delete([Label,Arg_vals]),[1,1]) :- ...

interpret_rule(replace(Old_var by New_var with Substs),[1,1]) :- ...
```

```
interpret_rule(R,NP)
      ;- R =.. [Rulc|Args], is_a_rule(Rule), !,
         f_rule(Rule,RArgs,Body,NPrulc),
         sem_unif_args(Args,RArgs,NPargs),
         rule_threshold(NPargs),
         interpret_rule(Body,NPbody),
         min_inf(NPargs,NPbody,NPimpl),
         min_comb(NPimpl,NPrule,NP).

interpret_rule(Q,NP) :- call_fie(Q,NP).      /* catch all others */

max_np(Goal,NP) :-
         asserta(maxnp(0,0)),
         interpret_rule(Goal,[Cn,Cp]),
         retract(maxnp(Ln,Lp)),
         ((Cn = 1, Cp = 1, NP = [1,1], !);
           (max(Ln,Cn,N), max(Lp,Cp,P),
            asserta(maxnp(N,P)), fail) ).

max_np(_,[N,P]) :- retract(maxnp(N,P)).

eval_range_expr(R,NP)                % for facts and objs
        :- R =.. [Fac,[Fac,Var]],
           (objdef(Fac,Defn);facdef(Fac,Defn)),!,
           getdummyvars(Defn,Var),
           G =.. [Fac|Var],
           call_fie(G,NP).

eval_range_expr(R,NP)    :- ...      % for relationships
           R =.. [Set,[Own,[Own_id|O_rec]],[Mem,[Mem_id|M_rec]]],
           reldef(Set,[Own,Mem]), !,
           Set_goal =.. [Set,Own_id,Mem_id],
           call_fie(Set_goal,NP),
           get_om_rec(Own,Own_id,O_rec),
           get_om_rec(Mem,Mem_id,M_rec).

eval_range_expr(R,NP)    :- ...      % for procedural know.
```

The following clauses combine **NP-measures** using min and max as t-norm and **co-t-norm** respectively. It is possible to extend the implementation to override these defaults.

```
min_comb([N1,P1],[N2,P2],[N,P])  :-  min(N1,N2,N), min(Pl,P2,P).

and_comb([N1,P1],[N2,P2],[N,P])  :-  min(Nl,N2,N), min(Pl,P2,P).

or_comb([N1,P1],[N2,P2],[N,P])   :-  max(Nl,N2,N), max(P1,P2,P).

min_inf(NP1,NP2,NP) :- min_comb(NP1,NP2,NP).
```

6.4.6 Fuzzy Inference Engine

Fuzzy Inference Engine **(call_fie)** performs the process of inferencing on the procedural and factual knowledge components of the environmental knowledge. In addition to the conjunction, disjunction and negation operations, this procedure performs the semantic unification, fuzzy comparison and evaluates the fuzzy expression pan of the Is predicate. The access to database objects and relationships is built into the **C-Prolog** interpreter and therefore it is transparent at this layer.

```
call_fie((A,B),NP)
 :- call_fie(A,NP1), and_chk_call_fie(NP1,B,NP).

    and_chk_call_fie([0,0],_,[0,0]).
    and_chk_call_fie(NP1,B,NP) :-
            call_fie(B,NP2),

            and_comb(NP1,NP2,NP).

call_fie((A;B),NP) :- ...

call_fie((A=B),NP) :- sem_unif(B,A,NP).

call_fie((not A), [N,P]) :- ...

call_fie(true,[1,1]).

call_fie((A i. B), [1,1]) :- eval(B,A).
call_fie((A > B),NP)      :-
            eval(A,EA), eval(B,EB), f_gt(EA,EB,NP).
call_fie((A < B).NP)      :- ...
call_fie((A > = B),NP)    :- ...
calLfie((A =< B),NP)      :- ...
call_fie((A \== B),NP)    :- ...

call_fie(FRel,NP):-   FRel =.. [Opr,A,B],
                      OprB =.. [Opr,B],
                      frdef(OprB,FVal,Defn),
                      call(Defn),
                      sem_unif(FVal,A,NP).
```

```
call_fie(G,NP):-  G =.. [Pred|Args],   % objects and facts
                  (objdef(Pred,_);facdef(Pred,_)),
                  repl_consts(Args,NArgs,Subst),
                  NG =.. [Pred|NArgs], !,
                  call(NG),
                  sem_unif_list(Subst,NP).

call_fie(G,NP) :-  G =.. [Rule|Args],
                   ruldcf(Rulc,_),!,
                   f_clause(Rule,RArgs,Body,NPrule),
                   sem_unif_args(Args,RArgs,NPargs),
                   rule_thrcshold(NPargs),
                   call_fie(Body,NPbody),
                   min_inf(NPargs,NPbody,NPimpl),
                   min_comb(NPimpl,NPrule,NP).

call_fie(G,[1,1]) :- call(G).      % catch all others
```

6.4.7 Semantic Unification

The procedure listed below unifies two variables and returns the result of unification as a pair of necessity-possibility measures.

```
sem_unif(V1,V2,[1,1])
 :- (var(V1); var(V2)),          % either V1/V2 is a var
    V1 = V2, !.

sem_unif(V1,V2,NP)
 :- eval(V1,Val1), eval(V2,Val2),
    expmf(Val1,EV1), expmf(Val2,EV2),
    nec_pos(EV1,EV2,NP).
```

% semantic unification of argument lists

```
sem_unif_args([],[],[1,1]).
sem_unif_args([A|RA],[B|RB],NP)
 :- sem_unif(A,B,NP1),
    sem_unif_args(RA,RB,NP2),
    min_comb(NP1,NP2,NP).
```

The instantiated variables are unified using the procedure nec_pos listed below. This procedure is a direct implementation of the definitions presented in tables 5.1 and 5.2. The clauses numbered 1 through 5 handle the unification of atomic (i.e. precise) patterns

against the tive different forms of datum. Let us consider clause 6 in order to elaborate the method of evaluating the necessity-possibility measures. This clause unifies a pattern of type crisp set (Csetp) with a datum of type fuzzy set (Fsetd), The necessity measure (NM) is the infimum of maximum of V1 and NegMil over all the members, with membership value Mu1, of the fuzzy set Fsetd. The goal inf finds the infimum of maximum of V1 and NegMu by successively proving the subgoals:

success(member(X,Csetp),Vl) and
NegMu is 1-Mu1.

The first subgoal success returns the success or failure of the goal specified in its first argument as 1 or 0 in its second argument. Any uninstantiated variables in the first argument get instantiated implicitly.

Similarly, the possibility measure (PM) is evaluated as the maximum of Mu2 which is the membership value of the elements of Csetp in Fsetd over all the elements of Csetp.

```
nec_pos(P,D,[NM,PM]) :-
        atomic(P),
        atomic(D),
        success(P = D,NM),
        PM is NM, !.

nec_pos(P,cset(Csetd),[NM,PM]) :- atomic(P),
    success([P]==Csetd,NM),  %P is the only element in Cset
    success(member(P,Csetd),PM), !.  % P is an element of Cset

nec_pos(P,fset(Fsetd),[NM,PM]) :- atomic(P),
        inf(get_fmember(U^Mu,Fsetd), %inf among u in Fset
            success(U = P,V1),          %v1 is pi(u) u=P
            V2 is 1-Mu, V1, V2, NM),  %v2 is l-m(u) for Fset
        fmember(P^PM,Fsetd), !.          %Poss is mu(u) for u=P
```

```
nec_pos(P,cposs([Ld,Ud]),[NM,PM]) :- atomic(P),
        success((P= Ld,P= Ud), NM), %1 if poss is 1 at p only
        cpdistr([Ld,Ud],P,PM), !.    % pi(p)

nec_pos(P,fposs(ABCDd),[NM,PM]) :- atomic(P),
        inf(fpmember(X^Mu,ABCDd),  %inf max(x=p,1-pi(x)) over
            success(X= P,Mul),      %support(D)
            Mu2 is 1-Mu,
            Mul,Mu2,NM).
        fpdistr(ABCDd,P,PM), !.


nec_pos(cset(Csetp).fset(Fsetd),[NM,PM]) :-
        inf(get_fmember(X^Mu1 ,Fsctd),
            success(member(X,Csetp),V1),
            NcgMu is 1-Mu1,
            V1,NegMu, NM),
        maxof(member(Y,Csetp),          %max of fsctd with
              fmember(Y^Mu2,Fsetd),% y over csctd
              Mu2, PM), !.

nec_pos(fset(Fsetp),fset(Fsetd),[NM,PM]) :-


nec_pos(fset(Fsetp),fposs(ABCDd),[NM,PM]) :-

nec_pos(cposs([Lp,Up]),fset(Fsetd),[NM,PM]) :-


nec_pos(fposs(ABCDp).fposs(ABCDd),[NM,PM]) :-
      pi_npi_intersect(ABCDp,ABCDd,NM).   % min inter. of P, ncg(D)
      pi_pi_intersect(ABCDp,ABCDd,PM).    % max inter. of distri.

% returns the success of a specified goal

success(G,1) :- call(G), !.
success(_,0).

% infimum : inf(RangeGoal,Goal1,Goal2,Val1,Val2,Inf)

inf(RGoal,_,_,_,_,0)
:-  nonempty(RGoal,T), T == 0, !.  % U is null

inf(RGoal,G1,G2,V1,V2,Inf) :-
    asserta(infval(1)),                 % max possible value
    call(RGoal),                        % to inst u value
    call(G1), call(G2),                 % to get v1 and v2
    max(V1,V2,Max),
    chkinf(Max,Inf)
```

```
chkinf(0,0) :-      retract(infval(_)).
chkinf(Max,_) :-    retract(infval(Lval)),
                    min(Max,Lval,Min),
                    asserta(infval(Min)),
                    fail.

inf(_,_,_,_,_,Inf) :- retract(infval(Inf)).


% supremum using the intersection of possibility distributions

pi_pi_intersect([A1,B1,C1,D1],[A2,B2,C2,D2], Int)
      :- D1 >= A2, D2 >= A1,    % supports overlap
                                i.e., not (Dl < A2 or D2 < A1
         try_intersect([A1,B1,C1,D1],[A2,B2,C2,D2],Int), !.

pi_pi_intersect(_,_,0).         % non overlapping supports

try_intersect([A1,B1,C1,D1],[A2,B2,C2,D2],Int)
      :- Cl < B2,               % core1 is less than core2
      intersect([[C1,1],[D1,0]],[[A2,0],[B2,1]],Int), !.

try_intersect([A1,B1,C1,D1],[A2,B2,C2,D2],Int)
      :- C2 < B1,               % core2 is less than core1
         intersect([[C2,1],[D2,0]],[[A1,0],[B1,1]],Int), !.

try_intersect(_,_,1).           % overlapping cores

% infimum using intersection of possibility distributions
% ABCD1 and neg (ABCD2)

pi_npi_intersect([A1,B1,C1,D1],[A2,B2,C2,D2],1)
      :- A2 >= B1, D2 =< C1, !. % support2 is in core1

pi_npi_intersect([A1,B1,C1,D1],[A2,B2,C2,D2],Int)
      :- A1 =< B2, C2 =< D1,    % non_zero intersection
         intersect([[A1,0],[B1,1]],[[A2,1],[B2,0]],Y1),
         intersect([[C1,1],[D1,0]],[[C2,0],[D2,1]],Y2),
         min(Y1,Y2,Int), !.

pi_npi_intersect(_,_,0).        % zero intersection
```

6.4.8 Fuzzy Eval

Tbe procedure eval evaluates the terms (see definition 3.11). That is, all tbe arithmetic functions, aggregate functions and argument

values of variables are evaluated by this procedure. In order to evaluate the aggregate functions the range expression evaluater is recursively used to generate the range of values over the aggregation is supposed to be performed.

```
eval((A+B),V)  :- eval(A,VA), eval(B,VB), f_add(VA,VB,V).
eval((A-B),V)  :- eval(A,VA), eval(B,VB), f_sub(VA,VB,V).
eval((A*B),V)  :- eval(A,VA), eval(B,VB), f_mult(VA,VB,V).
eval((A/B),V)  :- eval(A,VA), eval(B,VB), f_div(VA,VB,V).

eval((A.B),V)
 :- A = [Lable,Args],
    attr_no(Lable,B,N), get_arg(Args,N,MV), expmf(MV,V).

eval(max(Attr:RG),_) :-
        asserta(af_max(null)),
        eval_range_expr(RG,[N,P]),
        a_threshold([N,P]),
        af_max(Lm),                    % get last max
        eval(Attr,Cval),
        comp_gt(Lm,Cval,Nm),
        retract (af_max(Lm)),
        assert a(af_max(Nm)),
        fail

eval(max(_:_),M) :- retract(af_max(M)).

eval(min(Attr:RG),_) :- ...

eval(count(_:RG),_) :- ...

eval(sum(Attr:RG),_) :- ...

eval(A,EA) :- expmf(A,EA).  % catch all others

expmf(~(F),D) :- mfdef(F,D), !.   % expand lcdef
expmf(D,D).
```

6.4.9 Fuzzy Arithmetic

Fuzzy arithmetic operations subsume normal arithmetic operations as illustrated by the following set of clauses. Only some representative clauses for fuzzy addition are shown below. Numeric

values represented by crisp sets and fuzzy sets are approximated to continuous possibility distributions (cposs and fposs) before the addition is performed. This ensures the unimodal property in the resulting values.

```
f_add(V1,V2,V) :- atomic(V1), atomic(V2), V is V1+V2.
f_add(fposs([A1,B1,C1,D1]),fposs([A2,B2,C2,D2]),fposs([A,B,C,D]))
    :- A is A1+A2, B is B1+B2, C is C1+C2, D is D1+D2.

f_add(fposs([A1,B1,C1,D1]),cposs([L2,U2]),fposs([A,B,C,D]))
    :- A is A1+L2, B is B1+L2, C is C1+U2, D is D1+U2.

f_add(fposs([A1,B1,C1,D1]),fset(F),fposs([A,B,C,D]))
    :- f_approx(F,[A2,B2,C2,D2]),
       A is A1+A2, B is B1+B2, C is C1+C2, D is D1+D2.

f_add(fposs([A1,B1,C1,D1]),cset(C2),fposs([A,B,C,D]))
    :- c_approx(C2,[L2,U2]),
       A is A1+L2, B is B1+L2, C is C1+U2, D is D1+U2.

f_add(fposs([A1,B1,C1,D1]),V2,fposs([A,B,C,D]))
    :- A is A1+V2, B is B1+V2, C is C1+V2, D is D1+V2.

f_add(cposs([L1,U1]),fposs([A2,B2,C2,D2]),fposs([A,B,C,D]))
    :- A is L1+A2, B is L1+B2, C is U1+C2, D is U1 + D2.

f add(cposs([L1,U1]),cposs([L2,U2]),cposs([L,U]))
    :- L is L1+L2, U is U1+U2.

f_add(cposs([L1,U1]),fset(F),fposs([A,B,C,D]))
    :- f_approx(F,[A2,B2,C2,D2]),
       A is L1+A2, B is L1+B2, C is U1+C2, D is U1+D2.

f_add(cposs([L1,U1]),cset(C2),cposs([L,U]))
    :- c_approx(C2,[L2,U2]),
       L is L1+L2, U is U1+U2.

f_add(cposs([L1,U1]),V2,cposs([L,U]))
    :- L is L1+V2, U is U1+V2.

f_add(V1,fposs([A2,B2,C2,D2]), fposs([A,B,C,D]))
    :- A is V1+A2, B is V1+B2, C is V1+C2, D is V1+D2.

f_add(V1,cposs([L2,U2]),cposs([L,U]))
    :- L is V1+L2, U is V1+U2.
```

```
f_add(V1,fset(F),fset(NF)) :-
    addf_v(F,V1,NF).        % add value V1 to all members of F
f_add(V1,cset(C2),cset(NC))    :-
    addc_v(C2,Vl,NC).       % add value V1 to all members of C2
```

6.4.10 Fuzzy Comparison

The following clauses depict parts of the fuzzy comparison '>'. In this comparison, as in the case of fuzzy arithmetic, fuzzy sets and crisp sets are approximated to continuous possibility distributions represented by fposs and cposs respectively.

```
f_gt(fposs([A1,_,_,_]),fposs([_,_,_,D2]),[1,1]) :- Al  >= D2, !.
f_gt(fposs([_,___,D1]),fposs([A2,_,_,_]),[0,0]) :- A2  >= D1, !.

f_gt(fposs([A1,B1,C1,D1]),fposs([A2,B2,C2,D2]),[NM,1]) :-
            C1 > D2, !,
            pm_le(fposs([A1,B1,C1,D1]),fposs([A2,B2,C2,D2]),PM1),
            NM is 1-PM1.

fc^(fposs(ABCDl),fposs(ABCD2),[NM,PM])    :-
            pm_le(fposs([ABCD1),fposs(ABCD2),PM1), NM is 1-PM1,
            pi_pi_intersect(ABCD1,ABCD2,PM).
```

6.5 Summary

Implementation details of the prototype PlanAid are presented in this chapter. Various layers of interpreters built on top of a C-Prolog interpreter have been described and the relevant parts of the implementation are shown using Prolog clauses instead of an algorithmic notation. A C-Prolog interpreter has been extended to provide transparent access to databases of Admin (Bolloju and Kamath 1989b).

We believe that the selection of Prolog as an implementation language is justified by the effectiveness of Prolog features in the implementation of many of the proposed DSS environment facilities. An example session using this prototype is presented in Appendix B.

Chapter 7
Conclusions and Future Extensions

7.1 Conclusions

In the introductory chapter we have identified the characteristics of semi-structured decision problems at higher organisational levels. These characteristics in turn brought out certain requirements which must be satisfied by decision support environments catering to such decision problems.

The objectives aimed in this thesis are:

a) evolving a sound formalism for modelling of decision problems at higher organisational levels with facilities to represent and manipulate imprecision and uncertainty,

b) providing an integrated decision support environment facilitating the requisite tools and techniques in an unified framework, and

c) bridging the gap between the theory and practice in DSS at higher organisational levels.

A decision modelling formalism has been presented formally in the chapter 3. Based on this formalism an integrated decision support environment has been proposed and the characteristics of this environment have been described in detail in chapters 3, 4 and 5. A prototype implementation of this environment has been described in chapter 6 which demonstrates that the decision modelling formalism and the decision support environment are feasible and practical.

Implementation:

A prototype implementation of the proposed integrated decision support environment has been developed in C-Prolog on VAX-11/750. It has been found, as described in section 6.1 earlier, that Prolog has several advantages as the implementation language for the proposed environment. The user interface to this environment is, currently, a simple command oriented language and it needs to be improved using tools such as windows in order to enhance the user-friendliness of the interface.

Major emphasis laid on the implementation is to ascertain the effectiveness of the proposed decision modelling and decision support environment. Consequently, the efficiency of the prototype implementation has not been a major criteria in the development. It is possible to improve the efficiency by transferring non-symbolic computations to either efficient Prolog compiler or external language (such as C) functions. In the implementation of Problem-Processing System, the available decision models, decision stages and decision rules are searched sequentially for firing. It is possible to eliminate unnecessary search by preprocessing and maintaining groups of applicable decision models, stages and rules.

The strategy employed in processing the rules needs to be improved using more intelligent scheduling of rules based on the applicability of the rules for a given situation. Then propagation of rules used in action and postcondition parts of the specification of decision stages to preconditions and internal heuristics for

generation of alternatives can greatly improve the efficiency of the overall computations. Techniques such as the usage of integrity constraint information and meta-data (Smith 1983) can be used in the decision rule processing, particularly in the evaluation of quantified formulae and aggregate functions.

Contributions:

The prime contribution of this thesis is the decision modelling formalism with the capability to represent imprecision and uncertainty in the decision problems. This formalism facilitates hierarchical problem decomposition using the concepts of decision models, decision stages and decision rules.

Secondly, an integrated decision support environment based on the above formalism is proposed to provide the required facilities. This environment includes:

- imprecise and uncertain knowledge representation
- external database access
- high level user interface using a non-procedural language which is in tune with the decision modelling primitives,
- domain independent PPS, and
- a fuzzy query language for information retrieval.

Lastly, the decision modelling formalism and the integrated decision support environment have been implemented as a prototype system in C-Prolog on VAX-11/750. This implementation demonstrates that both the proposed formalism and the environment are feasible and

practical. This implementation contributes towards realisation of the objective of bridging the gap between the theory and the practice.

Ties to Relevant Work:

We shall now briefly discuss the methodology by which the three above mentioned contributions are realised and compare this methodology with the relevant work. The decision modelling formalism is centered around decision models, decision stages and decision rules and the representation and manipulation imprecision and uncertainty.

The imprecision and uncertainty that is prevalent in the modelling knowledge and the environmental knowledge have been represented by fuzzy sets and necessity-possibility measures in this proposal. As described in chapter 1, the fuzzy sets (or the possibility distribution) is the only method available for representation of imprecision. The fuzzy expert system shells presented in the appendix D also use the same for the representation of imprecision. While some of these systems use fuzzy sets (FRIL and System Z-II) and the others use continuous possibility distributions (SPII-1 and SPII-2).

We have integrated and extended these ideas such that imprecision can be represented using one of the following five appropriate constructs:

a) crisp sets,

b) possibility distributions with crisp boundaries,

c) fuzzy sets,

d) possibility distributions with vague boundaries, and

e) recursive definition of linguistic constants.

Some of the well-known approaches for the representation of uncertainty have been described in section 1.3.2. We believe that the handling of uncertainty using possibilistic methods is more appropriate in modelling of decision problems. This belief is also supported by (Dickinson and Ferrel 1985; Lagomasino and Sage 1985; Lebailly *et al* 1987).

A survey of fuzzy expert system shells is presented in appendix D. ARIES (Applebaum *et al* 1985) is an approximate reasoning inference engine based on propositional truth system based on interval valued logic. FLOPS (Siler *et al* 1987) uses single-valued confidences and the authors emphasise the need to replace this by two-valued confidences. On the other hand, System Z-II (Leung and Lam 1988) uses single fuzzy numbers as imprecise certainty factors and fuzzy arithmetic is used to manipulate these fuzzy certainty factors.

The representation of uncertainty using support pairs in FRIL (Baldwin 1987a, 1987b, 1988a, 1988b) is a generalisation of logic programming that combines fuzzy set theory and the theory of evidence. In SPII-1 and SPII-2 (Martin-Clouaire and Prade 1986) necessity and possibility measures are used. These measures, however, are normalised such that the possibility measure in any

necessity-possibility (NP) measure pair is equal to 1. We feel that this normalisation does not project the proper value of the result of comparison of two or more NP measures.

We provide, in this proposal, an unified method of dealing with the uncertain results of fuzzy pattern matching and the representation and propagation of uncertainty during the the process of inference.

Decision models enable decomposition of complex decision problems into many simpler decision problems. Then, decision stages are based on a problem solving approach similar to that used in STRIPS (Cohen and Feigenbaum 1982). Lastly, the basic modelling knowledge is represented using high level, non-procedural decision rules. These rules are based on the successful application of production rule systems in expert systems. However, the specification is at a much higher level to include quantification and aggregation with facilities to represent and propagate both imprecision and uncertainty. McCarthy (1987), in his Turing award lecture, points out that "… beyond that (Horn clauses of Prolog) lies full first-order logic including both existential and universal quantifiers and arbitrary first-order formulae". He, further, offers a futuristic remark "… reasoning and problem-solving programs must eventually allow the full use of quantifiers and sets

Thus, the formalism presented is generic and higher level, and thereby it is possible to model a wide range of decision problems easily and effectively.

The decision support environment proposed in this thesis integrates a number of relevant concepts from AI and ES. We have identified in the above discussion some of these concepts employed in decision modelling. Prolog-like knowledge representation scheme with extensions to represent imprecision and uncertainty using fuzzy set theory, possibility theory and necessity-possibility measures captures the environmental knowledge component. As part of this component, it is possible to access large external databases transparently. This approach demonstrates an integration of knowledge bases and databases.

Since all the problem specific knowledge is part of knowledge system component, the PPS is domain independent. Model activation can be performed by cither mere identification of the model or composing a decision model using other decision models and decision stages. The fuzzy query language provided in the environment does not distinguish between the database, factual and procedural knowledge and modelling knowledge. Through this unified high level non-procedural interface it is possible to perform exploratory information retrieval quite easily. Further, some of the decision problems could be solved by merely using the power of this interface to retrieve information based on decision rules.

It is also possible to define heuristic knowledge that can be used in the generation of alternatives by the PPS. Using heuristic knowledge the problem-processing system can generate better alternatives before considering weaker or inferior alternatives.

The environment also provides scope for user-defined t-norms and co-t-norms for combination and propagation of imprecision and uncertainty. Usage of t-norms and co-t-norms for combination and propagation of uncertainty has also been reported in (Applebaum and Ruspini 1985), (Martin-Clouaire and Prade 1986).

To summarise, one of the key characteristics of decision problems at higher organisational levels is identified as imprecision and uncertainty in the knowledge system. This thesis proposes a fuzzy rule-based decision modelling formalism for the representation of this class of decision problems. The framework proposed by Bonczek *et al* (1984) has been used only as a reference model for the presentation of various features of this proposed formalism. The implementation presented in this thesis illustrates the oprationalisation of their framework in an integrated decision support environment with the capability to represent and manipulate imprecision and uncertainty in both modelling knowledge and environmental knowledge. We strongly believe that the proposed decision support environment would enhance decision making activity in ad hoc decision problems by providing the requisite tools and techniques for rapid problem formulation and analysis of the alternative choices. We. however, need to substantiate this belief by conducting experiments with user community.

7.2 Future Extensions

Though the proposed decision modelling formalism and the decision support environment are effective in solving semi-structured decision problems, we can envisage extensions and enhancements in the three directions as discussed below:

Knowledge System:

The specification of decision rules can be made easier by incorporating pseudo-natural language features in the specification language.

Another area of improvement is in the environmental knowledge representation front. We had argued that, because of the generality in the knowledge representation scheme presented many forms of environmental knowledge can be represented. However, the usage of such knowledge is also left to the user and this could be a serious drawback to manipulate complex forms of knowledge.

Regarding the presentation of the results, we have not considered using user-definable presentation knowledge while reporting the results back to the user. This is a limitation that is consciously allowed to be present in the proposal. It is possible to provide facilities to capture and make available to PPS, this kind of knowledge.

Language System:

In the language system facilities such as explanation regarding why the reported alternatives are better and reasoning employed regarding how the system arrived at a given alternative can greatly enhance the usability of the environment. Further, the language system can be enriched with natural language like features to ease the formulation of decision models and queries by the decision maker.

Problem-Processing System:

Although the environment offers facilities to formulate decision problems by combining decision models and decision stages, the overall approach used is more of a passive nature. We can think of applying concepts such as active decision support (Raghavan and Chand 1988b), knowledge elicitation through a structured dialogue (Pearl *et al* 1982), cooperative problem solving (Fischer 1990) for assistance in this activity.

Bosman (1986) argues in favor of support for cognitive aids to the decision maker in problem processing activity. While categorising DSS as *left-brained* and *right-brained,* Young (1983) advocates the need for approaches to enhance right brained capabilities of DSS such as methods to aid creative thinking and problem solving by redefinition, restructure, etc.

The ranking of alternatives used in the prototype is *crisp* although the values themselves could be fuzzy. It is possible (and necessary

too) to rank the alternatives into groups such that all the alternatives belonging to any given group are *non-dominating* (Buckley 1985a, 1985b).

Lastly, the presentation of results is to performed by PPS using the presentation knowledge. Dos Santos and Holsapple (1989) propose the isolation of this function of PPS as a separate system called Presentation System (PS)

7.3 Summary

A fuzzy rule based decision support system environment has been proposed in this thesis. The key contributions made are listed below:

a) a fuzzy rule based decision modelling formalism,
b) an integrated decision support environment built around the above formalism, and
c) a prototype implementation to demonstrate that both the proposed formalism and the environment are feasible and practical.

We strongly believe that the proposed decision modelling formalism and the decision support environment would enhance the effectiveness in decision making activity in semi-structured decision problems at higher organisational levels.

References

Alley, H., **Bacinello** C.P. and **Hiper** K.W. (1979)
"Fuzzy set approaches to planning in the Grand River **basin"**, *Advances in Water Resources,* Vol. 2, March 1979.

Applebaum, L. and **Ruspini,** E.H. (1985)
"ARIES: An **Approximate** Reasoning Inference **Engine"** in *Approximate Reasoning in Expert Systems,* Gupta, M.M. *et al* (eds), **Elsevier** Science Publ., B.V. (N-H), 1985.

Avni, E., Kandel, A. and Gupta, M.M. (1985)
"Soft Relational Data Bases in CAD/CAM and Expert Systems" in *Approximate Reasoning in Expert Systems,* Gupta, M.M. *et al* (eds), Elsevier Science Publ., B.V. (N-H), 1985.

Badia, J. and **Martin-Clouaire,** R. (1989)
"Choice Under Imprecision: A Simple Possibility Theory-based Technique Illustrated with a Problem of Weed Identification", Tech. Rep., *Station de **Biometrie** et Intelligence **Artifielle, INRA,*** Cedex (France), 1989.

Baldwin, J.F. (1987a)
"An Uncertainty Calculus of Expert **Systems"** in *Approximate Reasoning in Intelligent Systems, Decision and **Control,*** Sanchez, E. and Zadeh, L.A. (eds), Pargamon Press, 1987.

Baldwin, J.F. (1987b)
"Evidential Support Logic Programming", *Fuzzy sets and systems* 24, pp. 1-26, 1987.

Baldwin, **J.F** (1988a)
"Fuzzy Artificial Intelligence", Tech. Rep., *University of Bristol,* England, 1988.

Baldwin, J.F. (1988b)
"Support Logic Programming", Tech. Rep., *Univ. of **Bristol,*** England, 1988.

Bhattacharjee, T.K. and Mazumdar A.K. (1989)
"FQUEL: A Query Language for a Fuzzy Relational Database System", in *Management of Data,* Naveen **Prakash** (ed), Tata-McGraw Hill, New Delhi, 1989.

Binaghi, E., Organ, D., and **Rampini,** A. (1989)
" Fuzzy Logic based Tools for **Classification** and Reasoning **with** Uncertainty", in *IEEE Intl. Workshop on Tools for **AI (TAI** 89),* IEEE CS Press, 1989.

Bolloju, N. (1989)
"Decision Modelling in PlanAid: A Fuzzy Rule-based DSS Shell", in *Management of Data*, Naveen Prakash (ed), Tata-McGraw Hill, New Delhi, 1989.

Bolloju, N. and Kamath, M.L. (1989)
"Prolog Interface to a Network DBMS", in *Management of Data*, Naveen Prakash (ed), Tata-McGraw Hill, New Delhi, 1989.

Bolloju, N. (1990a)
"Modelling of Imprecise and Uncertain Information", in *Current Trends in Management of Data*, Naveen Prakash (ed), Tata-McGraw Hill, New Delhi, 1990.

Bolloju, N. (1990b)
"Fuzzy Query Language in PlanAid", *CMC Tech. Rep. 90/6*, Secunderabad, India, 1990.

Bolloju, N. (1991a)
"Decision Modelling Formalism in a Fuzzy Rule-based DSS environment", *CMC Tech. Rep. 91/1*, Secunderabad, India, 1991. *(Communicated)*

Bolloju, N. (1991a)
"An Approximate Query Language "*CMC Tech. Rep. 91/2*, Secunderabad, India, 1991. *(Communicated)*

Blanning, R.W. (1985)
"A Relational Framework for Join Implementation in Model Management System", *Decision Support Systems*, Vol. 1, No. 1, 1985.

Blanning, R.W. (1986)
"A Relational Framework for Information Management", in *Decision Support Systems: A Decade in Perspective*, E.R. McLean and H.G. Sol (Eds.), Elsevier Science Publ., B.V. (N-H), 1986.

Bonczek, R.H., Holsapple, C.W. and Whinston, A.B. (1983)
"Specification of Modeling Knowledge in Decision Support Systems", in *Processes and Tools for Decision Support*, H. G. Sol (Ed), N-H Publ. Comp., IFIP, 1983.

Bonczek, R.H., Holsapple, C.W. and Whinston, A.B. (1984)
"Developments in Decision Support Systems*, in *Advances in Computers*, Vol. 23, Academic Press, 1984.

Borch, O.J. and Hartvigsen, G. (1990)
"STRATEX - A Knowledge-based System for Strategic Market Planning in Small Firms , *A1C0M*, Vol. 3, No. 1, 1990.

Bosnian, A. (1983)
" Decision Support Systems, Problem Processing and Coordination", in *Processes and Tools for Decision Support*, H. G. Sol (Ed), N-H Publ. Comp., IFIP, 1983.

Bosman, A. and Sol, H.G. (1985)
"Knowledge Representation and Information Systems Design", in *Knowledge Representation for Decision Support Systems*, Methlie, L.B. and Sprague, R.H. (eds), Elsevier Science Publ., N-H, 1985.

Bosman, A. (1986)
"Relations between Specific Decision Support Systems", in *Decision Support Systems: A Decade in Perspective*, E.R. McLean and H.G. Sol (Eds.), Elsevier Science Publ., B.V. (N-H), 1986.

Buchanan, B.G. and Sbortliffe, E.H. (1984)
Rule-Based Expert Systems, Addison Wesley, Reading, Mass., 1984.

Buckley, J.J. (1985a)
"Ranking Alternatives using Fuzzy Numbers", *Fuzzy sets and systems*, Vol. 15, pp. 21-31, 1985.

Buckley, J.J. (1985b)
"Fuzzy Hierarchical Analysis, in Fuzzy sets and systems, Vol. 17, pp. 233-247, 1985.

Buckley, J.J. (1988)
"Managing Uncertainty in a Fuzzy Expert System", *Intl. J. Man-Mach. St.*, Vol. 29, pp. 129-148, 1988.

Charniak, E. and McDennott, D. (1985)
Introduction to Artificial Intelligence, Addison-Wesley, Reading, Mass., 1985.

Clocksin, W.F. and Mellish, C.S. (1981)
Programming in Prolog, Springer-Verlag, New York, 1981.

Coelho, H. and Rodrigues, A.J. (1985)
"Knowledge Architecture for Management Environments", in *Knowledge Representation for Decision Support Systems*, Methlie, L.B. and Sprague, R.H. (eds), Elsevier Science Publ., N-H, 1985.

Cohen, P.R. and Fiegenbaum, E.A. (1982)
The Handbook of Artificial Intelligence, Vol. 3, Kaufmann, Los Altos, Cal., 1982.

Cohen, P.R. (1985)
Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach, Pitman Publ., London, 1985.

Date, C.J. (1981)
An Introduction to Database Systems (3rd ed.), Addison-Wesley, Reading, Mass., 1981.

Dickinson, D. and Ferrell, W.R. (1985)
" Fuzzy Set Knowledge Representation in a System to Recommend Management Decisions", in *Knowledge Representation for Decision Support Systems,* Methlie, L.B. and Sprague, R.H. (eds), Elsevier Science Publ., N-H, 1985.

Dos Stantos, B.L. and Holsapple, C.W. (1989)
"A Framework for Designing Adaptive DSS Interfaces", DSS Vol. 5, No. 1, pp. 1-11, 1989.

Dutta, A. and Basu, A. (1984)
"An Artificial Intelligence Approach to Model Management in Decision Support Systems", *IEEE Computer,* Sept 1984.

Dutta, A. and Basu, A. (1985)
"Representation and Manipulation of Modelling Knowledge in Decision Support Systems", in *Knowledge Representation for Decision Support Systems,* Methlie, L.B. and Sprague, R.H. (eds), Elsevier Science Publ., N-H, 1985.

Dutta, A. (1987)
"The Explicit Support of Human Reasoning in Decision Support Systems", in *Advances in Computers,* Vol. 26, pp. 1-45, Academic Press, 1987.

Fischer, G. (1990)
"Communication Requirements for Cooperative Problem Solving Systems", *Info. Sys.*, Vol. 15, No. 1, 1990.

Fox, M.S. (1985)
"Knowledge Representation for Decision Support" in *Knowledge Representation for Decision Support Systems,* Methlie, L.B. and Sprague, R.H. (eds), Elsevier Science Publ., N-H, 1985.

Gray, P. (1984)
Logic, Algebra and Databases, Ellis Horwood Ltd., UK 1984.

Green-Hall, N. (1987)
"A Fuzzy Decision Support System for Strategic Planning" in *Approximate Reasoning in Intelligent Systems, Decision and Control,* Sanchez, E. and Zadeh, L.A. (eds), Pargamon Press, 1987.

Henkind, S. J. and **Harrison, M.C.** (1988)
"An Analysis of Four Uncertainty Calculi", *IEEE SMC,* Vol. 18, No. 5, pp. 700-714, 1988.

Holtzman, S. and **Breese,** J. (1986)
"Exact Reasoning about Uncertainty: On the Design of Expert Systems for Decision Support", in *Uncertainty in Artificial Intelligence,* **Kanal,** L.N. and **Lemmer** J.F.(Eds), Elsevier Science **Publ.,** B.V. (N-H), 1986.

Hopple, G.W. (1988)
The State of the Art in Decision Support Systems, QED Info. Sci. Inc., Wellesley, Mass., 1988.

Jackson, P. (1986)
Introduction to Expert Systems, **Addison-Wesley,** 1986.

Keen, P.G.W. (1986)
"Decision Support Systems: The next Decade", in *Decision Support Systems: A Decade in Perspective,* E.R. McLean and H.G. Sol (Eds.), Elsevier Science Publ., B.V. (N-H), 1986.

Lagomasino, A. and Sage, **A.P.** (1985)
"Imprecise Knowledge Representation in Inferential Activities", **in** *Approximate Reasoning in Expert Systems,* Gupta, M.M. *et al* (eds), Elsevier Science Publ., B.V. (N-H), 1985.

Lebailly, J., **Martin-Clouaire,** R. and **Prade,** H. (1987)
"Use of Fuzzy Logic in a Rule-based System in Petroleum Geology", in *Approximate Reasoning in Intelligent Systems, Decision and Control,* Sanchez, E. and **Zadeh,** L.A. (eds), Pargamon Press, 1987.

Leung, K.S. and Lam, W. (1988)
"Fuzzy Concepts in Expert Systems", *IEEE Computer,* Sept 1988.

Martin-Clouaire, R. and Prade, H. (1985)
"On the Problems of Representation and Propagation of Uncertainty in **Expert Systems",** *Intl. J. Man-Machine St.,* Vol. 22, pp. 251-264, 1985.

Martin-Clouaire, R. and Prade, H. (1986)
"SPII-1: A Simple Inference Engine Capable of Accommodating both Imprecision and Uncertainly, in *Computer-Assisted Decision Making,* G.Mitra (ed.), North-Holland, 1986.

Martin-Clouaire, R. (1989a)
"Semantics and Computation of Generalized Modus ponens: the long paper", *Intl. J. of Appx. Reasoning,* Vol. 3, 1989.

Martin-Clouaire, R. (1989b)
"**Use** of Possibility Theory in Real Reasoning **Systems",** ***IFSA-89,*** Seattle, 1989.

McCarthy, J. (1987)
"**Generality** in Artificial Intelligence (Turing Award lecture)", ***Comm. of ACM,*** Vol. 30, No. 12, 1987.

McLean, E.R. and Sol, H.G. (1986)
Decision Support Systems: A Decade in Perspective, in *Decision Support Systems: A Decade in* ***Perspective,*** E.R. McLean and H.G. Sol (Eds.), Elsevier Science **Publ.,** B.V. (N-H), 1986.

Naveen **Prakash,** Parimala, N. and Bolloju, N. (1983)
"Data Definition Facilities in Admin", *Comp. J.,* Vol 26, No. 4, pp. 329-335, 1983.

Nilsson, NJ. (1980)
Principles of Artificial Intelligence, Tioga, Palo Alto, **Cal.,** 1980.

Parimala, N. *et al* (1989)
"A Query Facility to a Network DBMS", *Comp. J.,* Vol. **32,** No. **1,** pp. 55-62, 1989.

Pearl, J., Leal, A. and **Saleh,** J. (1982)
"**GODDESS**: A Goal-Directed Decision Structuring System", *IEEE Transactions on* ***PAMI,*** Vol. **PAMI-4,** No. 3, 1982.

Pfeifer, R. and **Lüthi,** H.J. (1987)
"Decision Support Systems and **Expert** Systems: A **Complementaty** Relationship?", in *Expert Systems and Artificial Intelligence in Decision Support* ***Systems,*** H.G. Sol *et al* (Eds.), D. Reidel Publ. Co. 1987.

Pradc, H. and **Testemale** (1987)
"**Application** of Possibility and Necessity measures to Documentary Information Retrieval", in *Uncertainty in Knowledge-Based Systems,* B.Bouchon and R.R. Yager (Eds.), Spnnger-Verlag, 1987.

Pradc, H **(1985a)**
"A Quantitative Approach to Approximate Reasoning in Rule-based Expert Systems*, in *Expert System* ***Applications,*** **Bolc,** L. and Coombs M.J. (eds), **Springer-Verlag,** 1985.

Pradc, H (1985b)
"**A** Computational Approach to Approximate Reasoning and Plausible Reasoning with Applications to Expert Systems', *IEEE Transactions on* ***PAMI,*** Vol. **PAMI-7,** No. 3, 1985.

Raghavan, S.A. and Chand, D.R. (1988a)
"A Perspective on Decision Support Systems", Tech. Rep., Bentley College, Waltham, MA, 1988.

Raghavan, S.A. and Chand, D.R. (1988b)
"Exploring Active Decision Support: The JANUS Project", Tech. Rep., *CIS Dept., Bentley College,* Waltham, MA, 1988.

Raju, K.V.S.V.N. and Mazumdar, A.K. (1988)
"Functional Dependencies and Lossless Join Decomposition of Fuzzy Relational Database System", *ACM Tr. on Database Sys.,* Vol. 13, No. 2, pp. 129-166, 1988.

Rich, E. (1983)
Artificial Intelligence, McGraw-Hill, 1983.

Rundensteiner, E.H. and Bic, L. (1989)
"Semantic Data Models and their Potential for Capturing Imprecision", in *Management of Data,* Naveen Prakash (ed), Tata-McGraw Hill, New Delhi, 1989.

Siler, W., Buckley, J.J. and Tucker, D. (1987)
"Functional Requirements for a Fuzzy Expert System Shell", in *Approximate Reasoning in Intelligent Systems, Decision and Control,* Sanchez, E. and Zadeh, L.A. (eds), Pargamon Press, 1987.

Smith, D.E. (1983)
"Finding All of the Solutions to a Problem", *AAAI-83,* pp. 373-377, 1983.

Sol, H.G. (1983)
"Processes and Tools for Decision Support: Inferences for Future Developments", in *Processes and Tools for Decision Support,* H. G. Sol (Ed), N-H Publ. Comp., IFIP, 1983.

Sprague. R.H. (1980)
"A Framework for the Development of Decision Support Systems", *MIS Quarterly,* Vol. 4, No. 4, 1980.

Teng, J.T.C., Mirani, R. and Sinha, A (1988)
"A Unified Architecture for Intelligent DSS" in TH0213-9/88, *IEEE,* 1988.

Turban, E. and Watkins. P.R (1985)
"Integrating Expert Systems and Decision Support Systems", in *Transactions of the Fifth Intl. Conf. on DSS (DSS '85),* 1985.

Ullman, J.D. (1984)
Principles of Database Systems, (2nd ed.), **Computer** Science Press, Maryland, **1984.**

Widmeyer, G.R. and Lee, R.M. **(1986)**
"Preference Elicitation in Decision Aiding: Application to Electronic **Shopping",** in *Decision Support Systems: A Decade in Perspective,* E.R. McLean and H.G. Sol (Eds.), **Elsevier** Science **Publ.,** B.V. (N-H), 1986.

Young, L.F. (1983)
"Computer Support for Creative Decision Making: Right Brained DSS", in *Processes and Tools for Decision* ***Support,*** H. G. Sol (Ed), N-H Publ. **Comp., IFIP,** 1983.

Zadeh, L.A. (1978)
"Fuzzy sets as a basis for a thoery of possibility", *Fuzzy sets and systems,* Vol. 1, pp. 3-28, 1978.

Zadeh, L.A. (1985)
"The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems" in *Approximate Reasoning in Expert Systems,* Gupta, M.M. *et al* (eds), Elsevier Science Publ., B.V. (N-H), 1985.

Zamankova-Leech, M. and **Kandel,** A. (1985)
"Uncertainty Propagation to Expert Systems" in *Approximate Reasoning in Expert Systems,* Gupta, M.M. *et al* (eds), Elscvier Science Publ., B.V. (N-H), 1985.

Bibliography

Alter, S.
Decision Support Systems: Current Practice and Continuing Challenges, Addison-Wesley, Reading, Mass., 1980.

Bonczek, R.H., Holsapplc, C.W. and Whinston, A.B.
Foundations of Decision Support Systems, Academic Press, NY, 1981.

Bouchon, B. and Yager, R.R.(eds)
Uncertainty in Knowledge-Based Systems, Springer-Verlag, 1987.

Buchanan, B.G. and Shortliffe, E.H.
Rule-Based Expert Systems, Addison-Wesley, Reading, Mass., 1984.

Charniak, E. and McDcrmott, D.
Introduction to Artificial Intelligence, Addison-Wesley, Reading, Mass., 1985.

Clocksin, W.F. and Mcllish, C.S.
Programming in Prolog, Springer-Verlag, New York, 1981.

Cohen, P.R. and Fiegenbaum, E.A.
The Handbook of Artificial Intelligence, Vol. 3, Kaufmann, Los Altos, Cal., 1982.

Cohen, P.R.
Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach, Pitman Publ., London, 1985.

Date, C.J.
An Introduction to Database Systems (3rd ed.), Addison-Wesley, Reading, Mass., 1981.

Dubois, D. and Prade, H.
Fuzzy Sets and Systems: Theory and Applications, Academic Press, New York, 1980.

Gray, P.
Logic, Algebra and Databases, Ellis Horwood Ltd., UK, 1984.

Gupta, M.M. *et al* (eds)
Approximate Reasoning in Expert Systems, Gupta, M.M. *et al* (eds), Elsevier Science Publ., B.V. (N-H), 1985.

Hopple, G.W.
The State of the Art in Decision Support Systems OED Info. Sci. Inc., Wellesley, Mass., 1988.

Jackson, P.
Introduction to Expert Systems, Addison-Wesley, 1986.

Kanal, L.N. and Lemmer, J.F.(Eds)
Uncertainty in Artificial Intelligence, Elsevier Science Publ., B.V. (N-H), 1986.

Keen, P.G.W. and Scott Morton, M.S.
Decision Support Systems: An Organizational Perspective, Addison-Wesley, Reading, Mass., 1978.

Kowalski, R.
Logic for Problem Solving, North-Holland, 1979.

Lloyd, J.W.
Foundations of Logic Programming, Heidelberg: Springer-Verlag, 1984.

McLean, E.R. and Sol, H.G. (eds)
Decision Support Systems: A Decade in Perspective, Elsevier Science Publ., B.V. (N-H), 1986.

Methlie, L.B. and Sprague, R.H. (eds)
Knowledge Representation for Decision Support Systems, Elsevicr Science Publ., N-H, 1985.

Mitra, G. (ed)
Computer-Assisted Decision Making, North-Holland, 1986.

Negoita, C.V.
Fuzzy Systems and Expert Systems, Benjamin Cummings Press, Menlo Park, CA, 1985.

Nilsson, NJ.
Principles of Artificial Intelligence, Tioga, Palo Alto, Cal., 1980.

Raghavan, S.A. and Chand, D.R.
"A Perspective on Decision Support Systems", Tech. Rep., *Bentley College,* Waltham, MA, 1988.

Rich, E.
Artificial intelligence, McGraw-Hill, 1983.

Sanchez, E. and Zadeh, L.A. (eds)
Approximate Reasoning in Intelligent Systems, Decision and Control, Pargamon Press, 1987.

Shafer, G.
A Mathematical Theory of Evidence, Princeton Univ. Press, Princeton, NJ, 1976.

Sol, H.G. (ed)
Processes and Tools for Decision Support, N-H Publ. Comp., IFIP, 1983.

Sol, H.G. *et al* (eds)
Expert Systems and Artificial Intelligence in Decision Support Systems, D. Reidel Publ. Co. 1987.

Spraguc, R.H. and Carlson, E.D.
Building Effective Decision Support Systems, Prentice-Hall, Englcwood Cliffs, 1982.

Spraguc, R.H. and Watson, H.J. (eds)
Decision Support Systems: Putting Theory into Practice Prentice-Hall Intl., 1986.

Ullman, J.D.
Principles of Database Systems, (2nd ed.), Computer Science Press, Maryland, 1984.

Appendix A
Syntax

In this appendix we shall present the syntax of the various components of knowledge system and language system in Extended BNF

Knowledge System:

ModellingKnowledge = {*DecisionModel*}
{*DecisionStage*}
{*DecisionRule*} .

DecisionModel = *DecisionModelHead* <- *DecisionModelBody* . .

DecisionModelHead = *DecisionModelName* [(*Arguments*)] .

Arguments = *Argument* [, *Arguments*] .

DecisionModelBody =
DecisionModelComp1 [-> *DecisionModelBody*] .

DecisionModelComp1 =
DecisionModelComp2 [|| *DecisionModelComp1*] .

DecisionModelComp2 =
(if *CondRuleBody*
then *DecisionModelBody* [else *DecisionModelBody*])
| (foreach *Variable* of *RangeExpression*
do *DecisionModelBody*)
| (while *CondRuleBody* do *DecisionModelBody*)
| *DecisionStageHead*
| ((*DecisionModelBody*)) .

```
DecisionStage = DecisionStageHead <- DecisionStageBody .

DecisionStageHead = DecisionStageName [ (Arguments) ] .

DecisionStageBody = select AltVariable of AltSpecification
                        [ using StageSpec1 ] .

AltVariable = Variable | subset Variable .

AltSpecification = PredicateName .

StageSpec1 = ( precond CondRuleBody [ , StageSpec2 ] )
           | StageSpec2 .

StageSpec2 = ( action ActionRuleBody [ , StageSpec3 ] )
           | StageSpec3 .

StageSpec3 = ( postcond CondRuleBody [ , StageSpec4 ] )
           | StageSpec4 .

StageSpec4 = [ rank RankRuleBody ] .


DecisionRule = DecisionRuleHead <- DecisionRuleBody .

DecisionRuleHead = DecisionRuleName [ ( Arguments ) ] .

DecisionRuleBody = ( CondRuleBody [ @ NecPosMeasure ] )
                 | ActionRuleBody
                 | RankRuleBody .

NecPosMeasure = [ NecessityDegree , PossibilityDegree ] .

CondRuleBody = CondRuleDisj [ and CondRuleBody ] .

CondRuleDisj = CondRulePred [ or CondRuleDisj ] .

CondRulePred = PredicateName(Arguments)
             | ( Expression RelationalOperator Expression )
             | ( Quantifier(Variable:RangeExpression,
                            CondRuleBody) )
             | ( CondRuleBody )
             | ( not ( CondRuleBody ) ) .
```

```
PredicateName = FactName
              | RuleName
              | ObjectName
              | RelationshipName
              | in .

RelationalOperator =  PredefinedRelOpr
                    | UserDefinedRelOpr .

PredefinedRelOpr = = | \= | > | >= | < | =< .

Quantifier =  exists | all
            | ( [Modifier] FuzzyQuantifier ) .

Expression = Term [( + | - ) Expression ] .

Term = Factor [ ( * | / ) Term ] .

Factor = Constant
       | Variable.AttrName
       | FunctionName(Arguments)
       | AggregateFunctionName(Variable:RangeExpression)
       | AggregateFunctionName(Term:RangeExpression)
       | ( Expression ) .

RangeExpression = PredicateName(Variable)
                  [ and ( CondRuleBody | RangeExpression ) ] .

ActionRuleBody = ActionRuleComp [ and ActionRuleBody ] .

ActionRuleComp =  ActionRuleSimp
                | ( foreach Variable of RangeExpression
                    do ActionRuleComp )
                | ( if CondRuleBody
                    then ActionRuleComp
                    [else ActionRuleComp ] ) .

ActionRuleSimp =
    insert(GroundPredName(AttrValueList))
  | delete(Variable)
  | replace(Variable1 by Variable2, (AttrValueAssignList) .

GroundPredName = FactName
               | ObjectName .

AttrValueList = AttrName : Term [ , AttrValueList ] .

AttrValueAssignList = AttrName := Term
                      [, AttrValueAssignList ] .

RankRuleBody = CondRuleBody | Term .
```

EnvironmentalKnowledge = {*Rule*}
{*Fact*}
{*MetaDefinitions*}

Rule = *RuleHead* :- *RuleBody* [@ *NecPosMeasure*] . .

RuleHead = *RuleName* [(*Arguments*)] .

RuleBody = *RuleDisj* [, *RuleBody*] .

RuleDisj = *RulePred* [; *RuleDisj*] .

RulePred = *PredicateName*(*AttrIdentValues*)
| *PrologPredicate*(*Arguments*)
| (not (*RuleBody*)) .

AttrIdentValues = [*AttrName* :] (*Constant* | *Variable*)
[, *AttrIdentValues*] .

Fact = *FactName*(*AttrIdentValues*) [@ *NecPosMeasure*] . .

MetaDefinitions = {*FactDef*}
{*RuleDef*}
{*LinguisticConstDef*}
{*FuzzyRelationDef*}
{*HeuristicDef*}
{*DatabaseDef*} .

FactDef = *FactName* fact_def (*Attrnames*). .

RuleDef = *RuleName* rule_def (*Attrnames*). .

LinguisticConstDef = *LingConstName* lc_def
(*ImpreciseConstant* | *LingConstSpec*) .

LingConstSpec = *LingConstName* [(and | or) *LingConstSpec*] .

FuzzyRelationDef = *FuzzyRelationName* fr_def(*LeftOperand*,
RightOperand,
OperProcedure). .

```
HeuristicDef = DecisionStage
               heu_def( heu_var:Variable,
                        heu_val:Term,
                        subset_size:[Integer1,Integer2],
                        domain_size:Integer). .

DatabaseDef = db_def( schema:SchemaName,
                      subschema:SubschemaName). .
```

Language System:

```
LanguageSystemCommands = ActivateCommmand
                       | query
                       | list (ekb | mkb)
                       | load (ekb | mkb)
                       | edit (ekb | mkb)
                       | help
                       | dssdom DSSApplicationName
                       | listdom
                       | exit .

ActivateCommand = activate DecisionModelBody . .

RetQuery= get TermList [ where RangeExpression
                                   [ @ Threshold] ] . .

TermList = Term [, TermList] .

InsQuery = ins FactName(AttrValueList)
               [where RangeExpression [ @ Threshold ]] . .

DelQuery = del Variable
           where RangeExpression [ @ Threshold ] . .

RepQuery = rep Variable with (AttrValueAssignList)
           where RangeExpression [ @ Threshold ] . .

Argument = Variable | Constant

Constant = PreciseConstant
         | ImpreciseConstant .

PreciseConstant = AtomicValue .

ImpreciseConstant = ~LinguisticConstantIdent
                  | cset( [ AtomicValues ])
                  | fset( [ AtomicValueMembershipPairs ])
                  | cposs([ LowerBound , UpperBound ])
                  | fposs([ ValueA,ValueB,ValueC,ValueD]) .
```

AtomicValues = *AtomicValue* [, *AtomicValues*] .

AtomicValueMemebershipPairs = *AtomicValue*^*MembershipValue*
[, *AtomicValueMemebershipPairs*] .

DecisionModelName = *Identifier* .
DecisionStageName = *Identifier* .
DecisionRuleName = *Identifier* .
FactName = *Identifier* .
RuleName = *Identifier* .
ObjectName = *Identifier* .
RelationshipName = *Identifier* .
AttrName = *Identifier* .
LignConstName = *Identifier* .
FuzzyRelationName = *Identifier* .
SchemaName = *Identifier* .
SubschemaName = *Identifier* .
DSSAplicationName = *Identifier* .

UserDefinedRelOpr = *Identifier* .
Modifier = *Identifier* .
FuzzyQuantifier = *Identifier* .
OperProcedure = *Identifier* .

PrologPredicate = ...

Variable1 = *Variable* .
Variable2 = *Variable* .
LeftOperand = *Variable* .
RightOperand = *Variable* .

Threshold = *NecPosMeasure* .
NecessityDegree = *Number* .
PossibilityDegree = *Number* .

Integer1 = *Integer*.
Integer2 = *Integer*.

Variable = (_ | *UpperCaseLetter*) { *Letter* } .

Identifier = *LowerCaseLetter* { (_ | *Letter*) } .

AtomicValue = *Idenfier* | *Number* .

Number = *Integer* | *Real* .

Integer = (+ | -) *Digit* {*Digit* } .

Real = *Integer* . *Digit* {*Digit*} .

Appendix B
An Example Session on PlanAid

In this appendix we shall present an example session using the prototype implementation PlanAid. All the commands entered by user are shown in *italics*. Boxed Annotations have been added to improve the readability of the session.

The command *pashell* invokes a PlanAid session as shown below:

```
$ pashell

pa> help        ---- lists all the pashell commands with a brief description

Command    -  Description
-------------------------------------------------------
dssdom D       changes the decision problem domain
listdom        lists all the decision problem domains
load ekb       loads environmental knowledge
load mkb       loads modelling knowledge
edit ekb       edits environmental knowledge
edit mkb       edits modelling knowledge
list ekb       lists environmental knowledge
list mkb       lists modelling knowledge
activate F     activates a predefined decision model or
                    a formulated decision model
explore        exploratory information retrieval
exit           exits from PlanAid shell

pa> listdom

Applications under PlanAid:
---------------------------
1.  proj_selection

pa> dssdom proj_selection   ---- -A new identifier initiates a new dss domain.
                                 -Mkb and Ekb for new domain can be entered using the edit commands. Edit command invokes the system editor.
```

The domain proj_selection is active now.

pa> *load ekb*

Ekb of the domain *proj_selection* get loaded. And now it possible to perform information retrieval

Environmental knowledge is being loaded ...

pa> *load mkb*

Modelling knowledge gets loaded. After this, it is possible to activate decision models or query using decision rules

Model base is being loaded ...
Stage base is being loaded ...
Rule base is being loaded ...

pa> *list ekb*

% template definitions

comments are specified with '%' symbol at the beginning of the comments; the comment ends at the end of the line

```
prop_proj           fact_def   (pid,title,duration,manpower,
                                 type,area,complexity).
promising_area      fact_def   (aid,area).

proposed_project    rule_def   (pid,title,duration,type,area,
                                 complexity,cost).
probable_manager    rule_def   (no,name,qual,experience,age,
                                 leadership_apt).

select_projects     heu_def    (heu_var:PJ,heu_val:PJ.cost,
                                 subset_size:[2,3],domain_size:4).
```

Heuristic to be used during the activation of decision stage *select_projects*. This indicates that the probable alternatives are generated using projects ordered in descending order on *cost* and only the top *4 (domain_size)* projects are to be used. Further, the alternatives are subsets of the cardinality 2 to 3 *(subset_size)*.

The definition of such heuristics enable the decision maker to consider heuristically better alternatives before inferior ones. We are not aware of any other system providing such a facility.

```
db_def(schema:personnel,subschema:projemp).
```

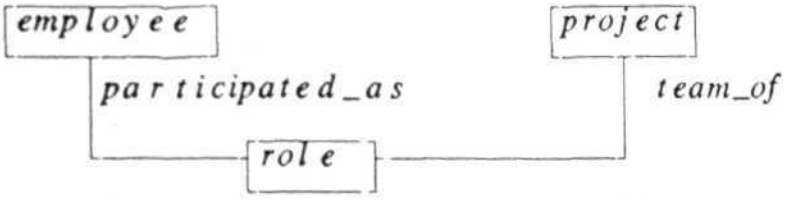
The following object and relationship definitions are automatically included as a result of the above *db_def*

Most of the existing systems in this area offer only in-core databases (logic based). There are some systems, (not under DSS), which offer interface to relational databases through languages like Prolog.

This feature demonstrates the integration of knowledge bases and databases (network model based) with transparent data access.

```
employee    obj_def (no, name,qual,experience,
                     age, leadership_apt).
project     obj_def (pid, title, area,status).
role        obj_def (rid, role).

participated_as    rel_def (employee,role).
team_of            rel_def (project,role).
```



The above schema diagram depicts the objects are the relationships between the objects

```
% linguistic constant definitions
% manpower
about_5                 lc_def    fposs([3,5,5,7]).
about_4                 lc_def    fposs([2,4,4,6]).
two_to_three            lc_def    cposs([2,3]).
around_4                lc_def    fset([2^0.5,3^0.8,4^1,
                                        5^0.8,6^0.5]).
either_3_or_5_or_6      lc_def    cset([3,5,6]).

% project complexity
quite_complex           lc_def    fposs([0.8,0.9,1,1]).
moderately_complex      lc_def    fposs([0.6,0.8,0.8,0.9]).
complex                 lc_def    fposs([0.5,0.7,0.8,0.9]).
not_so_complex          lc_def    cposs([0.3,0.5]).
```

```
% project duration (in months)
about_2_years          lc_def    fposs([18,24,24,30]).
above_2_years          lc_def    fposs([21,24,30,36]).
about_18_months        lc_def    fposs([15,18,18,21]).

% leadership aptitude
excellent              lc_def    fposs([60,90,100,100]).
very_good              lc_def    fposs([60,70,90,100]).
good                   lc_def    fposs([50,60,70,80]).
above_average          lc_def    fposs([50,60,60,70]).
average                lc_def    fposs([40,50,50,60]).
above_good             lc_def    (~excellent or ~very_good).

% experience (in years)
fairly_senior          lc_def    fposs([5,8,15,15]).
senior                 lc_def    fposs([3,5,7,10]).
about_6y               lc_def    fposs([5,6,6,7]).
about_9y               lc_def    fposs([8,9,10,10]).
below_4y               lc_def    fposs([2,4,4,4.2]).

% age in years
about_30y              lc_def    fposs([28,29,31,32]).
about_32y              lc_def    fposs([30,31,33,34]).
about_33y              lc_def    fposs([31,32,34,35]).
young                  lc_def    fposs([0,20,25,28]).

% cost in millions of Rs.
expensive              lc_def    fposs([1,7,10,20]).
moderately_expensive   lc_def    fposs([0,2,5,15]).

% number of projects being handled
heavily_loaded         lc_def    fposs([2,3,5,5]).
```

These linguistic constant definitions provide five types of definitions where as the other systems normally provide either fuzzy sets or possibility distributions.

```
% environmental knowledge: factual knowledge

% proposed projects

prop_proj(pid:p1,
          title:dist_dbms,
          manpower:~about_5,
          duration:cposs([36,48]),
          type:developmental,
          complexity:~quite_complex,
          area:database).
```

Factual knowledge can be specified either with names of arguments or according to the order specified in fac_def

```
prop_proj(p2,dss_generator, ~ about_2_years,3,research,
          decision_support, ~ moderately_complex).
prop_proj(p3,tp_package,36, ~ about_4,developmental,
          operating_systems, ~ complex).
prop_proj(p4,prolog_compiler, ~ above_2_years, ~ about_5,
          research,programming_languages, ~ complex).
prop_proj(p5,gims,12,15,turnkey,
          games_management, ~ not_so_complex).
prop_proj(p6,expsys_shell, ~ about_18_months,3,research,
          artificial_intelligence, ~ complex).

% promising areas
promising_area(a1,four_GL).
promising_area(a2,database) @ [0.9,1].
promising_area(a3,networks).

promising_area(a4,programming_languages) @ [0.8,1].
promising_area(a5,operating_systems) @ [0.7,1].
promising_area(a6,artificial_intelligence).
promising_area(a7,decision_support).
promising_area(a8,games_management) @ [0.6,0.9].

% procedural knowledge of environmental knowledge
% rules for computation of project cost in millions of Rs.

proposed_project(Pid,Title,Duration,research,
                 Area,Complexity,Cost) :-
        prop_proj(Pid,Title,Duration,Manpower,
                  research,Area,Complexity),
        Cost is Manpower*Duration*0.025.

proposed_project(Pid,Title,Duration,turnkey,
                 Area,Complexity,Cost) :-
        prop_proj(Pid,Title,Duration,Manpower,
                  turnkey, Area,Complexity),
        Cost is Manpower*Duration*0.020.

proposed_project(Pid,Title,Duration,developmental,
                 Area,Complexity,Cost) :-
        prop_proj(Pid,Title,Duration,Manpower,
                  developmental, Area,Complexity),
        Cost is Manpower*Duration*0.022.
```

uncertainty as an NP-measure is attached to these facts

Project cost is computed as a function of manpower required and the duration of the project by the three following rules of EKB. These rules correspond to *research*, *turnkey* and *developmental* project types. It is also possible to use the argument names as defined in rule_def instead of the positional arguments.

Complete set of Prolog language constructs is available for specification of procedural knowledge. Consequently, it is possible to represent any complex form of procedural knowledge.

```
% rules specifying probable managers of the employees
probable_manager(E,Name,Qual,Exp,Age,LeadApt) :-
        employee(E,Name,Qual,Exp,Age,LeadApt),
        participated_as(E,R),
         role(R,leader).
```

All the employees who have lead some project earlier certainly belong to the set of probable managers.

It can also be noticed that the access to the external database is transparent.

The employees who have participated as *deputy_leader* in some project are probable managers to a lesser extent as indicated by the NP-measure [0.7,1].

```
probable_manager(E,  Name,Qual,Exp,Age,LeadApt) :-
        employee(E,Name,Qual,Exp,Age,LeadApt),
        participated_as(E,R),
        role(R,deputy_leader) @ [0.7,1].
```

Note: Pages B-19 and 20 list the contents of of the database using the get command

```
pa> list mkb
```

modelling knowledge under the three categories of decision models, decision stages and decision rules is displayed

```
% decision models (model base)

% decision model to select a project and assign
% a project leader to the selected project.

sel_proj_and_assign_leader(Budget)  <-
     select_a_project(PJ,Budget) ->
     assign_project_leader(L,PJ).
```

2 decision stages are used to compose this decision model PJ represents the project selected

```
% decision stages (stage base)

% decision stage to select a set of projects P from the
% set of proposed projects

select_projects(Max_budget) <-
  select subset P of proposed_project
      using
          precond (meets_cost_constr(P,Max_budget) and
                   good_mix_of_proj(P) and
                   market_driven(P)),
          rank     sum(PJ.cost:in(PJ,P)).
```

Alternative P is a subset satisfying the precondition

Rank the alternatives based on total cost of projects selected, i.e., the best utilisation of the budget

```
% decision stage to select a project P from the set of
% proposed projects

select_a_project(P,Budget) <-
  select P of proposed_project
      using
          precond ((promising(P) or
                    expertise_available(P)) and
                    (P.cost =< Budget)),
          rank     100/P.duration+2*P.cost.
```

Alternative P is a proposed project satisfying precondition

Alternatives are ranked as a function of *duration* and *project cost*

```
% decision stage to assign a project leader from the list
% of probable managers to a given project P

assign_project_leader(L,P) <-
  select L of probable_manager
      using
          precond    can_handle(L,P.area,P.type),
          rank       poss_success(L,P).
```

```
% decision rules (rule base)

% set of projects, P, meets the cost constraint if the total
% cost of the projects is below the available budget and
% there are not more than 2 expensive projects in the set P

meets_cost_constr(P,Max_budget) <-
        (sum(PJ.cost:in(PJ,P)) =< Max_budget) and
        (count(PJ:(in(PJ,P) and PJ.cost = ~expensive)) =< 2).
```

Aggregate functions are defined using the predicate *in*

```
% the set of projects P is a good project mix provided P
% contains at least one turnkey, one developmental and one
% research project.

good_mix_of_proj(P) <-
        existsa(P,turnkey) and
        existsa(P,developmental) and
        existsa(P,research).

% the set of projects P is possibly a good project mix if
% P contains at least one turnkey and one
% developmental project

good_mix_of_proj(P) <-
      existsa(P,turnkey) and
      existsa(P,developmental) @ [0.8,1].

existsa(P,Type) <-
        exists(PJ:in(PJ,P), PJ.type = Type).

% the set of projects P is market driven if there is at least
% one project of a promising area

market_driven(P)  <-
        exists(PJ:in(PJ,P),
                exists(A:promising_area(A),A.area = PJ.area)).

promising(P) <- exists(A:promising_area(A),A.area = P.area).
```

An NP-measure is used to indicate the uncertainty

Nested existential quantification

The typical requirement during the modelling of decision problems is aggregate functions and quantifications. The decision rules of this environment provide this important facility.

```
% expertise is certainly available in the organisation for
% project P if there is at least one employee who has lead
% (or is leading) another project in the area of P

expertise_available(P) <-
        exists(E:(employee(E)              and
                  participated_as(E,R)     and
                  team_of(PJ,R)            and
                  PJ.area = P.area),
               R.role = leader).
```

transparent access to external database

```
% expertise available to a lesser extent for
% project P if there is at least one employee who has lead
% (or is leading) another project in the area of P
```

usage of relationship predicates *team_of* and *participated_as*

```
expertise_available(P) <-
        exists(E:(employee(E)              and
                  participated_as(E,R)     and
                  team_of(PJ,R)            and
                  PJ.area = P.area),
               R.role = deputy_leader)  @ [0.8,1].

% little expertise available in the organisation for
% project P if there is at least one employee who has lead
% (or is leading) another project in the area of P
expertise_available(P) <-
        exists(E:(employee(E)              and
                  participated_as(E,R)     and
                  team_of(PJ,R)            and
                  PJ.area = P.area),
               R.role = member)          @ [0.5,1].

% the probable manager P can certainly handle a project
% of a given area and type if he/she has past experience
% in that area
can_handle(P,Area,Type) <-
        has_past_experience(P,Area).

% the probable manager P can possibly handle a project
% if he/she has lead other projects
can_handle(P,Area,Type) <-
        has_lead_other_projects(P) @ [0.9,1].

% the probable manager P can more or less handle a project
% of a given type if he/she has the requisite aptitude and
% is qualified for the project type
can_handle(P,Area,Type) <-
        has_requisite_aptitude(P) and
        qualified(P,Type) @ [0.8,1].
```

```
% rules defining the aptitude in terms of leadership aptitude
has_requisite_aptitude(P) <-
        P. leader ship_apt = ~ excellent.

has_requisite_aptitude(P) <-
        P.leadership_apt = ~ above_average @ [0.8,1].

% rules defining qualified in terms of qualifications and
% age

qualified(P,research) <- P.qual = phd.
qualified(P,research) <- P.qual = mtech @ [0.7,1].
qualified(P,developmental) <- P.qual = mtech.
qualified(P,developmental) <-
        P.qual = btech and P.age > 30 @ [0.7,1].
qualified(P,turnkey) <- P.age > 32.


% if P has lead a project of a given area in the past
% then P certainly has the past experience

has_past_experience(P,Area) <-
        exists(PJ:project(PJ),(PJ.area = Area and
                               team_of(PJ,R) and
                               R.role = leader and
                               participated_as(E,R) and
                               E.no = P.no)).

% if P has participated in a project of a given area
% in the past as a deputy leader then P possibly has the
% past experience

has_past_experience(P,Area) <-
      exists(PJ:project(PJ),(PJ.area = Area and
                             team_of(PJ,R) and
                             R.role = deputy_leader and
                             participated_as(E,R) and
                             E.no = P.no)) @ [0.8,1].

% P has lead at least one project in the past

has_lead_other_projects(P) <-
        exists(R:role(R), (R.role = leader and
                           participated_as(E,R) and
                           E.no = P.no)).
```

```
% P is over loaded if he/she is already participating in
% approximately between 3 to 5 projects

overJoaded(P) <-
        count(R:(role(R) and participated_as(E,R)),
              E.no = P.no)
        = ~heavily_loaded.

% possibility M leading P successfully is defined as
% the past record of M or likely success of M leading P and
% M having the knowledge of project area.

poss_success(M,P) <-
        past_success(M) or
        (likely_success(M,P.type) and
         has_knowledge(M,P.area)).

% the past record is good if all the projects in which M
% participated as leader or deputy leader have been
% successfully completed

past_success(M) <-
        all(J:(project(J) and team_of(J,R) and
                (R.role = leader or R.role = deputy_leader)
                and participated_as(E,R) and E.no = M.no),
            J.status = success).


likely_success(M,research) <-
        M.qual = phd and M.experience > = 5.
likely_success(M,research) <-
        M.qual = mtech and M.experience >= 9 @ [0.9,1].
likely_success(M,developmental) <-
        M.qual = mtech and M.experience >= ~senior.
likely_success(M,turnkey) <-
        M.experience = ~fairly_senior.

has_knowledge(M,Area) <-
        exists(P:(project(P) and P.area = Area).
                (team_of(P,R) and participated_as(E.R) and
                 E.no = M.no)).

pa> activate select_a_project(P,3).
```

Activation of a predefined decision model. The second parameter is budget available

```
-------------------------------------------------------------------
Decision Stage :select_a_project                        Alternative # 1
Rank value     :fposs([6.83533,10.0033,13.3378,17.5618])
Precondition   :[0.39394,1]
Postcondition  :[1,1]
-------------------------------------------------------------------
proposed_project( pid         :p1,
                  title       :dist_dbms,
                  duration    :cposs([36,48]),
                  type        :developmental,
                  area        :database,
                  complexity  :~quite_complex,
                  cost        :fposs([2.376,3.96,5.28,7.392]).
-------------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n

-------------------------------------------------------------------
Decision Stage :select_a_project                        Alternative # 2
Rank value     :fposs([7.0119,8.25555,8.25555,9.81666])
Precondition   :[1,1]
Postcondition  :[1,1]
-------------------------------------------------------------------
proposed_project( pid         :p6,
                  title       :expsys_shell,
                  duration    :~about_18_months,
                  type        :research,
                  area        :artificial_intelligence,
                  complexity  :~complex,
                  cost        :fposs([1.125,1.35,1.35,1.575]).
-------------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n

-------------------------------------------------------------------
Decision Stage :select_a_project                        Alternative # 3
Rank value     :fposs([6.03333,7.76666,7.76666,10.0555])
Precondition   :[1,1]
Postcondition  :[1,1]
-------------------------------------------------------------------
proposed_project( pid         :p2,
                  title       :dss_generator,
                  duration    :~about_2_years,
                  type        :research,
                  area        :decision_support,
                  complexity  :~moderately_complex,
                  cost        :fposs([1.35,1.8,1.8,2.25]).
-------------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : s
```

results of rank rule, pre- and postcondition rules

n to get the next alternative

s to skip this decision stage

Since the activated decision model is composed of only one decision stage, *activate* command gets terminated

```
pa> activate select_projects(fposs([10,12,14,16])).
```

Activation of decision stage *select_projects* with an imprecise budget value specifying *approximately between 12 and 14*

This feature enables the decision maker to supply imprecise input parameters.

It can be noticed that the alternatives are subsets or groups of proposed projects

```
-------------------------------------------------------------------
Decision Stage :select_projects                        Alternative # 1
Rank value     :fposs([6.30899,8.118,8.118,9.92699])
Precondition   :[1,1]
Postcondition  :[1,1]
-------------------------------------------------------------------
proposed_project( pid        :p6,
                  title      :expsys_shell,
                  duration   :~about_18_months,
                  type       :research,
                  area       :artificial_intelligence,
                  complexity :~complex,
                  cost       :fposs([1.125,1.35,1.35,1.575]).
proposed_project( pid        :p5,
                  title      :gims,
                  duration   :12,
                  type       :turnkey,
                  area       :games_management,
                  complexity :~not_so_complex,
                  cost       :3.6).
proposed_project( pid        :p3,
                  title      :tp_package,
                  duration   :36,
                  type       :developmental,
                  area       :operating_systems,
                  complexity :~complex,
                  cost       :fposs([1.584,3.168,3.168,4.752]).
-------------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n
```

```
-------------------------------------------------------------------------
Decision Stage :select_projects                              Alternative # 2
Rank value     :fposs([6.534,8.56799,8.56799,10.602])
Precondition   :[0.789091,1]
Postcondition  :[1,1]
-------------------------------------------------------------------------
proposed_project( pid        :p2,
                  title      :dss_generator,
                  duration   :~about_2_years,
                  type       :research,
                  area       :decision_support,
                  complexity :~moderately_complex,
                  cost       :fposs([1.35,1.8,1.8,2.25]).
proposed_project( pid        :p5,
                  title      :gims,
                  duration   :12,
                  type       :turnkey,
                  area       :games_management,
                  complexity :~not_so_complex,
                  cost       :3.6).
proposed_project( pid        :p3,
                  title      :tp_package,
                  duration   :36,
                  type       :developmental,
                  area       :operating_systems,
                  complexity :~complex,
                  cost       :fposs([1.584,3.168,3.168,4.752]).
-------------------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n

-------------------------------------------------------------------------
Decision Stage :select_projects                              Alternative # 3
Rank value     :fposs([7.10099,8.91,10.23,12.567])
Precondition   :[0.744195,1]
Postcondition  :[1,1]
-------------------------------------------------------------------------
proposed_project( pid        :p6,
                  title      :expsys_shell,
                  duration   :~about_18_months,
                  type       :research,
                  area       :artificial_intelligence,
                  complexity :~complex,
                  cost       :fposs([1.125,1.35,1.35,1.575]).
proposed_project( pid        :p5,
                  title      :gims,
                  duration   :12,
                  type       :turnkey,
                  area       :games_management,
                  complexity :~not_so_complex,
                  cost       :3.6).
```

```
proposed_project( pid        :p1,
                  title      :dist_dbms,
                  duration   :cposs([36,48]),
                  type       :developmental,
                  area       :database,
                  complexity :~quite_complex,
                  cost       :fposs([2.376,3.96,5.28,7.392]).
-------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n

-------------------------------------------------------------
Decision Stage :select_projects                    Alternative # 4
Rank value     :fposs([7.326,9.35999,10.68,13.242])
Precondition   :[0.713371,1]
Postcondition  :[1,1]
-------------------------------------------------------------
proposed_project( pid        :p2,
                  title      :dss_generator,
                  duration   :~about_2_years,
                  type       :research,
                  area       :decision_support,
                  complexity :~moderately_complex,
                  cost       :fposs([1.35,1.8,1.8,2.25]).
proposed_project( pid        :p5,
                  title      :gims,
                  duration   :12,
                  type       :turnkey,
                  area       :games_management,
                  complexity :~not_so_complex,
                  cost       :3.6).
proposed_project( pid        :p1,
                  title      :dist_dbms,
                  duration   :cposs([36,48]),
                  type       :developmental,
                  area       :database,
                  complexity :~quite_complex,
                  cost       :fposs([2.376,3.96,5.28,7.392]).
-------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n

No more alternatives to the decision stage select_projects.
```

Activate command terminates with this message after exhausting all the probable alternatives.

```
pa> activate sel_proj_and_assign_leader(3).
```

Activation of a predefined decision model with budget value as 3

```
-------------------------------------------------------------
Decision Model :sel_proj_and_assign_leader
Decision Stage :select_a_project                  Alternative # 1
Rank value     :fposs([6.83533, 10.0033, 13.3378, 17.5618])
Precondition   :[0.39394,1]
Postcondition  :[1,1]
-------------------------------------------------------------
proposed_project( pid        :p1,
                  title      :dist_dbms,
                  duration   :cposs([36,48]),
                  type       :developmental,
                  area       :database,
                  complexity :~quite_complex,
                  cost       :fposs([2.376,3.96,5.28,7.392]).
-------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n

-------------------------------------------------------------
Decision Model :sel_proj_and_assign_leader
Decision Stage :select_a_project                  Alternative # 2
Rank value     :fposs([7.0119, 8.25555, 8.25555, 9.81666])
Precondition   :[1,1]
Postcondition  :[1,1]
-------------------------------------------------------------
proposed_project( pid        :p6,
                  title      :expsys_shell,
                  duration   :~about_18_months,
                  type       :research,
                  area       :artificial_intelligence,
                  complexity :~complex,
                  cost       :fposs([1.125,1.35,1.35,1.575]).
-------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n
```

```
-------------------------------------------------------------------
Decision Model :sel_proj_and_assign_leader
Decision Stage :select_a_project                        Alternative # 3
Rank value     :fposs([6.03333,7.76666,7.76666,10.0555])
Precondition   :[1,1]
Postcondition  :[1,1]
-------------------------------------------------------------------
proposed_project( pid          :p2,
                  title        :dss_generator,
                  duration     :~about_2_years,
                  type         :research,
                  area         :decision_support,
                  complexity   :~moderately_complex,
                  cost         :fposs([1.35,1.8,1.8,2.25]).
-------------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : c
```

commit this alternative

Project *p2* is committed for the first decision stage and now PlanAid proceeds to the alternatives of the second decision stage to assign a project leader

```
-------------------------------------------------------------------
Decision Model :sel_proj_and_assign_leader
Decision Stage :assign_project_leader                   Alternative # 1
Rank value     :[1,1]
Precondition   :[0.9,1]
Postcondition  :[1,1]
-------------------------------------------------------------------
probable_manager( no           :p4,
                  name         :murali,
                  qual         :phd,
                  experience   :~about_9y,
                  age          :32,
                  leadership_apt :~excellent).
-------------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n
```

```
Decision Model :sel_proj_and_assign_leader
Decision Stage :assign_project_leader                    Alternative # 2
Rank value     :[1,1]
Precondition   :[0.9,1]
Postcondition  :[1,1]
-------------------------------------------------------------------------
probable_manager( no              :p2,
                  name            :vinay,
                  qual            :mtech,
                  experience      :9,
                  age             :~about_33y,
                  leadership_apt  :~very_good).
-------------------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : n

-------------------------------------------------------------------------
Decision Model :sel_proj_and_assign_leader
Decision Stage :assign_project_leader                    Alternative # 3
Rank value     :[0.9,1]
Precondition   :[1,1]
Postcondition  :[1,1]
-------------------------------------------------------------------------
probable_manager( no              :p1,
                  name            :narsi,
                  qual            :mtech,
                  experience      :9,
                  age             :32,
                  leadership_apt  :~good).
-------------------------------------------------------------------------
Enter n - next alt | s - skip stage | c - commit : s

No more alternatives to the decision stage select_a_project.

pa> query        —— exploratory information retrieval
```

The query language provides an unified high level non-procedural interface for accessing environmental knowledge base. Also, this language facilitates specification of imprecise and/or uncertain queries to access possibly imprecise and/or uncertain information. Lastly, this system also provides usage of decision rules in the query specification which can eliminate specification of simple decision models.

```
pa/q> get P.title where prop_proj(P).
```

get all the titles of of the *prop_proj*

```
prop_proj
title                    NP
-----------------------------
dist_dbms                [1,1]
dss_generator            [1,1]
tp_package               [1,1]
prolog_compiler          [1,1]
gims                     [1,1]
expsys_shell             [1,1]
```

```
pa/q> get P.title,P.cost where proposed_project(P).
```

This query uses a rule (procedural know.) *proposed_project* to derive project *cost*

```
prop_proj
title                cost                                          NP
-------------------------------------------------------------------------
dss_generator        fposs([1.35,1.8,1.8,2.25])                    [1,1]
prolog_compiler      fposs([1.575,3,3.75,6.3])                     [1,1]
expsys_shell         fposs([1.125,1.35,1.35,1.575])                [1,1]
gims                 3.6                                           [1,1]
dist_dbms            fposs([2.376,3.96,5.28,7.392])                [1,1]
tp_package           fposs([1.584,3.168,3.168,4.752])              [1,1]
```

```
pa/q> get P.title,P.duration where
          prop_proj(P) and P.duration = ~about_2_years.
```

retrieve proposed projects with duration *about 2 years*

```
prop_proj
title                duration                          NP
-------------------------------------------------------------
dss_generator        fposs([18,24,24,30])              [0.5,1]
prolog_compiler      fposs([21,24,30,36])              [0,1]
expsys_shell         fposs([15,18,18,21])              [0,0.333]
```

Values of duration attribute are expanded from lc_def to numerical values. And, the degree to which the condition is satisfied is reported in the last column

```
pa/q> get P.title,P.duration where
        prop_proj(P) and P.duration = ~about_2_years
      @[0.5,1].
```

Threshold NP-measure further restricts the selected tuples

```
prop_proj
title              duration                NP
-------------------------------------------------
dss_generator      fposs([18,24,24,30])    [0.5,1]

pa/q> get count(P:(proposed_project(P) and
                   P.cost = ~expensive)).

2.091
```

aggregate function *count* returns fuzzy count of *expensive* projects

```
pa/q> get E.all where employee(E).

employee
no    name     qual    experience  age          leadapt      NP
----- -------- ------- ----------- ------------ ------------ -------
p1    narsi    mtech   9           32           ~good        [1,1]
p2    vinay    mtech   9           ~about_33y   ~very_good   [1,1]
p3    bgp      btech   ~below_4y   ~young       ~average     [1,1]
p4    murali   phd     ~about_9y   32           ~excellent   [1,1]
p5    suren    mtech   5           ~about_30y   ~good        [1,1]
p6    phule    mtech   9           ~about_32y   ~very_good   [1,1]
p7    chandu   mtech   ~about_6y   ~about_30y   ~good        [1,1]

pa/q> get P.all where project(P).

project
pid   title      area              status       NP
----- ---------- ----------------- ------------ -------
j1    net_dbms   database          success      [1,1]
j2    rel_dbms   database          success      [1,1]
j3    db_appl    applications      success      [1,1]
j4    db_mach    database          abandoned    [1,1]
j5    dss        decision_support  in_progress  [1,1]

pa/q> get E.name,P.title,R.role where
        participated_as(E,R) and team_of(P,R).
```

This query lists all instances of the relationships *participated_as* and *team_of*

employee name	project title	role	NP
narsi	net_dbms	deputy_leader	[1,1]
narsi	dss	leader	[1,1]
vinay	rel_dbms	leader	[1,1]
bgp	rel_dbms	member	[1,1]
murali	db_appl	leader	[1,1]
suren	net_dbms	member	[1,1]
suren	db_appl	deputy_leader	[1,1]
phule	rel_dbms	leader	[1,1]
phule	db_mach	deputy_leader	[1,1]
chandu	net_dbms	member	[1,1]
chandu	db_appl	deputy_leader	[1,1]

```
pa/q> get E.name,
        count(R:(participated_as(E,R) and
                 R.role = leader))
      where employee(E).
```

employee name	count	NP
narsi	1	[1,1]
vinay	1	[1,1]
bgp	0	[1,1]
murali	1	[1,1]
suren	0	[1,1]
phule	1	[1,1]
chandu	0	[1,1]

```
pa/q> get E.name where employee(E) and
          exists(R:participated_as(E,R),R.role=leader).
```

employee name	NP
narsi	[1,1]
vinay	[1,1]
murali	[1,1]
phule	[1,1]

```
pa/q> get E.name where employee(E) and
                 all(R:participated_as(E,R),R.role=leader).
```

employee name	NP
vinay	[1,1]
murali	[1,1]

```
pa/q> get P.name where probable_manager(P).

probable_manager
name            NP
-----------------
narsi           [1,1]
vinay           [1,1]
murali          [1,1]
phule           [1,1]
suren           [0.7,1]
chandu          [0.7,1]
```

procedural knowledge component is used to derive the list of probable managers

```
pa/q> get P.name where probable_manager(P) and
                       has_past_experience(P,database).

probable_manager
name            NP
-----------------
narsi           [0.8,1]
vinay           [1,1]
phule           [1,1]
```

combination of procedural knowledge and decision rule

```
pa/q> get L.name,P.name where probable_manager(L) and
                    proposed_project(P) and
                    can_handle(L,P.area,P.type) @ [0.5,1].
```

List of all project managers who *can handle* the proposed projects

probable_manager name	proposed_project title	NP
narsi	dss_generator	[1,1]
narsi	prolog_compiler	[0.9,1]
narsi	expsys_shell	[0.9,1]
narsi	gims	[0.9,1]
narsi	dist_dbms	[0.9,1]
narsi	tp_package	[0.9,1]
vinay	dss_generator	[0.9,1]
vinay	prolog_compiler	[0.9,1]
vinay	expsys_shell	[0.9,1]
vinay	gims	[0.9,1]
vinay	dist_dbms	[1,1]
vinay	tp_package	[0.9,1]
murali	dss_generator	[0.9,1]
murali	prolog_compiler	[0.9,1]
murali	expsys_shell	[0.9,1]
murali	gims	[0.9,1]
murali	dist_dbms	[0.9,1]
murali	tp_package	[0.9,1]
phule	dss_generator	[0.9,1]
phule	prolog_compiler	[0.9,1]
phule	expsys_shell	[0.9,1]
phule	gims	[0.9,1]
phule	dist_dbms	[1,1]
phule	tp_package	[0.9,1]

pa/q> *ins prop_proj(p7, gks, ~about_2_years, 2, developmental, graphics, ~complex).*

Assertion of a new proposed project with the argument values specified in the predefined order

pa/q> *get P.title, P.duration, P.type where prop_proj(P).*

This new asserted fact is available for further queries

prop_proj title	duration	type	NP
gks	~about_2_years	developmental	[1,1]
dist_dbms	cposs([36,48])	developmental	[1,1]
dss_generator	~about_2_years	research	[1,1]
tp_package	36	developmental	[1,1]
prolog_compiler	~above_2_years	research	[1,1]
gims	12	turnkey	[1,1]
expsys_shell	~about_18_months	research	[1,1]

```
pa/q> ins db_experts(P.name) where probable_manager(P) and
                  can_handle(P,database,research) @ [0.5,1].
```

Assertion of derived information

```
pa/q> get D.name where db_experts(D).

db_experts
name        NP
----------------------
narsi       [0.9,1]
vinay       [1,1]
murali      [0.9,1]
phule       [1,1]

pa/q> ins proj_part( P.name,
                     count(R:(participated_as(P,R) and
                              R.role = leader)),
                     count(R:(participated_as(P,R) and
                              R.role = deputy_leader)))
      where employee(P).
```

Extraction of aggregate information from the external database

This feature improves efficiency by eliminating repeated access to the databases.

```
pa/q> get PP.all where proj_part(PP).

proj_part
name        lead    dylead    NP
-------------------------------------
chandu      0       1         [1,1]
phule       1       1         [1,1]
suren       0       1         [1,1]
murali      1       0         [1,1]
bgp         0       0         [1,1]
vinay       1       0         [1,1]
narsi       1       1         [1,1]

pa/q> del P where prop_proj(P) and
                  P.area = games_management.
```

```
pa/q> get P.title,P.duration,P.type where prop_proj(P).

prop_proj
title                duration                type            NP
-------------------- ----------------------- --------------- -----
gks                  ~about_2_years          developmental   [1,1]
dist_dbms            cposs([36,48])          developmental   [1,1]
dss_generator        ~about_2_years          research        [1,1]
tp_package           36                      developmental   [1,1]
prolog_compiler      ~above_2_years          research        [1,1]
expsys_shell         ~about_18_months        research        [1,1]

pa/q> del P where prop_proj(P) and P.duration > 30
          @ [0.5,1].

pa/q> get P.title,P.duration,P.type where prop_proj(P).

prop_proj
title                duration                type            NP
-------------------- ----------------------- --------------- -----
gks                  ~about_2_years          developmental   [1,1]
dss_generator        ~about_2_years          research        [1,1]
prolog_compiler      ~above_2_years          research        [1,1]
expsys_shell         ~about_18_months        research        [1,1]

pa/q> rep P with (P.duration:=P.duration*1.2)
          where prop_proj(P).

        Increase the duration of all projects
        by 20% (fuzzily). The new values of
        duration are shown below.

pa/q> get P.title,P.duration where prop_proj(P).

prop_proj
title                duration                                NP
-------------------- --------------------------------------- -----
gks                  fposs([21.6,28.8,28.8,36])              [1,1]
dss_generator        fposs([21.6,28.8,28.8,36])              [1,1]
prolog_compiler      fposs([25.2,28.8,36,43.2])              [1,1]
expsys_shell         fposs([18,21.6,21.6,25.2])              [1,1]

pa/q> exit      ---- Exit from explore command

pa> exit        ---- End of PlanAid session

$
```

Appendix C
A Comparison of DSS and ES

There has been a lot of debate on the comparison of DSS vs ES. Majority of researchers feel that these systems are complementary (Pfeifcr and Luthi 1987). On the contrary, the arguments such as expert systems are a special class of decision support systems, have also been prevalent. Then, there are arguments in favor of the integration of ES and DSS technologies (Turban and Watkins 1985; Teng *et al* 1988). Obviously, these systems do exhibit properties which make the comparison possible. A comprehensive comparison is presented in the table below based on (Turban and Watkins 1985; Pfeifcr and Luthi 1987; Raghavan and Chand 1988). Majority of the differences arise from the difference in the basic philosophy of mimicking and assisting the human being.

Despite the differences, it was generally agreed that it would be advantageous to use the concepts of AI/ES in the development of DSS. Keen (1986) suggests that one of the areas in which decision support can be extended is through semi-expert systems and the DSS field can exploit the opportunities of ES technology. He, further, favors the idea of providing a consultant rather than an assistant and merging of Al tools into DSS so that they help decision maker point to or even select alternatives rather than evaluating the decision maker's set of choices.

The domains of DSS and ES (see table below) can be thought of two extremities of broad-shallow and narrow-deep. It is, however, possible that applications fall in between these extremes. Consequently, decision problems having intersecting characteristics of ES and DSS can best exploit the merger of AI/ES technologies into that of DSS. Borche and Hartvigsen (1990) report a knowledge based system for strategic market planning and choice of market segments which exhibits the characteristics of ES and DSS.

Teng *et al* (1988) propose a unified architecture for intelligent DSS, integrating DSS and ES concepts. Their architecture comprises of database, knowledge base, natural language dialog for user-interface and a central intelligence manager consisting of an inference engine and intelligent supervisor. The possible methods of integration of ES and DSS have also been described by Turban and Watkins (1985).

Comparison of DSS and Expert Systems

DSS		ES
Assist decision maker	**Objective**	Enhance decision making Replicate/replace experts
With the user	**Control**	With the system
Complex, broad & shallow	**Domain**	Narrow & Deep
Support intuition	**Goal of system**	Complete solution
Ill-specified	**Goal type**	Well-specified
User is more intelligent Capable of solving the problem on his own	**User**	System is more intelligent Support problem solving process of the machine
Enhance decision making	**Philosophy**	Capture/transfer expertise
Not predictable From many domains	**Factors of influence**	Predictable Restricted
User queries the machine	**Query direction**	Machine queries the user

Appendix D
Fuzzy Expert System Shells

In this appendix we shall briefly present and discuss some expert system shells applicable to the area of approximate reasoning.

ARIES:

An approximate reasoning inference engine, ARIES, is one of the first of this category of expert system shells reported by Applebaum *et al* (1985). This is a propositional truth system based on interval valued logic. The lower bound of this interval indicates the amount by which proposition and inference rules along the path *support* the truth of the goal, while the upper bound, called *plausibility,* is a corresponding indication of the extent to which the same evidence fails to refute it. Propagation of intervals provides indications of support for the goal proposition and for its negation.

The authors suggest the application of t-norms and associated **co-t-norms** for combination of truth values. The system provides default truth formulae and it is possible to override the defaults by user-definable functions. The system, however, does not employ any fuzzy set concepts for the representation of imprecision.

SPII-1 and SPII-2:

These arc, perhaps, the first complete fuzzy expert system shells. SPII-1 is the first version of a fuzzy inference engine capable of handling both imprecision and uncertainty (Martin-Clouaire and Prade 1986). These inference engines employ fuzzy pattern matching for unification of imprecise and uncertain values represented by normalised δ-trapezoidal possibility distributions. The fuzzy pattern matching results in an uncertain truth value denoted by the necessity-possibility measures. These measures are normalised to confirm to the definitions of possibility theory. Further, these systems address the methods by which inferences can be made using multiple rules. Lebailley *et al* (1987) report the second version of SPII along with an application in the field of Petroleum Geology. This application produces the estimates of recoverable hydrocarbon volumes in a exploration prospect before drilling. The authors point out that "...through the use of possibilistic modelling it is possible to avoid the use of simulation methods for the computation of numerical quantities pervaded with imprecision and uncertainty".

Martin-Clouaire and Prade (1985) discuss various problems related to the representation and propagation of uncertainty and remark the equivalence of the degrees of belief and disbelief of MYCIN to the necessary measure and the complement to 1 of possibility measure respectively. They also suggest pruning criteria that can be employed in the control of the inference engine. The semantics and computation of generalised modus ponens using single rule or a collection of rules is described in (Martin-Clouaire 1989a). Further, the applicability of a number of t-norms and co-t-norms and implication functions are discussed for the propagation of imprecision and uncertainty. The computational efficiency that can be achieved through the usage of δ-trapezoidal possibility distribution for imprecise and uncertain values is also described in this paper. Martin-Clouaire (1989b) argues that a pair of NP-measures "make it possible to represent ignorance and to distinguish it from total uncertainty".

FRIL:

FRIL language is a super set of Prolog based on support logic programming (Baldwin 1987a, 1987b, 1988a, 1988b). Support logic programming is a generalisation of logic programming that combines fuzzy set theory and theory of evidence to form a suitable framework of reasoning with uncertainty. The uncertainty is represented by a support pair indicating the necessary and possible supports and the actual degree of support lies in between these supports.

A support logic program is a set of clauses of the form:

$$A \text{ :- } B_1, B_2, \ldots, B_n \bullet [S_n, S_p]$$

where A is an atom (same as that in Prolog)*

$B_1, \ldots, B_n$ are literals (positive/negative), and

$[S_n, S_p]$ is the support pair.

The interpretation of the support pairs is described below:

If $B_1, \ldots, B_n$ are all true then

A is necessarily supported to degree S_n and

$\neg A$ is necessarily supported to degree $1\text{-}S_p$.

Thus, S_n and S_p are necessary and possible supports respectively of A given $B_1, \ldots, B_n$ true.

A calculi of support logic programming is described for reasoning with uncertainty and propagation of uncertainty. Fuzzy sets and possibility distributions are used to represent imprecision in argument values and semantic unification (synonymous to fuzzy pattern matching) is used for unifying imprecise values. Another

interesting feature of this language is regarding the closed world assumption (CWA). FRIL docs not use the the CWA and therefore it is possible to model falsity (or unknown) and reason. The system combines support pairs from different proof paths by using intersection rule for same conclusion and by using Dempster rule for independent conclusions.

FLOPS:

Siler *et al* (1987) describe the features of a fuzzy expert system shell, FLOPS. This system, according to the authors, has been developed to meet the requirements of an application to classify the region clusters by analysing echo cardiogram images. FLOPS is a production system with extensions to represent and manipulate fuzzy numbers. Propagation of single-valued confidences in FLOPS is discussed in (Buckley 1988). The authors, while describing the desirable features of fuzzy expert system shells, point out that the single-valued confidences needed to be replaced by dual-valued confidences subsequently.

System Z-H:

Leung and Lam (1988) present an expert system shell having the capabilities to represent and perform reasoning on imprecise and uncertain knowledge. The system uses fuzzy sets to represent imprecision and certainty factors to represent uncertainty. The numeric certainty factors (between 0 and 1) can be defined with modifiers such as *close to 1, around 0.8,* etc. to capture imprecision in the certainty factors. The authors define the methods to propagate this kind of uncertainty during the process of inference using a notion of *similarity* M based on necessity - possibility measures as shown below:

$$M = P(F \mid F') \quad \text{if } N(F \mid F') > 0.5$$
$$= (N(F \mid F') + 0.5) * P(F \mid F') \quad \text{otherwise}$$

where P and N are possibility and necessity measures of matching the pattern F with datum F'.

The rules for combining evidence towards a conclusion are based on fuzzy arithmetic and it is notionally equivalent to the crisp arithmetic used for combining evidence in MYCIN. Fuzzy sets are mapped to linguistic values using the process of linguistic approximation.

Appendix E
Fuzzy Databases and Query Languages

The domains of DSS arc broad and shallow compared to expert systems, and arc built on large databases. The decision problems at higher organisational levels use data from both internal and external sources during information retrieval and model execution. Imprecision and uncertainty can arise in databases for decision support either because the database could be consisting of imprecise and uncertain data or the queries to be specified are of imprecise and uncertain nature or both. In this section we shall briefly survey some fuzzy databases and fuzzy query languages which could be effectively used in decision making.

Sources and various types of imprecise and uncertain data in modelling of data has been discussed in detail in (Bolloju 1990a). While the above paper deals with the modelling in general, a number of researchers have proposed extensions to relational data model with fuzzy set theory (Avni *et al* 1985; Raju and Mazumdar 1987; Zemankova-Leech and Kandel 1985). Avni *et al* (1985) use fuzzy sets and possibility distributions in the representation of imprecision in attribute values. Subsequently, the discuss a query language capable of specifying queries with fuzzy relational operators, connectives and quantifiers. The uncertainty in the results of query is. however. represented by possibility and certainty measures with the tuples selected.

In a similar approach, Zemankova-Leech and Kandel (1985) describe a fuzzy relational database model and the retrieval of data based on natural language like statements using fuzzy relational algebra. The query language extensively uses fuzzy sets defined using other fuzzy sets and modifiers. fuzzy set intersection/union operations. They point some drawbacks of necessity possibility measures (section 1.3.2) and present alternative definitions for these measures.

Fuzzy set theoretic extensions to relational data model have been defined by Raju and Mazumdar (1988) and a fuzzy query language as an extension of QUEL (Date 1981; Ullman 1984) is reported in (Bhattacharjee and Mazumdar 1989).

An interesting application of necessity possibility measures in the retrieval of documents through index terms is presented in (Prade and Testemale 1987). The authors claim that in their system it is possible to retrieve information using queries such as:

retrieve all the recent *documents which are* certainly *devoted to information retrieval and to incomplete information processing and* possibly *deal with databases.*

Rudensteiner and Bic (1989) argue that because of the simplicity of the relational data model it is not possible to capture many other forms of imprecision in relational databases. consequently, they advocate extension of semantic data models to capture imprecision in various constructs and pesent a generalised scheme of representation using fuzzy set theory.

Appendix F
T-norms and Co-T-nonns

A triangular norm is a function T with two arguments from [0,1]x[0,1] to [0,1] such that:

a) $T(a,b) = T(b,a)$

b) $T(a,T(b,c)) = T(T(a,b),c)$

c) $T(a,b) \leq T(c,d)$ if $a \leq c$ and $b \leq d$, and

d) $T(1,a) = a$ and $T(0,0) = 0$.

T-norms can be ordered as:

$$T_l(a,b) \leq max(0,a+b-1) \leq a.b \leq min(a,b)$$

where T_l is the least of the t-norms defined as:

$$T_l(a,b) = \begin{cases} a & if\ b=1 \\ b & if\ a=1 \\ 0 & othe\ r\ wise \end{cases}$$

Associated with each t-norm, by duality, there exists a co-t-norm defined as a function C with two arguments from [0,1]X[0,1] to [0,1] such that:

a) $C(a,b) = C(b,a)$

b) $C(a,C(b,c)) = C(C(a,b),c)$

c) $C(a,b) \leq C(c,d)$ if $a \leq c$ and $b \leq d$, and

d) $C(1,1) = 1$ and $C(a,0) = a$.

Similar to the T-norms, co-t-norms can also be ordered as:

$$max(a,b) \leq a+b-a.b \leq min(1,a+b) \leq C_g(a,b)$$

where C_g is the greatest of the co-t-norms defined as:

$$C_g(a,b) = \begin{cases} a & if\ b=0 \\ b & if\ a=0 \\ 1 & othe\ r\ wise \end{cases}$$

T-norms and corresponding co-t-norms have been considered applicable for fuzzy set intersection and union respectively.