# HANDLING FUZZY COMPLEX OBJECTS IN KNOWLEDGE-BASED SYSTEMS

A Master's Thesis
Presented by
Meltem TURHAN

to
the Graduate School of Natural and Applied Sciences
of Middle East Technical University
in Partial fulfillment for the Degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

MIDDLE EAST TECHNICAL UNIVERSITY
ANKARA
February 1994

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. İsmail Tosun
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Neşe Yalabık
Chairman of the Department

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Adnan Yazıcı
Supervisor

Examining Committee in Charge:

| | |
|---|---|
| Prof. Dr. Fatoş Yarman Vural | ........................... |
| Assoc. Prof. Dr. Adnan Yazıcı | ........................... |
| Asst. Prof. Dr. İlyas Çiçekli | ........................... |
| Asst. Prof. Dr. İsmail Hakkı Toroslu | ........................... |
| Asst. Prof. Dr. Göktürk Üçoluk | ........................... |

## ABSTRACT

### HANDLING FUZZY COMPLEX OBJECTS IN KNOWLEDGE-BASED SYSTEMS

Meltem TURHAN
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Adnan Yazıcı
February 1994, 121 pages

In this study a prototype of a knowledge-based system to represent and implement fuzzy complex objects along with their possible incomplete and missing attribute values is developed, and is implemented by utilizing a logic programming language, namely Turbo-Prolog. The work also includes the formal description of the proposed knowledge-based system of such objects. The prototype can represent, update, query and retrieve these fuzzy complex objects.

**Keywords:** fuzzy, complex, logic, Prolog.

Science code: 619.02.03

## ÖZ

### BILGIYE DAYALı SISTEMLERDE FUZZY KARMAŞıK NESNELERIN İŞLENMESI

Meltem TURHAN
Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı
Tez Yöneticisi: Assoc. Prof. Dr. Adnan Yazıcı
Şubat 1994, 121 sayfa

Bu tezde, mantıksal programlama dili Turbo-Prolog kullanılarak fuzzy, karmaşık ve aynı zamanda eksik ve tamamlanmamış alan değerleri içerebilen nesnelerin örneklenmesini sağlayan, bilgiye dayalı bir sistemin prototipi geliştirildi. Çalışma, ayrıca sözkonusu nesneleri içeren bilgiye dayalı sistemin formal tanımlanmasını kapsamaktadır. Prototip, karmaşık fuzzy nesnelerin gösterilimini, veri tabanından alınmasını, sorgulanmasını ve değiştirilmesini gerçekleştirmektedir.

**Anahtar sözcükler:** fuzzy, karmaşıklık, matık, Prolog

Bilim dalı sayısal kodu: 619.02.03

## ACKNOWLEDGEMENTS

It is great pleasure to express my gratitude to Assoc. Prof. Adnan Yazıcı for his continuous support and guidance throughout this work and his very valuable friendship.

I am debtful to my friend Onur Şehitoğlu for all his encouragement and help while I was spending sleepless nights preparing this thesis.

I want also to express my deep appreciation to Assis. Prof. Dr. Göktürk Üçoluk for his outstanding expertness in Prolog and Latex. He had always a working solution when one needed it.

I would like to thank my family for their continuous patience, understanding and support.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

Most of the existing knowledge-intensive applications such as Office Automation Systems, Decision Support Systems, Geographic Information Systems, Expert Database Systems, Engineering Desing, etc. of the real world often involves not only complex information but also some fuzzy and other kinds of uncertain properties. In general, information is found naturally, besides being complex, to be imprecise or fuzzy when representing personalities, physical features of individuals, subjective opinions and judgements in situations such as medical, economic forecasting or personal evaluation in many other applications other than the knowledge-intensive ones. Also in natural languages, numerous quantifiers (e.g., many, few, some, almost, etc.) are used when conveying vague information. Because of this wide spectrum of the need for handling complex and fuzzy information existing in the real world, there have been various researches in the last decade attempting to deal with complex objects in the frame of database systems ($NF^2$ DB models [2, 41, 48, 17]), (OODB Models [22, 51]), in knowledge-based systems [3], and in coupled systems [33]. As separate research attempts, there have also been advances dealing with fuzzy information in database systems [9, 23, 33, 54, 52] and in knowledge-based systems [31, 58] and with other uncertain information in both sytems [47, 34]. Although there have been some theoretical studies for representing and implementing the complex objects along with their fuzzy attributes in a database model [54], there has not been any serious attempt for handling such fuzzy complex objects in a knowledge-based systems. One of the most important objectives of knowledge-based systems is to let the user operate in an environment that models the real world as closely as possible. Hence, we believe that an attempt for handling some of the real world situations such as fuzzy and complex natures of the objects of many non-traditional knowledge-intensive applications in knowledge-based systems will result in natural and realistic systems in the sense that real life concepts are represented more accurately.

This study introduces a prototype of the representation and implementation of fuzzy complex objects including some other uncertain information, such as incomplete and null values. The prototype as a knowledge-based system is realized by using a logic programming environment.

The prototype developed during this study consists of a user interface which incorporates a parser for querying fuzzy complex valued attributes, a fuzzy evaluator, a complex fuzzy manipulation handler.

The work is covered in five chapters.

In Chapter II the background is explained in three sections: Logic Programming and Databases, Complex Objects and Fuzzy concepts. The first section contains description of the non first normal form data model (representing complex objects used). The second section is about fuzzy concepts in which fuzzy set theory, membership functions and similarity relations are explained as subsections.

Chapter III is on knowledge representation which states the formal descriptions, information retrieval and fuzzy evaluation. The original design of the system is covered in this chapter. The implementation is explained in chapter IV including reasons to choose Turbo Prolog, system architecture, and program description. Finally conclusions are stated in chapter V. In Appendix module dictionary, predicate dictionary and Prolog as a logic programming language are introduced.

# CHAPTER II

# BACKGROUND

## 2.1 Logic Programming and Databases

Interaction between Logic Programming and Databases is an important aspect of computer science which are explored in order to discover and profit from their common concepts.

The two fields developed independently throughout the 1970's and no commercial research has been employed to merge them. Surveys about building an interface between Logic Programming and Databases can be found in [19, 29].

Databases and Logic Programming have much to obtain from each other. Logic Programming can be used to extend Databases with many powerful features. These features are deduction, storage of non-record oriented data, information, and metadata. Also Logic Programming provides a uniform way of implementing uncertain values [8, 42]. On the other hand, database management systems can be used to give some features to logic programming for handling massive amount of data.

We can make a comparison of logic programming and database management systems by looking at them in detail.

First comparison is the view of databases they are dealing with. Logic programming systems manage small, single user databases which lie on main memory and consist of factual information and rules of deduction. On the other hand, database management systems deal with large, shared and massive amount of data. They are specialized in supporting efficient and reliable retrieval and update of data.

Both Logic programming and Databases show differences in queries. In logic programming, a query is answered by building chains of deductions by the combination of the facts and rules. In database management systems, the importance is focused on retrieving and updating data in massive storage in the most efficient way.

In both systems constraints are declared in a different manner. Constraints are correctness conditions for databases. In logical programming, constraints expressed through general-purpose rules, which are activated whenever the database is modified. In database systems, only limited number of them are typically expressed using the data definition language.

Logic programming offers a greater power for expressing queries and constraints as compared to that offered by data definition and manipulation languages of database systems. Furthermore, query and constraint representations are possible in a homogeneous formalism and their evaluation requires the same inference mechanisms, hence enabling more sophisticated reasoning about the database content. On the other hand, logic programming systems do not yet provide an effective technology for managing large, shared, persistent and reliable data connections [10].

Taking the advantages of both systems, some studies have been done for combining deductive capabilities of a logic programming system in formulating queries and constrains with the efficient and reliable data retrieval capabilities of database management systems. Most of these studies are based on the integration of Prolog and relational databases. Some of this studies can be briefly stated as follows:

PRO-SQL , a system for coupling Prolog to the system SQL/DS, developed at the IBM Research Center at Yorktown Heights. To support the logic programming (Prolog) with database facilities it uses some SQL statements. PRO-SQL runs under the VM/CMs operating system. Queries submitted to SQL are treated by dynamically defined query facility subsystem, which is typically used for casual queries [13].

EDUCE , a system for coupling Prolog to the database system Ingres, developed at the European Computer Industry Research Center in Munich. EDUCE is a CPR system, supporting both loose and tight coupling. It uses the INGRES database system [7].

The ESTEAM interface, developed in the framework of the Esprit Project ESTEAM, for coupling generic Prolog and database systems. The approach used by the ESTEAM interface is that of recognizing large base conjunctions. The system uses loose coupling in interactions with database. Tight coupling is utilized when the building of base conjunction queries interleaved with the computation of Prolog program. The ESTEAM interface supports disjunction, negation, recursion and the cut special predicate [15].

BERMUDA , a prototype developed at the University of Winsconsin, for coupling Prolog to the Brittom-Lee Intelligent Database Machine IDM 500. The focus of the project is on architectural issues, in particular those rising from employing multiple database interfaces at the same time.Several

Prolog engines might be active simultaneously, each executing its own Prolog program and communicating with the Bermuda Agent which receives query request and passes them to database interface [27].

CGW , an architecture for coupling Prolog to a database system developed at Stanford University, and PRIMO, a prototype of an interface between ARITY-PROLOG and the database system ORACLE, developed at the university of Modena, Italy. The rationale of CGW approach is to reduce accesses to database by cacheing queries and data. The proposed approach uses a low level interface to interact with the database, and control access to pages of data. A meta interpreter plays the role of Prolog interface, controlling the matching of individual database predicate [11, 12].

The QUINTUS interface between QUINTUS-PROLOG and the Unify Database System, is a product developed by Quintus Computer Systems of Mountain View, California. The interface operates at two levels: the relational level, providing a tight, one tuple at atime retrieval of tuples from underlying relations, and the view level, in which an entire Prolog rule is translated into a single query to the database, including joins as well as aggregate operations [43].

## 2.2 Complex Objects

Requirement for supporting complex objects have been increased in recent years in many areas of computing applications like engineering design, text managements, geographical information system, office automation, and computer aided design. Complex objects consist of attributes which can be not only simple attributes but also attributes which are objects themselves. Complex hierarchical terms as present in logic [59] ,CAD design objects [5] or objects used in office automation systems [4] are examples of complex objects. Relational data model which is based on first normal form required to be extended to deal with complex objects like report generators. Several complex object models [25, 40] have been proposed.

Although the relational data model is one of the most popular data models for database systems, it shows some weaknesses in representation of the complex objects of the non-traditional applications mentioned above. New extensions to deal with these weaknesses are proposed. These extensions include nested relations and aspects of object-oriented and functional programming.

The fundamental assumption in relational databases is the first normal form which restricts of all attributes of a relation to be atomic. The first normal form results in a homegeneous data model, it limits the naturalness of data [51].

Consider the emp relation of Table 2.1 and its non-first-normal form in Table 2.2. Although both versions contain same information, the relation in Table 2.2 is more compact and it respects the semantics of the information.

Table 2.1: First-normal-form-relation Employee

| employee | skill | child name | child age |
|---|---|---|---|
| Jones | Carpenter | Jack | 23 |
| Jones | Carpenter | Jim | 16 |
| Jones | Mechanic | Jack | 23 |
| Jones | Mechanic | Jim | 16 |
| Smith | Mechanic | Sam | 10 |
| Hurt | Cartoonist | Duffy | 5 |
| Hurt | Cartoonist | Sally | 7 |

Table 2.2: Non-normal-form-relation Employee

| | | children | |
|---|---|---|---|
| employee | skill | child name | child age |
| Jones | {Carpenter,Mechanic} | {(Jack,23), (Jim,16)} | |
| Smith | {Mechanic} | {(Sam,10)} | |
| Hurt | {Cartoonist} | {(Duffy,5),(Sally,7)} | |

In Table 2.2 both set-valued and record-valued attributes are accepted. Skill which is a set-valued attribute contains the list of skills and child which is a record valued attribute contains two more attributes: child name and date of birth. The resulting scheme is:

employee(name,skill,child(childname,bdate)).

That is, a relation contains a set of skill and (childname,bdate) tuples as attributes.

Four main reasons can be stated for eliminating the requirement of first normal form by comparing these two tables:

1. Non-first-normal-form representations of the information may be more compact, thus, a preferred physical representation of data.

2. Non-first-normal-form representations of information may provide a more intuitive user view of data.

3. Non-first-normal-form reduces the number of tuples and eliminates redundancy.

4. Related to the stated three reasons, querying is also simplified [41].

The nested relational data model (Non-first-normal-form) [2, 24, 35, 41, 46, 47, 48, 18, 28] was developed in order to extend the support for the complex objects in the relational data model. Nested relations can naturally represent the hierarchy of complex objects by allowing recursively defined relation-valued attributes in addition to atomic attributes. We will give a more formal definition of the $NF^2$ model in the next section.

### 2.2.1 The $NF^2$ Relational Data Structure

In the $1NF$ case basic domains are assumed to be n sets $D_i, \quad i = 1, 2, \ldots, n$ whose elements are not decomposable by the DBMS. The domain of a relational algebra is $D := D_1 \cup \cdots \cup D_n$.

Powerset of a set S is denoted with $P(S)$ and Cartesian product of two sets $S_1$ and $S_2$ with $S_1 \times S_2$.

The elements of this Cartesian product is denoted with $< \cdots >$, which means $s = < s_1, s_2 >$, with $s_1 \in S_1$ and $s_2 \in S_2$, where $s \in S_1 \times S_2$. Note that $\emptyset$, $(\exists)$ and $(\forall)$ are used for the empty set, existential and universal quantifiers respectively.

**Definition 1 (Complex Domains $C$)**
The set $C$ of "complex domains" is defined by:

1. $D \in C$
2. $C_1, C_2, \ldots, C_k \in C \quad \Rightarrow \quad P(C_1 \times C_2 \times \cdots \times C_k) \in C$
3. nothing else is in $C$

This definition allows powersets of Cartesian products of the basic or complex domains to be also a complex domain.

**Definition 2 (Schema $S$ and Description $B$.)**
Let $SetS$, $B$, $SetN$ be the set of allowed schemata, descriptions and names respectively. $SetS$ and $B$ are defined by:

1. $N \in SetN \Rightarrow < N, \emptyset > \in B$

2. $b_i = < N_i, S_i > \in B, \quad i = 1, 2, \ldots, k, \quad N_i \neq N_j$ for $i \neq j \quad \Rightarrow \{b_1, b_2, \ldots, b_k\} \in SetS$

3. $N \in SetN \wedge S \in SetS \quad \Rightarrow \quad < N, S > \in B$

Note that, the intention of this is to have a pair as a description consisting of a name as first component and a schema as second. But here, a schema is a set of descriptions which might also be an empty set. Whereas, the names which appear in the descriptions $b_i$ of a single schema must be unique.

**Definition 3 ($NF^2$ Relation)**
Given $b = < N, S > \in B$ a pair $R = < b, v >$ is an $NF^2$ relation, iff $S \neq \emptyset$ and $v \in dom(b)$, where $dom(b) = P(\underset{y \in sch(R)}{\times} dom(y))$.

In this definition $v$ is called "instance" or "value" of the schema defined by $b$.

In order to define attribute of $NF^2$ relation we shell give some preliminary definitions.

For the two components of a relation $R$, $des(R)$ is used for the first component, $val(R)$ is used for the second component. For the two components of a description $b \in B$, $nam(b)$ is used for the first component, $sch(b)$ is used for the second component. If $b = < N, S > = des(R)$ is given as the description of relation $R$ then,

$nam(des(R))$ the name ($N$)
$sch(des(R))$ the schema ($S$)

can be reached.

Since the schema of a relation is the set of attribute descriptions and not only the set of attributes (names), we can introduce $attr(R)$ to be the set of attributes as follows:

$$attr(R) = attr(des(R)) = \{N | (\exists) y \in sch(R) \wedge N = nam(y)\}$$

where $y \in sch(R)$ attribute description and $nam(y)$ is attribute of relation $R$. Note that single scheme have different attribute names.
Let $t \in val(R)$ be a tuple of a relation $R$, where $val(R) \in P(\underset{y \in sch(R)}{\times} dom(y))$.
Therefore a tuple has as many components as the cardinality of $sch(R)$. If $sch(R) = \{y_1, y_2, \ldots, y_k\}$ and if the order is fixed for the moment, we can state $t = < t_{y_1}, t_{y_2}, \ldots, t_{y_k} >$ with $t_{y_i} \in dom(y_i)$ [48].

## 2.3 Null Values

In this section a way to incorporate the semantics of different null types in the system is proposed [32].

(1) **'value exists but is unknown at the present time'** For example, we may record that the CITY, where a given PERSON works is unknown at the present time

(2) **'value does not exist' or 'value is non-applicable'** For example, we may record that no CHILDREN exist for a given PERSON.

(3) **'no information is available for the value'** For example, we may record that a TELNO is either unknown (Employee may have telephone but the number may not be known) or does not exist (Employee may not have it).

## 2.4 Fuzzy Concepts

In computer science, most of the available tools for formal modeling, reasoning and computing are deterministic, complete and precise in character. For instance, each piece of information is required to be exactly true or false in conventional database systems and classic logic. No intermediate value between true and false is allowed. Demands for the measurability of variables and numerical precision produce some simplifications, approximations and distortions. Generally precision implies that the model contains no ambiguities. Many well-known decision making methods often demand more exact information than decision maker has [26]. Certainty, on the other hand, implies the assumption of all of the parameters and structures required in the model are definitely and totally known and complete.

However, in many applications existing in real-life, information is rarely deterministic, crisp and cannot be precisely described because of the imprecise nature of the real-life knowledge [44, 57]. Trying to model this real-life applications with currently available models unavoidably cause loss of information between the model and the reality. Natural languages have many linguistic terms which have fuzzy characteristics. For example, we say:

Mary is young and tall.
Usually high quality goes with high price.
The smaller the drink, the cooler the blood, the cleaner the head.

In natural language,a lot of quantifiers like many, few, most, some, almost, probably, likely etc. are used when conveying vague information. These quantifiers are also important in soft sciences. In many cases, it is not possible to put these statements into a more precise form like assigning a numerical value or percentage. The solutions of real life problems would require more precise, detailed data than a human being can recognize, understand and process. Therefore, the solution found is not the real solution but it represents the idealized special case. Making precise statements becomes more difficult as the complexity of the system increases. Usually, partial or approximate information is used by the humans in the problem solving and decision making process. For example, most of the doctors diagnose diseases with partially known symptoms.

It is usually true that much human knowledge is neither totally certain nor totally consistent [33]. Nevertheless, uncertainty, imprecision and incompleteness are not unfortunate predictments to be overcome. Because, in most of the tasks performed by humans, only a low degree of certainty and precision is required, an approximate characterization is sufficient. For example information that the weather will be windy is enough to wear coat, there is no need to know the speed of the wind. The human brain uses the advantage of these tolerance for precision and encodes the task relevant information into fuzzy set labels that are approximate relations to the primary data.This advantage distinguishes human intelligence from the intelligence embodied in current computers.

As mentioned above, sometimes using fuzzy terms can be more descriptive than using the exact values. Another example in the statements is:

1. Tom run fast.

2. Tom run at an average speed of 30 km/h.

The first one is more descriptive and it is easier to understand and assign a truth value.

This capability of processing fuzzy information is subject of many researches on artificial intelligence and knowledge engineering and databases [23, 36]. Since, human thinking is rarely binary, it is often very difficult to represent and process human knowledge formally with binary systems. So it is difficult to model the uncertainty, imprecision and incompleteness in human knowledge by the computer.

The concept of a fuzzy set is introduced by L.A. Zadeh in 1965 which is an alternative to classical set theory. Fuzzy set theory proposes an alternative approach to handle the uncertainty in the nature of human knowledge.

## 2.5 Fuzzy Set Theory

In classical set theory, a set is defined normally as a collection of objects, which can be finite, countable or uncountable. Each single object can either belong to or not belong to set $A$. An object $x$ which belongs to a set $A$ is called an element of the set $A$. A set with no elements is called empty set and denoted as $\emptyset$, whereas a set including all elements of the universe of discourse is called a complete set and denoted by $\Omega$. As a mathematical model, classical set theory is a basic tool for describing crisp objects precisely. However classical set theory cannot be sufficient for describing and handling fuzzy objects.

First of all, to use classical set theory for treating fuzzy objects, particularly in human sciences, problem should be simplified or separation lines should be drawn artificially among objects. Hence the inherent fuzzy characteristics of the objects are lost. Artificially drawing separation lines is a distortion of the original characteristics of objective events especially near the separation line. An essential characteristic of a vague concept is that the boundaries of the domain of its applicability are not fixed.

Moreover, the parameters of the crisp objects are well-defined and precise so that exact mathematical models can be constructed. However in fuzzy environments, precise data necessary for constructing the mathematical model is not provided. Therefore, a mathematical model for the fuzzy objects cannot be defined by using classical set theory. In general, human systems include many subjective factors and heavy fuzziness and conventional mathematics cannot be used as a universal method to achieve the quantification of social and human sciences.

Lastly, for complex events, precision is usually unnecessary, causes useless information and increases the complexity. So to surmount the difficulties,a new theory should be used. The comparison of fuzzy logic and classical logic systems are given in the Table 2.3.

Table 2.3: Comparison of Fuzzy and Classical Logic

| | CLASSICAL | FUZZY |
|---|---|---|
| Truth | True, False | Not exactly true |
| Predicates | Mortal, Even | Tall, hot, beautiful |
| Modifiers | Not | Very, more or less |
| Quantifiers | Univ., Existential | Few, usually |
| Probabilities | 70%, 0.3 | Likely, around 0.3 |
| Possibilities | Possible, Imposs. | Quite impossible |

It is important to observe that fuzzy logic increases the number of possible values which are available in classical logic system. In this respect, fuzzy logic may be viewed as an extension of such systems rather than as a system of reasoning which is in conflict with classical systems.

### 2.5.1 Fuzzy Sets

The original interpretation of fuzzy set introduced by [57, 56] as a generalization of the classical set theory has been applied to the description of imprecise, vague notions. The most important assumption in representing a fuzzy object is instead of binary approach of classical set theory ($\mu_A(x): \quad \Omega \mapsto 0$ or $1$), fuzzy set theory ($\mu_A(x): \quad \Omega \mapsto [0,1]$) is defined. Therefore, various degrees of membership for all objects $x$ are allowed. That is, membership of an element to a set becomes a fuzzy concept which is a membership value. Thus the truth value of a proposition need not to have only the true or false values but any degree of truth or false is allowed.

For any $x \in \Omega$, the grade of membership $\mu$ of $x$ which belongs to fuzzy set $A$ is in the interval $[0,1]$. Thus, if $\Omega$ is a universe of discourse, we say $A$ is a fuzzy subset of $\Omega$.

For instance, the fuzzy term tall can be defined by the following fuzzy set:

| Height | Grade of Membership |
|---|---|
| 1.55 | 0 |
| 1.60 | 0.1 |
| 1.65 | 0.2 |
| 1.70 | 0.5 |
| 1.80 | 0.8 |
| 1.85 | 1 |

It is clear that a fuzzy set is an association between objects and decimals, a correspondence that assigns to a given object, one and only one number in the unit interval $[0,1]$.

A fuzzy set $A$ in a universe of discourse $\Omega$ is characterized by the function

$$\mu_A(x): \quad \Omega \mapsto [0,1]$$

$\mu_A(x)$ is called the membership function or grade of membership of $x$ in $A$.

The above example is symbolized as

$$tall : height \mapsto [0,1]$$

### 2.5.2 Operations on Fuzzy Sets

The basic operations are **union** ($\cup$), **intersection** ($\cap$) and **complement** ($\neg$), as in the classical set theory.

The membership function of the **union** $A \cup B$ is pointwise defined by

$$\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}, \qquad x \in \Omega.$$

The membership function of the **intersection** $A \cap B$ is defined by

$$\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}, \qquad x \in \Omega.$$

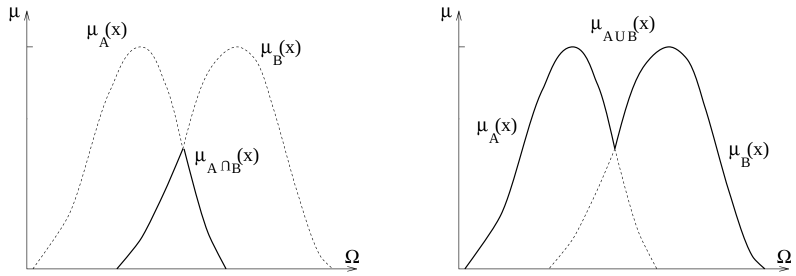


Figure 2.1 : Fuzzy Operators

The membership function of the **complement** of a fuzzy set $A$, is defined by

$$\mu_{\neg A}(x) = 1 - \mu_A(x)$$

The min-max operators are not the only operators that could have been chosen to model the intersection or union of fuzzy sets respectively [60, 55]. However the following reasons are the necessary properties provided by this operators:

1. The grade of membership in a compound fuzzy subset depends on the grade of membership in the elementary fuzzy subsets that form it, but on nothing else.

2. min and max operators are commutative, associative and mutually distributive operators.

$$\begin{aligned}
\min(a, b) &= \min(b, a) \\
\max(a, b) &= \max(b, a) \\
\min(a, \min(b, c)) &= \min(\min(a, b), c) \\
\max(a, \max(b, c)) &= \max(\max(a, b), c) \\
\max(a, \min(b, c)) &= \min(\max(a, b), \max(a, c)) \\
\min(a, \max(b, c)) &= \max(\min(a, b), \min(a, c))
\end{aligned}$$

3. The operators min and max are continuous and nondecreasing with respect to each of their arguments.

4. $\min\{1,1\} = 1$ and $\max\{0,0\} = 0$

### 2.5.3 Membership Functions

A fuzzy set can be characterized by a membership function. Membership function is one of the fundamental concepts in fuzzy set theory and any theory or research dealing with fuzzy information. Membership function distributes grades of membership to every object in the universe of discourse. In other words it describes the transition of the object from membership to nonmembership.

Usually membership functions are defined as continuous functions rather than discrete set of values. However they may be divided into discrete intervals for practical purposes. This choice of continuous functions is consistent with the characteristic of fuzzy knowledge. However, high precision of computation is insignificant and overaccurate computation is unnecessary. More information about membership function construction is given in [6].

### 2.5.4 Similarity Relations

To affect the result of certain operations in equivalent classes or eliminate the redundant tuples, an implied identity relation over a given domain $D_i$ is used in conventional databases. The identity relation in conventional databases is replaced by an explicitly declared similarity relation [56] in fuzzy databases.
Definition: A similarity relation, $s(x,y)$, for given domain, $D_i$, is a mapping of every pair of elements in the domain into the unit interval [0,1]:

$$s : D_i \times D_i \mapsto [0,1]$$

A similarity relation is a generalization of equivalence relations in that if $a,\ b,\ c \in D_i$, then $s$ is:

**reflexive** : $s(a,a) = 1$
**symmetric:** $s(a,b) = s(b,a)$
**transitive** : $s(a,c) >= \max_{\forall b \in D_i} \{\min[s(a,b), s(b,c)]\}$

Here is an example of a similarity relation for a finite scalar domain satisfying the conditions above:

$$\begin{array}{c c} & \begin{array}{cccc} w & x & y & z \end{array} \\ \begin{array}{c} w \\ x \\ y \\ z \end{array} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0.5 & 0.3 \\ 0 & 0.5 & 1 & 0.3 \\ 0 & 0.3 & 0.3 & 1 \end{bmatrix} \end{array}$$

where $D_i = \{w, x, y, z\}$

# CHAPTER III

# FUZZY COMPLEX OBJECTS

## 3.1 Introduction

In the work that is subject to this thesis the main driving motivation was the fact that a database that is 'natural' is favorable. To understand what 'natural' is, we have to look at our daily life concepts. One component of the world-picture that we are used to is the fact that objects and concepts are in nested relations with each other. For example we can speak about a car as

vehicle(car,made(1985,type(volkswagen,jetta)),plate(06AK789))

The example is easily extensible if one thinks of an 'owner' relation that relates the name of the owner, his/her age, some personal information, etc. Another component is that there are many objects or concept collections for which a clear description of membership cannot be defined. For these type of collections this concept is more or less a statistical concept which sometimes even exhibits cultural variations. For example we can speak about the same car, as being identified as an 'old' car or even as a 'young' car. Even sticking a certain cultural frame of reference, may not fix the problem. For example in our country, lets say 20% of people will call this car an 'old car', 50% of people a 'middle aged car', and 30% of them a 'young car'. Furthermore in natural language, we find numerous quantifiers (e.g. many, few some, about, almost, nearly, etc.) when conveying vague information. However, problem, solving, decision making should usually be done when information about the problem is partial or approximate. It is usually true that much world knowledge is neither totally certain nor totally consistent, especially representing personalities, physical features of individuals and judgments in situations such as medical, economic forecasting and personal evaluations need a system structure that permits fuzzy representations. In order to deal with objects which are inherently complex and fuzzy, we need to use an environment to represent and manipulate them. There exist some studies extending relational model and relational algebra to handle such objects. The representation power of logic programming is stronger than relational model and also each function expressable in relational algebra is expressible as a non-recursive logic program. We utilize logic programming here to handle fuzzy complex objects with incomplete properties. In this study, a logic programming paradigm, namely Turbo Prolog, is used to represent and implement the complex structure of the extended $NF^2$ data model [52] rather than extending the relational algebra. Before we explain all details of the implementation, we will give the formal definitions of the fuzzy complex object representation in logic.

## 3.2 Formal Description of Fuzzy Complex Objects

Here we give the formal definitions of the basic features of the logic system proposed. This system supports the representations and manipulation (or deduction) of complex entities along with their uncertain properties.

More specifically, the logic system that we describe here particularly concentrates on the representation of not only flat relations but also nested relations. The relations have attributes of various types. Therefore, the description here will first start from the definition of types. We are not interested in all possible types but only those that can be expressed through suitable definitions and therefore utilized on the developed system.

**Definition 1** A **type** $\mathcal{T}$ is defined as follows:

1. An **atomic-type** is represented by an **identifier** that is being used as a name, these *identifier*s must be renamings (denoted by '$\equiv$') of the types
   - integer
   - real
   - string

   Example: $\boxed{\textsf{namestring} \quad \text{where} \quad \textsf{namestring} \equiv \textsf{string}}$

2. A **set-type** $\mathcal{S}$, denoted as $\mathcal{S}\{t\}$. Here $\mathcal{S}$ is the naming of the set type, and each of the elements is of type $t$, where $t$ is of any type $\mathcal{T}$. All elements in the set must be of the same type, that is the set has to be *homogeneous*.

   Example: $\boxed{\textsf{courselist}\{\textsf{integer}\}}$

3. A **range-type**, denoted as $\mathcal{R}[t]$ where $\mathcal{R}$ is the naming of the range type. For a specific name $\mathcal{R}$, $t$ is either integer or real, or an atomic-type which is a renaming of one of these types.

Example: $\boxed{\textsf{age[integer]}}$

4. **Fuzzy-type** $\underline{\mathcal{F}}$, is one of the following two collections of two types:

   (a) integer | string

   (b) real | string

   *(Here '|' stands for the logical 'or' operator)*
   It is also allowed to use atomic-types which are renamings of the types used above. The collection

   integer | real

   is not allowed. This restriction comes from the way we deal with the fuzzyness. When we do not have a precise value of a specific type, then we supply a fuzzy linguistic value which is a type of string in our approach. In another word, we extend the domains of the types integer and real with some fuzzy linguistic string types. A fuzzy-type is represented by underlining it.

   Example: $\boxed{\underline{\textsf{salarytype}} \quad \text{where} \quad \underline{\textsf{salarytype}}\equiv\textsf{real}|\textsf{string}}$

5. **Null-type** $\overline{\mathcal{N}}$, which is a collection of two types, exactly the same as it is with the **Fuzzy-type** defined above. But, for the **null-type** the string values can only be one of the three strings "unknown", "notexist", "nil". A null-type is represented by overlining it.

   Example: $\boxed{\overline{\textsf{telnotype}} \quad \text{where} \quad \overline{\textsf{telnotype}}\equiv\textsf{string}}$

6. **Record-type** (or relation-valued type), denoted as $\mathcal{R}ec(s_1, \ldots, s_n)$. Here $\mathcal{R}ec$ is the naming of the **record-type**, and $s_1, \ldots, s_n$ are attributes. The type components of $s_i$'s are $t_i$, respectively, where $t_1, \ldots, t_n$ are of $\mathcal{T}$. *(Note that this definition is recursive and hence $t_i$ can also be of record-type. This definition enables us to represent arbitrary nested relations in this logic system.)*

   Example: $\boxed{\textsf{child}(\mathit{childname}{:}\textsf{namestring}, \mathit{childage}{:}\textsf{age[integer]})}$

**Definition 2** Now, we can define **attribute**s. Let $\mathcal{A}$ be a set of labels called attribute names.

An attribute $a$ is a pair $< n, t >$ where $n$ is an attribute name $n \in \mathcal{A}$ and $t$ is one of the types $\mathcal{T}$. $t$ is called the type of $a$. An attribute $a$ is represented in either one of the following ways:

1. $n : t$, where $n \in \mathcal{A}$ and $t$ is the syntactic denotation of one of the types of atomic-type, fuzzy-type, null-type and range-type. We call these attributes **base attributes**.

2. $n : s$, where $n \in \mathcal{A}$ and $s$ is a set-type. We call such an attribute a **set-valued attribute** and can include values of any kind of attributes.
3. $n : r(a_1, \ldots, a_k)$, where $n \in \mathcal{A}$, $r$ is a naming of a record-type and $a_i$ is any attribute, where $1 \leq i \leq k$. This kind of attribute is called **relation-valued attribute**.

**Example:**

| |
|---|
| person(*name*:string,*no*:integer,*tel*:$\overline{telnotype}$,<br>childlist{child(*childname*:string, *childage*:childage[integer])},<br>*height*:$\underline{\text{string}\|\text{real}}$) |

In this example the person relation has five attributes. The first and the second are named *name* and *no*, respectively, and both are of atomic type. The third attribute has the name tel which is of null type. The fourth attribute, which has the name childlist is a set type attribute. The elements of this set is of the record type child, that owns two attributes, named childname and childage. The attribute that is named as childname is of base type string and the second attribute, named as childage, is a range-type that was also named as childage where the range is defined over the integer type. (Note that the overlapping in the name childage is not a degeneracy, since in the first case it is a member of the set $\mathcal{A}$, hence an attribute name, and in the second it is a set-type naming). The last attribute of the above example is a fuzzy attribute, named as height.

**Definition 3** A **relation scheme** is a pair $< sch, S_r >$, denoted by $sch(S_r)$, where $S_r$ is a finite list of attributes with distinct attribute names. We refer to a relation scheme simply by $sch$ if $S_r$ is understood.

**Definition 4** Instances of types are called **terms**, and we call the set of all terms of a type $t$ as $\mathcal{P}(t)$.

An element of $\mathcal{P}(t)$ is either **simple** or **complex**. If it is a **simple** term, then one of the following holds:

1. it is a constant $c$ of type $t \rightarrow c \in \mathcal{P}(t)$
2. it is a variable $v$ with a value of type $t \rightarrow v \in \mathcal{P}(t)$

If it is a **complex** term, than one of the following is true:

1. $\mathcal{S}\{p\}$ is a set-type
   $\wedge\ \forall i \in \{1, \ldots, n\} \quad \wedge\ \ x_i \in \mathcal{P}(p) \rightarrow \mathcal{S}\{x_1, \ldots, x_n\} \in \mathcal{P}(\mathcal{S})$
2. $\mathcal{R}[m]$ is a range-type $\wedge\ \ n, n' \in \mathcal{P}(m) \rightarrow \mathcal{R}[n, n'] \in \mathcal{P}(\mathcal{R})$

3. $\mathcal{R}ec(s_1,\ldots,s_n)$ is a record-type
   $\wedge\ \forall i \in \{1,\ldots,n\} \quad \wedge \quad r_i \in \mathcal{P}(\text{type of(s}_\text{i}))), m_i \equiv \text{name of(s}_\text{i}) \rightarrow$
   $f(m_1:r_1,\ldots,m_n:r_n) \in \mathcal{P}(\mathcal{R}ec)$.
4. $\underline{x}$ is a constant value or a variable of a fuzzy-type $\underline{\mathcal{F}} \rightarrow \underline{x} \in \mathcal{P}(\underline{\mathcal{F}})$
5. $\overline{x}$ is a constant value or a variable of a null-type $\overline{\mathcal{N}} \rightarrow \overline{x} \in \mathcal{P}(\overline{\mathcal{N}})$

**Example:**

```
person(name:"meltem turhan",no:501,tel:noinfo,
        childs: childlist{child(childname:"ayse",childage:childage[3,5]),
                          child(childname:"ali",childage:childage[7,7])},
        height:1.65)
```

This is an example for instances of types, namely a term. Here Person "meltem turhan" with employee number 501 has two children, namely, "ayse" and "ali". Ayse is in the age range of [3-5] and ali's age is exactly 7. The null-valued attribute tel has a value "noinfo". The last attribute of the example is a fuzzy one with a real value instance 1.65.

**Definition 5** An **atom** is an element of $\mathcal{P}(\mathcal{R}ec)$ where $\mathcal{R}ec$ is a record-type (relation valued-type).

**Definition 6** A **complex rule** is a formula of the form

$$q \leftarrow \alpha_1,\ldots,\alpha_n$$

where $q$, the head of the rule, is an atom and $\alpha_1,\ldots,\alpha_n$ the body of the rule, is a conjunction of atoms.

**Example:**

```
friend(name1:X,name2:Y) ← likes(name1:X,name2:Y),
                          person(name:X,hobies:h{H1,H2}),
                          person(name:Y,hobies:h{H1,H2}).
```

*If two person have same hobies and they like each other, then they are inferenced as being friend.*

**Definition 7** A **complex fact** is an atom with arguments that are ground (variable free).

**Example:**

`person(name:"meltem turhan",hobies:h{"reading","tennis"}).` Person has an atomic type attribute name "meltem turhan" and set type attribute hobies with instances "reading" and "tennis".

**Definition 8** If $q$ is defined only by facts, the $q$ is called **database predicate**, otherwise it is a **derived predicate**. 'friend' is an example of a derived predicate.

**Definition 9** A **goal**, $G$, is a complex rule with empty head or a conjunction of atoms.
**Example:** ?- friend(*name1*:X,*name2*:Y). *This goal queries database for predicate friend with parameters X and Y.*

**Definition 10** A **complex logic program** is a finite set of complex rules.

**Definition 11** A **complex logic system** consists of two parts; one is the intentional database (IDB) consisting of all rules defining derived predicates and the extensional database (EDB) consisting of all ground facts namely complex facts (also called **base predicates**).

**Definition 12** A **complex query** is a pair $Q =< G, sch >$, where $G$ is a complex goal and $sch$ is a logic schema.

Example:

<*name*=ahmet, person(*name*:string,*no*:integer,*tel*:$\overline{telnotype}$)>

## 3.3 Fuzzy Complex Object Description in Prolog

For the implementation the Turbo-Prolog dialect is chosen. Although a detailed explanation elaborating on this choice will follow in the next chapter, it is worthwhile to stress on one property of Turbo-Prolog, namely its being strongly typed. This is not common to all Prolog implementations and suits for our purpose quite well.

Before displaying the details about the implementation the type structures and instance structures of the implementation will be given in a tabular form. The reason for using both the tabular form of the extended $NF^2$ data model and corresponding logic representation is to point out the close relationship between them [50]. Then, the logic form will be used instead of developing extended $NF^2$ relational algebra for processing the fuzzy complex objects.

**person**

| name | no | tel | age | address | height | distance | tp |
|---|---|---|---|---|---|---|---|
| meltem turhan | 501 | $\overline{2230812}$ | [23,26] | bahcelievler | 1.65 | $\underline{\text{near}}$ | 1 |
| adnan yazici | 5005 | $\overline{2101000}$ | [29,30] | eryaman | $\underline{\text{medium}}$ | $\underline{\text{far}}$ | 1 |
| tayfun hiz | 200 | $\overline{noinfo}$ | [19,23] | kizilay | $\underline{\text{tall}}$ | 10 | 2 |

**Prolog representation:**

```
person(name,no,tel,age,address,height,distance,tp)
person("meltem turhan",e(501),k(2230812),age(23,26),
        "bahcelievler",k(1.65),uk("near"),1)
person("adnan yazici",e(5005),k(2101000),age(29,30),
        "eryaman",uk("medium",uk("far"),1)
person("tayfun hiz",s(200),uk("noinfo"),age(19,23),
        "kizilay",uk("tall"),k(10),2)
```

As it can be seen from the example, the Prolog representation is very similar to the type description. Differences are in the fuzzy, null and set type representations. Both for fuzzy and null types k() and uk() predicates are used. In the first tuple, for example, name of the person is "meltem turhan" with employee number 501, her telephone number is known and it is 2230812, her age is between 23 and 26. Her address is bahcelievler. Height is known and it is 1.65. Distance is a fuzzy value "near".

## employee

| e-no | salary | childl | |
|---|---|---|---|
| | | **childname** | **childage** |
| 501 | 6.0 | {ayse<br>ali | [5,10],<br>[7,7]} |
| 5005 | <u>high</u> | { | } |

**Prolog representation:**

```
employee(501,k(6.0),[childl("ayse",cage(5,10)),child("ali",cage(7,7))])
employee(5005,uk("high"),[])
```

Salary of the employee with number 501 is known and it is 6.0 Million. The third attribute of this relation is a list of another relation child with two attributes childname and childage.

## student

| sno | courses | mdept | | |
|---|---|---|---|---|
| | | **deptname** | | **deptno** |
| | | **facname** | **dname** | |
| 200 | {112,230,316,405} | eng | comp | 71 |

**Prolog representation:**
student(200,[112,230,316,405],mdept(deptname("eng","comp"),71))

The notation used in the student relation is the same as employee. The student number is just an integer. The second argument is a list of courses that the student has taken. Last attribute is a two level nested relation containing the department name, code and faculty code.

The **types** $\mathcal{T}$ of this implementation are defined in the DOMAIN part of Turbo-Prolog. Here is how this is done:

**a)** An atomic type is defined by one of the standard domains string, integer, real, namely by one the followings

DOMAIN $identifier =$ string
DOMAIN $identifier =$ integer
DOMAIN $identifier =$ real

**b)** A set-type $\mathcal{S}\{p\}$ is defined by the notation

DOMAIN $\mathcal{S} = s$*

**c)** A range-type $\mathcal{R}[n]$ is defined by the notation

DOMAIN $\mathcal{R} = \mathcal{R}(identifier,identifier)$

where prior to this definition the $identifier$ was defined to be a renaming of the standard domains **integer**, **real**.

**d)** A record-type $\mathcal{F}(s_1{:}p_1, \ldots, s_n{:}p_n)$. is defined by the notation

DOMAIN $\tilde{\mathcal{F}} = \mathcal{F}(p_1, \ldots, p_n)$.

Here $\tilde{\mathcal{F}}$ can be an abbreviation of $\mathcal{F}$ as well as $\mathcal{F}$ itself. $p_i$'s are $identifier$s that are namings of any type. *(Here is a slight difference from the formal definition: an attribute which is not an atomic, has to be named (by a prior* DOMAIN *declaration), and that name identifier has to be used in the $p_i$ position).* Also note that the attribute names $s_i$'s are not present in the DOMAIN declarations. This is so because in Prolog the tuples are represented by predicates, and predicates have their argument slots distinct. So, there is no need to identify an attribute of a relation refers to the predicate in Prolog by a special label, since the position of the attribute is known and fixed among the predicate's arguments. But at the implementation level, it is more suitable to provide the user with such naming (so the queries, for example, can be made by some field name, and not a argument slot). For this purpose the implementation associates a unique 'keyword' at the user interface level for each attribute in the database.

**e)** A **fuzzy-type** $\underline{\mathcal{Z}}$ is defined by the notation

$$\mathsf{DOMAIN} \quad \mathcal{Z} = \mathsf{k(real);\ uk(string)}$$

Which says that a fuzzy type attribute is either a real or a string, and depending on which of these it is, it is packed in a predicate k or uk, respectively.

**e)** A **null-type** $\overline{\mathcal{N}}$ is defined by the notation

$$\mathsf{DOMAIN} \quad \mathcal{N} = \mathsf{k(real);\ uk(string)}$$

Which says that a null type attribute is either a real or a string, and depending on which of these it is, it is packed in a predicate k or uk, respectively.

Above $\mathcal{S}$, $\mathcal{R}$, $\mathcal{F}$, $\tilde{\mathcal{F}}$, $\mathcal{Z}$ $\mathcal{N}$ under the DOMAIN declarations are *identifiers*.

In Turbo-Prolog it is possible (and sometimes necessary) to name multi-types with a single *identifier*. The **(e)** and **(f)** items above are good examples of this.

## 3.4 Implementation Examples for Description

In the previous section the basic Prolog implementation strategies were stated. In this section we pose some examples from the implementation for clarity.

AN ATOMIC TYPE

**domains**
name = string
id_no = integer
salary = real

**database**
person(name,id_no,salary)

A SET-TYPE

**domains**
s_name = string
course_no = integer
course_list = course_no*

**database**
student(s_name,course_no,course_list)

**A Range-Type**

**domains**
name = string
lower = integer
upper = integer
age = age(lower,upper)

**database**
person(name,age)

**A Record-Type**

**domains**
name = string
no = integer
child_name = string
child_age = integer
child = child(child_name,child_no)

**database**
person(name,no,child)

**A Fuzzy-Type**

**domains**
name = string
height = k(real),uk(string)
salary = k(real),uk(string)

**database**
person(name,height,salary)

**A Null-Type**

**domains**
name = string
tel_no = k(real),uk(string)

**database**
person(name,tel_no)

In the system combinations of complex fuzzy attributes are also accepted.

**Range Inside the Relation**

**domains**
no = integer
lower = integer
upper = integer
age = age(lower,upper)
child_name = string
child = child(child_name,child_age)

**database**
employee(no,child)

**List of Relation**

**domains**
no = integer
lower = integer
upper = integer
child_age = child_age(lower,upper)
child_name = string
child = child(child_name,child_age)
child_list = child*

**database**
employee(no,child_list)

Two level Nesting of Relation

**domains**
s_name = string
d_name = string
fac = string
dept_no = integer
dept_name = dept_name(fac,d_name)
mdept = mdept(dept_name,dept_no)

**database**
student(s_name,mdept)

## 3.5 Information Retrieval & Evaluation of Fuzzy Complex Objects

As it is expected, information stored in the forms explained in the preceding sections, has to be retrieved for various purposes. Query and deletion are two of them. Now the questions is: "How is the solution set formed from a given a form of criteria ?". Before answering this question a more essential question has to be posed. "What kind of questions are possible ?"

- A query is made over any attribute of any relation in the database.
- Three comparison operators are provided (implemented) as:
  - '=' The 'equality' test.
  - '<' The 'less than' test.
  - '>' The 'greater than' test.

  The left hand side of the comparison is the attribute name subject to the query, and the right hand side is a constant value. This value will be allowed to be a fuzzy constant if that attribute is a fuzzy one.
- The logical combinations of two queries by the operators
  - AND
  - OR

  is also a valid query. In this case the queries joined by these logical operators will be called 'sub-queries'.

The possibility that the queries can be over fuzzy attributes and that logical operations are allowed requires that all the results coming form the sub-queries, and hence the queries, must carry a fuzzy information. Furthermore, some binary fuzzy set operations must be defined as well. Therefore the system is designed such that all the query results are fuzzy sets. The logical AND operations that join two sub-queries will cause an **intersection** of the result sets coming from these sub-queries. Similarly a OR operation means a **union** of the result sets of the sub-queries.

A fuzzy set is not much different than a normal set. It is a collection of some entities. An order among the entities is not defined. The difference lies in the concept of 'being an element of the set'. In the classical set theory an entity is either the member of a set or not. In the fuzzy set theory one speaks of the degree of being a member of a set. So, each entity has a 'membership value' (which will be abbreviated as MV, from now on) concerning a fuzzy-set which tells *how strong* it is a member of the set. We will assume (without loss of generality) that this MV is represented by a real value in the range [0,1]. The fuzzy approach incorporates the classical sets. A classical set can be considered as a fuzzy set where all of its members have MVs that are equal to 1. Any entity that is not a member of the set has an MV of 0. On the following pages we introduce the MV values defined by means of functions over the numerical domains of the fuzzy concept *(e.g. if the fuzzy concept is 'distance' then the domain is distances in kilometers).* As said, these functions are mappings into the real numbers in the range [0,1] which are the MV values.

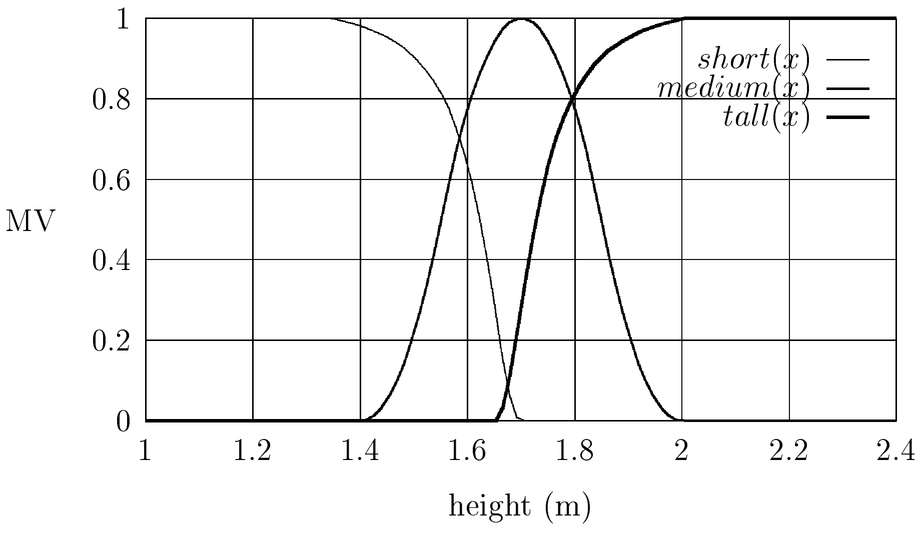


Figure 3.1 : Fuzzy Membership Functions of Height

Short

$$short(x) = \begin{cases} 1 & x < 1.35 \\ 1.05 \times \left[1 + \left(\dfrac{c}{x - \alpha}\right)^2\right]^{-1} & 1.35 \leq x < \alpha \\ 0 & x \geq \alpha \end{cases}$$

$$\alpha = 1.7\ , \qquad c = 1.2$$

**MEDIUM**

$$medium(x) = \begin{cases} 0 & x < 1.4 \\ c(x-1.4)^2 & 1.4 \le x < 1.55 \\ 1 - c(x-1.7)^2 & 1.55 \le x < 1.85 \\ c(x-2)^2 & 1.85 \le x < 2.0 \\ 0 & x \ge 2.0 \end{cases}$$

$$c = 22.222$$

**TALL**

$$tall(x) = \begin{cases} 0 & x \le \alpha \\ 1.05 \times \left[1 + \left(\dfrac{c}{x-\alpha}\right)^2\right]^{-1} & \alpha \le x < 2.0 \\ 1 & x \ge 2.0 \end{cases}$$

$$\alpha = 1.65\ , \qquad c = 0.08$$

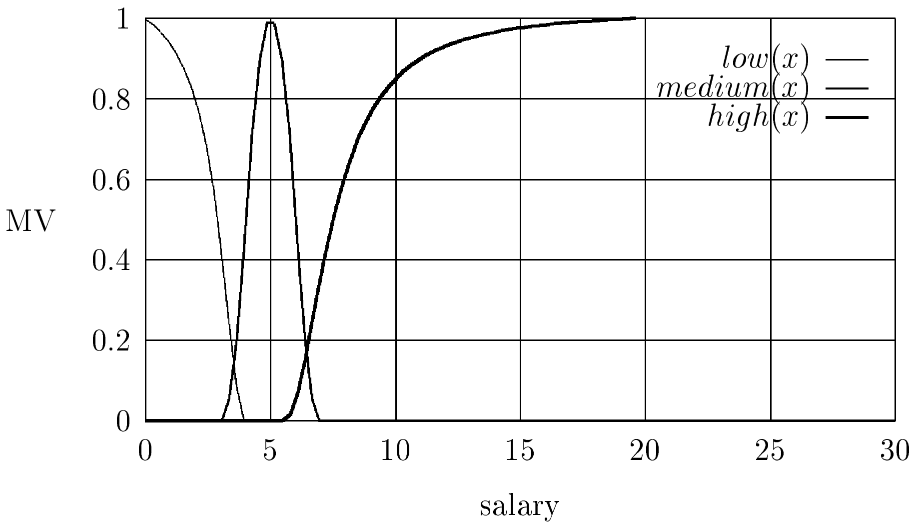


Figure 3.2 : Fuzzy Membership Functions of Salary

Low

$$low(x) = \begin{cases} 1.087 \times \left[1 + \left(\dfrac{c}{8 - x - \alpha}\right)^2\right]^{-1} & x < \alpha \\ 0 & x \geq \alpha \end{cases}$$

$$\alpha = 4\ , \qquad c = 1.2$$

Medium

$$medium(x) = \begin{cases} 0 & x < 3 \\ 2\left(\frac{x-3}{c}\right)^2 & 3 \leq x < 4 \\ 1 - 2\left(\frac{x-5}{c}\right)^2 & 4 \leq x < 6 \\ 2\left(\frac{x-7}{c}\right)^2 & 6 \leq x < 7 \\ 0 & x \geq 7 \end{cases}$$

$$c = 2$$

High

$$high(x) = \begin{cases} 0 & x \leq \alpha \\ 1.02 \times \left[1 + \left(\frac{c}{x-\alpha}\right)^2\right]^{-1} & x > \alpha \end{cases}$$

$$\alpha = 5.5\ , \qquad c = 2$$

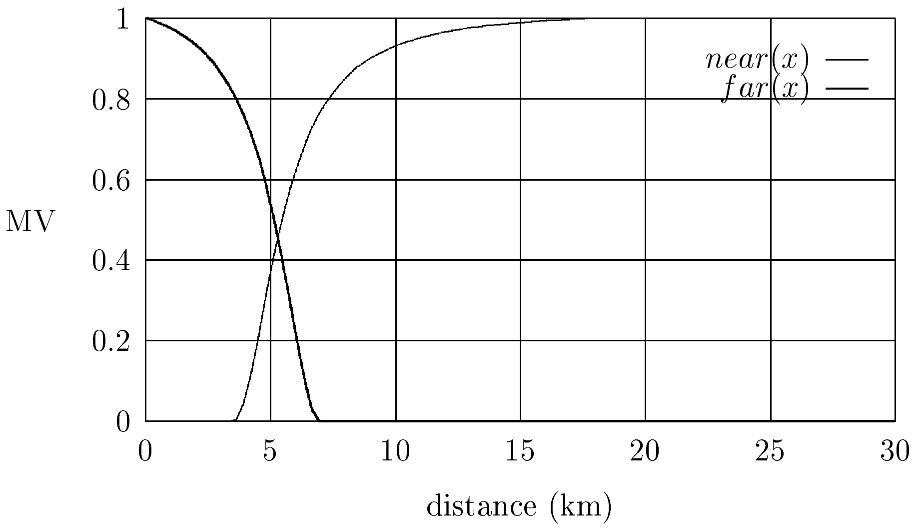


Figure 3.3 : Fuzzy Membership Functions of Distance

**NEAR**

$$near(x) = \begin{cases} 1 & x < 1.35 \\ 1.05 \times \left[1 + \left(\dfrac{c}{x-\alpha}\right)^2\right]^{-1} & 1.35 \leq x < \alpha \\ 0 & x \geq \alpha \end{cases}$$

$$\alpha = 1.7 \;, \qquad c = 1.2$$

**FAR**

$$far(x) = \begin{cases} 0 & x \leq \alpha \\ 1.05 \times \left[1 + \left(\dfrac{c}{x-\alpha}\right)^2\right]^{-1} & \alpha \leq x < 2.0 \\ 1 & x \geq 2.0 \end{cases}$$

$$\alpha = 1.65 \;, \qquad c = 0.08$$

The two basic operations defined on classical sets, namely **union** ($\cup$) and **intersection** ($\cap$) have their counterparts in the fuzzy set theory, as well. As expected these definitions for the fuzzy sets reduces to the ordinary *union/intersection* operations if the sets involved have elements with MVs of 1 or 0 only. The **fuzzy union** of two fuzzy sets $A$ and $B$ is defined as:

$$A \cup B \triangleq \{x \mid MV_{A \cup B}(x) = \max(MV_A(x), MV_B(x))\}$$

Similarly, the **fuzzy intersection** of two fuzzy sets $A$ and $B$ is defined as:

$$A \cap B \triangleq \{x \mid MV_{A \cap B}(x) = \min(MV_A(x), MV_B(x))\}$$

Of course at execution time it is nonsense to hold the entities with MV of 0, in the representation of the set. These entities are just discarded.

So, the fuzzy evaluation constitutes of two main parts.

1. evaluation of the comparison(s)

2. joining the results of the sub-queries (if there exist more than one) by the appropriate fuzzy set operations **union/intersection**.

The second entry is explained, already. But the first one has to be explained for its various possibilities. The left hand side of the comparison is the attribute name and the right hand side is a constant value. Where the comparison operation is one of '=', '<', '>'. Consider a specific query of this form. If this query is over an attribute that may admit fuzzy values, than it is true that

1. The user can have entered a fuzzy instance value as well as a non-fuzzy instance value on the RHS.

2. For each database entry this attribute can have a fuzzy value or a non-fuzzy value.

For simplicity we will call non-fuzzy values as *deterministic*. So, all the four combinations are possible:

| **LHS** *Database entry's attribute value* | **RHS** *User's query value* |
|---|---|
| Deterministic | Deterministic |
| Deterministic | Fuzzy |
| Fuzzy | Deterministic |
| Fuzzy | Fuzzy |

The system is so designed that for each entry in the database the attribute value and the query value are submitted to an algorithm which determines the MV value, for this pair. This algorithm acts in three different ways for the following three categories:

1. Both are deterministic.
2. One is deterministic the other is fuzzy.
3. Both are fuzzy.

Below the behaviors of the algorithm for these categories are described:

**Both are deterministic :** For the '$<$' and '$>$' operators the values are compared, and depending on this result a MV of 1 or 0 is returned.

For the '$=$' case the action changes. *'equality'* is interpreted as 'being *about*'. Following the definition in [33], a function that defines MV for *'about'* is introduced:

$$MV(x) \mid_{about\ x_0} = e^{-c(x-x_0)^2} \quad , \qquad (c > 0)$$

MV that stays below TV (TV is the Threshold Value and is used to serve as a comparison value to accept/reject a solution. This concept is explained below in more detail) are taken as 0.

**One is deterministic the other is fuzzy :** In this case we have a fuzzy value *(*far*,* short*,* tall*)*. For each of these fuzzy values, an empirically determined function is defined. These functions are mappings from the numerical value domain (the fuzzy value is about), onto the range $[0, 1]$ of real numbers (which stands for the MV values). So, for example, if $f_{short}(x)$ is the function associated with 'shortness', and yields

$$f_{short}(1.55) = 0.7$$

we would say that the height value: 1.55 meter is a member of the fuzzy set '*short*' with an MV value of 0.7.

This explains what is done for the *'equality'* test: The deterministic value is plugged into the specific function associated with the fuzzy value, and the MV returned by the function is used. Now, it is quite possible that some evaluations will yield results which are significantly **small**, hence a mechanism has to exist to reject this type of MV values. For this purpose a cut-off value (we will call it the 'Threshold Value', and abbreviate it by TV) is introduced. Threshold value is a user defined value to reject or accept tuples according to their membership values. Any MV value that cannot exceed the TV will be taken as 0 (zero) and so that specific element will have no chance to exist in the fuzzy result set. This means TV should be less than or equal to MV.

For the '<' and '>' comparisons the evaluation scheme changes. Let us take the '<' comparison first. Assume the query was of the form

$\Omega < \xi$ (For example: distance<far)

and a candidate $x_c$ value is known. The algorithm will proceed as follows:

- Find the area $S_{x_c}$ that is under the $MV_\xi$ curve in the range $(x_c, \infty)$
- Find the area $S_\Omega$ that is under the whole $MV_\xi$ curve, i.e. in the range $(-\infty, \infty)$
- Calculate $S_{x_c}/S_\Omega$ and compare this value with the TV. If it exceeds the TV, accept it and return it as the MV value for $x_c$, otherwise reject this candidate.

The idea behind this can be phrased as:

> *The resulting MV of the comparison is the sum (integral) of the MV values for that specific fuzzy value that are in the range not satisfying the comparison criteria divided by the whole area under the MV curve.*

For the '>' comparison, the process is similar. Assume the query was of the form

$\Omega > \xi$ (For example: distance>near)

and a candidate $x_c$ value is known. The algorithm will proceed as follows:

- Find the area $S_{x_c}$ that is under the $MV_\xi$ curve in the range $(-\infty, x_c)$
- Find the area $S_\Omega$ that is under the whole $MV_\xi$ curve, i.e. in the range $(-\infty, \infty)$
- Calculate $S_{x_c}/S_\Omega$ and compare this value with the TV, if it exceeds the TV accept it and return it as the MV value for $x_c$, otherwise reject this candidate.

**Both are fuzzy :** All the comparison cases are treated similarly. For each comparison there exist a 'similarity matrix'. The two indices of the matrix take their values from the same domain, namely the possible fuzzy values. (e.g. far, near). The entries are precalculated, using the technique described by [57], and correspond to the MVs directly. This value is retrieved from the matrix and compared to the TV. If it exceeds than the candidate is accepted with that MV else rejected.

**HEIGHT**

$$\begin{array}{r c}
 & \begin{array}{ccc} short & medium & tall \end{array} \\
\begin{array}{r} short \\ medium \\ tall \end{array} & \begin{bmatrix} 1.0 & 0.6 & 0.2 \\ 0.6 & 1.0 & 0.5 \\ 0.5 & 0.2 & 1.0 \end{bmatrix}
\end{array}$$

**SALARY**

$$\begin{array}{r c}
 & \begin{array}{ccc} low & medium & high \end{array} \\
\begin{array}{r} low \\ medium \\ high \end{array} & \begin{bmatrix} 1.0 & 0.5 & 0.1 \\ 0.5 & 1.0 & 0.6 \\ 0.1 & 0.6 & 1.0 \end{bmatrix}
\end{array}$$

**DISTANCE**

$$\begin{array}{r c}
 & \begin{array}{cc} near & far \end{array} \\
\begin{array}{r} near \\ far \end{array} & \begin{bmatrix} 1.0 & 0.2 \\ 0.2 & 1.0 \end{bmatrix}
\end{array}$$

# CHAPTER IV

# IMPLEMENTATION

## 4.1 Introduction

The implementation language is chosen as Prolog. In particular the Turbo-PROLOG 2.0 brand is used and the professional toolbox provided herewith is made use of. The reasons for this choice will be stated and evaluated later in this chapter. In the forthcoming sections a general introduction to Prolog will be followed by the horizontal structure of the implementation program, and then the vertical structures of it. The ingredients of the program will be explained in details.

## 4.2 Prolog

Prolog is a logic programming language used for solving problems that involve *objects* and the *relationships* between objects. The foundations of the language goes back to 1970 but the boom of its usage starts in 1982. Since then it is extensively used in

- Relational databases
- Mathematical logic
- Abstract problem solving
- Natural language processing
- Design automation
- Symbolic algebraic computation
- Many other areas of AI

Prolog as a declarative programming language has a tempting appearance of giving something for nothing: the programmer simply states facts and rules which are relevant to the solution, and this information is then used as a program to solve the problem without the need for more advice on how to handle the information. Actually this is true for short Prolog programs. For those which are not quite short, Prolog often shows a behaviors which illustrates what can be called an improvement of the popular saying:

> "You can't get something for nothing, except if you're prepared to wait for an infinite time to get it"

Because any system for declarative programming must spends most of its time searching for solutions. These solutions are consistent with the facts and rules which are given. To cut the size of the search space to a minimum for any specific problems, one needs a very careful declaration of the problem (what it is, and what it is actually not!) and sometimes old-fashioned procedural programming is needed as well. As the main attraction of declarative computing is the avoidance of procedural methods, one may have to live with this type of inefficiency as a price for economy in the effort of programming. Despite this fact, Prolog is a practical and efficient (bound by the words above!) implementation of many aspects of "intelligent" program execution, such as non-determinism, parallelism and pattern-directed procedure call. Generally speaking (this does not hold exactly for Turbo-Prolog), Prolog provides a uniform data structure, called the *term*, out of which all data, as well as Prolog programs, are constructed. A Prolog program consists of a set of clauses, where each clause is either a fact about the given information or a rule about how the solution may relate to or be inferred from the given facts.

## 4.3 About Turbo-Prolog

The Borland product Turbo-Prolog 2.0, which was chosen as the implementation language for this work, has several differences from the traditional Prolog interpreters/compilers. The main difference is the design philosophy. To be fair and objective we will state the pros and cons of this dialect, and justify our choice.

### 4.3.1 Pros of Turbo-Prolog

**Is a compiler.** Turbo-Prolog is a Prolog compiler available in the DOS environment. The code that it generates compiles with the DOS's OBJ standard, hence is linkable with OBJ generated using other high-level languages. It is for sure that compiled programs run much faster then the interpreted ones. Furthermore Turbo-Prolog, is quite a good compiler in the sense of producing efficient code. For example, tail-recursion removal, efficient use of (cut) information, extensive flow analysis are performed.

**Provides many useful system predicates.** The built-in predicate library is waste. The user has all the low level control for text/graphics windows to hand; a full implementation of $B^+$ trees and data chains are present. Turbo-Prolog provides even a screen editor callable as a predicate. Also routines for run time error detection and exception handling is extensively provided.

**Has an integrated development environment.** The user, as it is with all Borland products, has the programming facility of a fully menu driven user interface to the compiler: All the editing, compiling, running, debugging are done in a single window environment where all these different task have their own windows. The system also provides the user with an on-line help facility.

**Allows modular programming.** Turbo-Prolog provides the concept of 'Project'. Here the main programming task can be subdivided into smaller tasks. Then these sub-tasks can be programmed into rather stand alone packages, which have some of their types and predicates explicitly declared for interfacing and all the other programming bits are private to that package. These packages are called 'modules'. When these modules are compiled each will generate an object file.

**Provides a programmer's toolbox.** A software package for Turbo-Prolog, commercially available, implements various teddy but useful building blocks. Among these are pull-down menus, pop-up windows, hot-keys, data entry screen design/usage.

### 4.3.2 Cons of Turbo-Prolog

**Is a strongly typed language.** The way Turbo-Prolog approaches the prolog philosophy is against the tradition. All predicates have to be carefully defined for their arguments. Furthermore multitypes are forbidden. This has obscure results, like the necessity that lists have to be homogeneous. Renaming of basic data types are not recognized to be identical.

**Does NOT inherit rule-assertion, operators**

- At run time the program cannot assert rules, dynamically. Although facts can be asserted.

- The Prolog parser is not open to the user. Hence, the classic predicate that introduce new infix, prefix, postfix operators, namely `op` is not present.

**Artificial definition discrimination are forced.** To help the compiler's code generation predicates have to be declared 'deterministic' or 'non deterministic', 'global' or 'local', etc. Furthermore facts that may be added/retracted at run-time have to be defined separately as 'global database' or 'local database'.

**Some standard predicates do not exist and some functions are different.** The '=..', so called (univ) operator/predicate does not exist, arithmetic operators or automatically evaluated, the '`is`' operator is substituted by '=' and does not allow uninstantiated variables in arithmetic expressions.

### 4.3.3 Why was Turbo Prolog chosen?

The reasons can be itemized as follows:

- The strong data typing is the most important reason to use Turbo Prolog as far as our specific implementation is concerned. Since, the theoretical study done in this work is heavily based on type definitions a possibility of a one-to-one implementation of the defined type structure is considered as an asset.
- It is providing facilities for building a reasonable user interface since it has its own toolbox.
- The department's software infra-structure, in the period of implementation, was not providing a professional Prolog *(like Quintus, which is now available)* on workstations. So, a PC implementation was forced. Turbo-Prolog was the only compiler available on the DOS environment, the other interpreters available on the PC environment were non-professional freewares.These interpreters are good for trying out some small test programs since their resource limitations are inappropriate for a relatively large project implementations.

## 4.4 Layout of the Program

As it is stated above, Turbo-Prolog 2.0 allows modular programming where each module is compiled separately into object files and then linked by the operating system linker into an executable binary. By default all names used in a module are local. Turbo-Prolog programs communicate across module boundaries using the predicates defined in the **global predicates** and **global database** sections. The implementation consists of the following modules (each module is stored in a file with the same name and a following extension of '.pro':

1. addition
2. query_del
3. parseval
4. toolbox
5. main

The system architecture is given in Figure 4.1 .

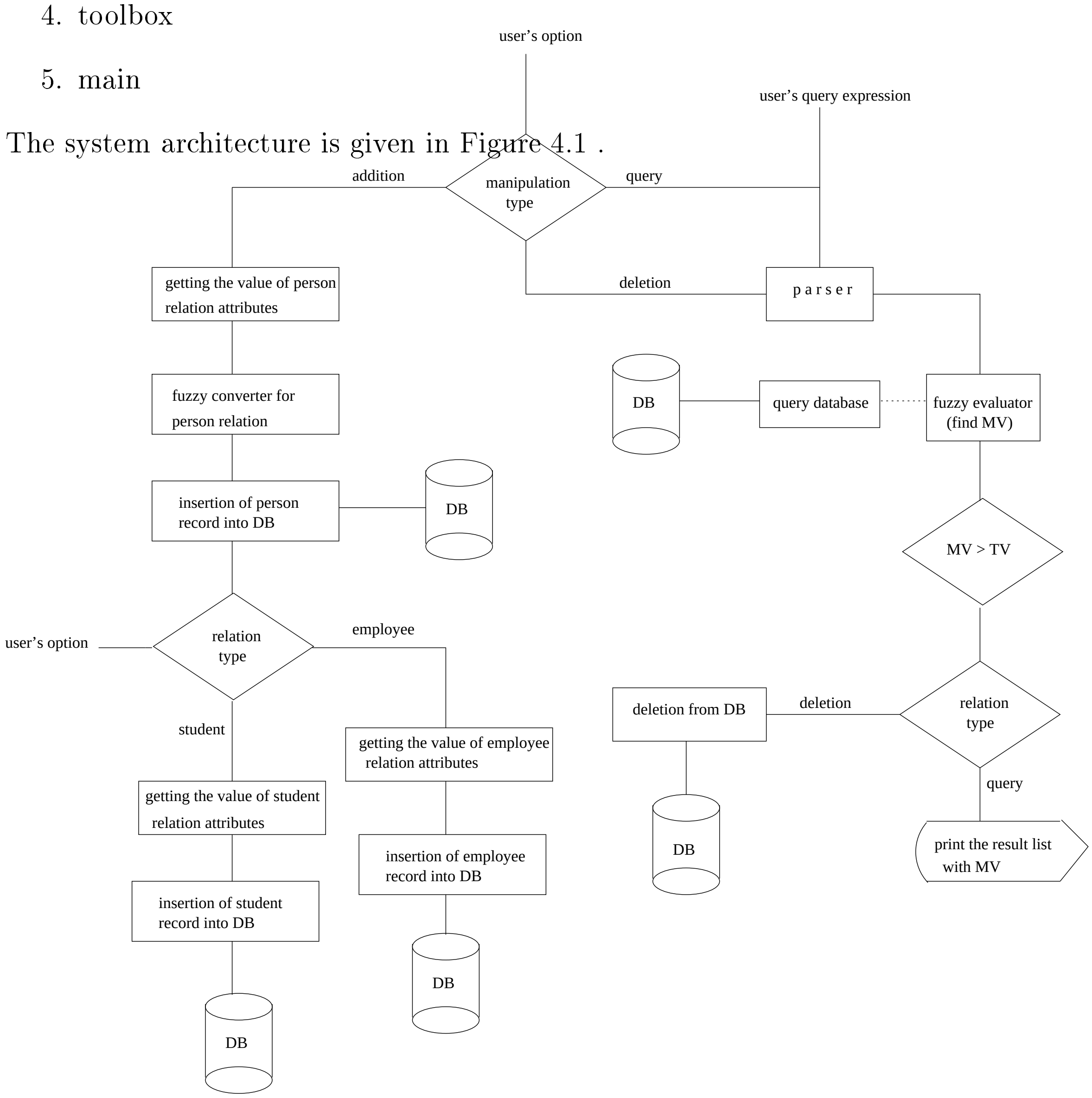


Figure 4.1 : Architecture of the System

**addition** New record addition to the database is done here. It prompts the user with a fill-in screen and gets the record information. After making the necessary conversions it is inserted to the database.

**query_del** query and deletion, save/retrieve of the database is done here, also the fuzzy handlings are mainly located in this module.

**main** contains the 'goal' clauses of Turbo-Prolog. In a sense it is the main execution block of the implementation.

**parseval** is the parser+evaluator interface for the user queries and the database engine. The user input to this module is an expression which complies with a precedence grammar defined in the previous chapter. This module parses the user input, converts it to prolog predicates and then performs the evaluation making use of the database access and process predicates defined in the dbase module.

**toolbox** contains predicates for text window handling, pull-down menus, data entry/update screen handling.

### 4.4.1 The Database Predicates

The program saves/retrieves its state to/from an ASCII file named: `data.dba`. This file contains instances of three basic relations, also various parameters like the threshold value is stored here. The three basic predicates of the database are as follows:

- **person** defines a person type relation, the attributes are as follows:

  person(*name,id-no,tel-no,age,address,height,distance,type*)

  *name* : string
  : Is the name of the subject person.

  *id-no* : integer
  : Is a unique number which serves also as the 'join attribute' with the other instances in the database. It is the primary key value, hence redundant use is prohibited.

  *tel-no* : k(real) or uk(string)
  : Is either a certain number or null value. if it is certain then this value is embedded in a predicate k() (k stands for known, and this choice is made for all the attributes which can be certain/null). Similarly if it is a null value then a string that represents the null instance is embedded in a predicate uk(). For this *tel-no* the possible null instances are

- "noinfo"
- "notexist"
- "unknown"

*age* : age(integer,integer)
: This field holds a range information. Under the predicate age. The first is the lower bound value of the range, and the second is the upper bound.

*address* : string
: This field is intended to hold the address of the person.

*height* : k(real) or uk(string)
: Holds the information of the height of the person. Similar to *tel-no* this field can also be fuzzy valued. If it is a certain value then it is packed in the predicate k(), otherwise (it is a fuzzy instance) and it is packed in the predicate uk(). The possible fuzzy instances that can be used here are:

  - "tall"
  - "medium"
  - "short"

*distance* : k(real) or uk(string)
: Holds the 'distance' the person must travel each day to reach home. Very like *height*, this field can be fuzzy valued, as well. Possible fuzzy instances (embedded in uk()) are:

  - "far"
  - "near"

*type* : 1 or 2
: For internal use of the program.

- **employee** defines a employee type relation, the attributes are as follows:

  employee(*id-no,salary,childlist*)

  *id-no* : string
  : Has the same attribute of the person relation.

  *salary* : k(real) or uk(string)
  : Similar to height or distance, can hold certain or fuzzy values. Possible fuzzy instances are:

    - "low"
    - "medium"

- "high"

*childlist* : [child(string,integer),...,child(string,integer)]
This field is a relation valued attribute. It consists of a list (which can be empty as well). Each of the elements of this list corresponds to a child of the employee. An element is a predicate child, with two fields, which combines two information, namely the name and the age of that child, the first is a string and the second is an integer value, respectively.

- **student** defines a student type relation, the attributes are as follows:

student(*id-no,courselist,dept-info,intelligence*)

*id-no* : string
This field is similar to the ones of person and employee.

*courselist* : [integer,...,integer]
A list of course numbers, the student has taken.

*deptinfo* : mdept(dept_name(string,string),integer)
is a two level nested attribute. It is a relation containing another relation inside. mdept is a predicate that combines the department name and the department number information. The first is another predicate and the second is an integer. This predicate is named as dept_name and combines again two information, namely the faculty name and the department name. Both of them are string valued.

*intelligence* : integer.
An attribute for the IQ level of the student.

### 4.4.2 Addition

The addition module has a structure given in Figure 4.2 . Addition is fired with the call of the predicate add. This predicate has no parameters. It first draws a menu with the aid of the toolbox utilities. Then, it asks to which relation an addition will be done. There are two choices

- employee
- student

Since both of them are persons without considering the selected option, first an addition to the person relation in the database is performed. For this purpose the doperson predicate is called with its parameter Qchoice instantiated to one the integers 1 or 2, meaning that the person is an employee or a student,

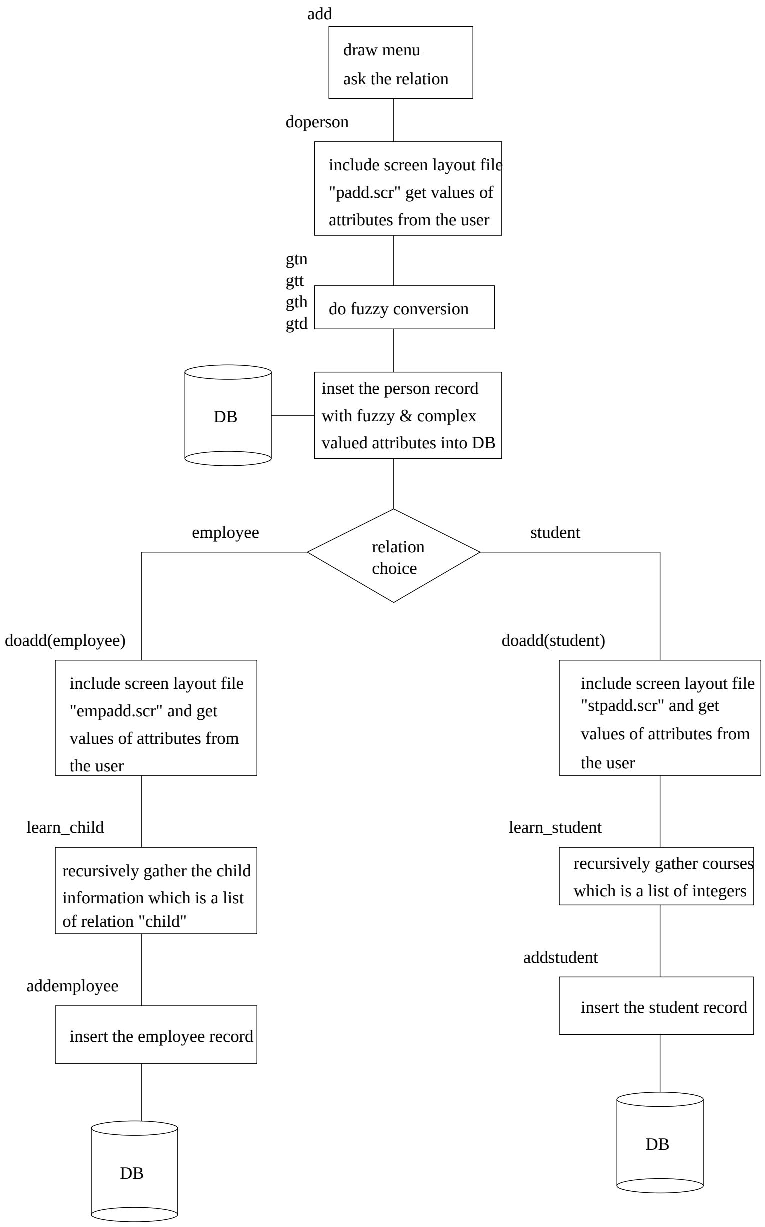


Figure 4.2 : Structure of the Addition Module

respectively. For each relation there is a predefined screen layout file existing in a file. This is necessary for the toolbox, which brings up a data entry screen. So, for the person relation addition person.scr file is consulted. The toolbox provides each field's user entry as an dynamically asserted predicate, value. value has two arguments, both of which are string valued. The first serves as a label for the field, and was defined through the toolbox, and the second is the user-entered value for that field (always returned as a string). Hence, if the desired value is a string for that field, the value of the second argument can be used straight away, otherwise a conversion must be performed (e.g. from string to integer). Depending on the nature of the field, another post process of the field value may be necessary. It might be the case that the field allows fuzzy value entries. Then the user-entered value (which is to hand as a string) has to be investigated to be of numerical content (like "1.85") or to be a keyword (like "tall"). This checking is done by gtn for number, gtt for telephone, gt for height and gtd for distance values.

After the necessary type conversions, and encapsulation processes, the person relation is constructed and asserted into the internal database. This database is also checkpointed to the file data.dbf . The last subgoal of the predicate add is a call to the doadd predicate. This call has two arguments, namely Qchoice and No. Being an employee or student record addition, Qchoice is instantiated as 1 or 2, respectively. No is the primary key value, a unique integer.

So, for example, an argument instantiation for doadd as doadd(1,4589) will start a employee addition with id-no of 4589. Similar to the person case described above, now a screen layout file empadd.scr, is loaded by the toolbox routines and the user is prompted to fill the fields of the screen out. Similar value checks, conversions, encapsulations are performed, as necessary. The entry of the child information is somewhat different. Since this attribute is a list by itself, a special data entry predicate is called. learn_child serves this purpose. For each individual child, this predicate calls a ASK_SINGLE_CHILD recursively. This predicate obtains the child name and the child age by consulting the user. Whenever "last" is typed in for the child name, the recursion is no more fired. The process predicate carry on the job, and performs the final adjustments of data, that means the packing of the gathered child name and child age under the predicate child_info. Then the assertion is performed.

Consider that the instantiation for doadd was doadd(2,4589). This would mean that a new student record is going to be added. The screen layout file "stadd.scr" would be loaded by the toolbox. As always, the results of the user-entry would dynamically stored under the value predicate as string values. The department code will be converted to numerical value. The course attribute of the student is an integer list. For this reason, a predicate learn_course which asks and get each course number is called recursively, until a "0" is entered by the user, indicating the end of course input. Then the assertion is performed.

### 4.4.3 Query & Delete

Query (or similarly deletion) described in Figure 4.3 is directly typed as an expression without considering to which attribute the query is done. When the query choice is chosen from the pull-down menu, the user is prompted to input his/her query expression. Here is an example for a possible entry by the user:

name> mehmet and (age <35 or age>40) and height=tall or distance<far

This sentence is parsed by activating predicates in the module parseval, but before that the expression has to be 'tokenized', that means disregarding the white spaces, the expression has to be chopped into its building blocks, namely the atomics. str_tokenlist does this job. str_tokenlist called with the above given input as a whole string instantiated to its first argument returns a list of tokens:

["name",">","mehmet","and","(","age","<","35","or","age",">","40",
")","and","height","=","tall","or","distance","<","far"]

The query predicate calls now the match predicate with the token list, which performs the parsing according to the precedence grammar for query defined below in the BNF notation:

$$
\begin{array}{rcl}
<expression> & ::= & <term_1> \;\; or \;\; <term_1> \;\; | \;\; <term_1> \\
<term_1> & ::= & <term_2> \;\; and \;\; <term_2> \;\; | \;\; <term_2> \\
<term_2> & ::= & (\, <expression> \,) \;\; | \;\; <comparison> \\
<comparison> & ::= & <attribute> \;\; <sign> \;\; <value> \\
<value> & ::= & <number> \;\; | \;\; <fuzzyvalue> \\
<attribute> & ::= & \mathsf{telephone} \mid \mathsf{height} \mid \mathsf{distance} \mid \mathsf{salary} \\
<sign> & ::= & < \;\; | \;\; = \;\; | \;\; > \\
<fuzzyvalue> & ::= & \mathsf{noinfo} \mid \mathsf{notexist} \mid \mathsf{notknown} \mid \mathsf{tall} \mid \\
 & & \mathsf{moderate} \mid \mathsf{short} \mid \mathsf{far} \mid \mathsf{near} \mid \mathsf{low} \mid \mathsf{high}
\end{array}
$$

The result which will be returned from the second argument of match will be a nested predicate expression like:

or(and($\square_1$,and(or($\square_2$,$\square_3$),$\square_4$)), $\square_5$)

where

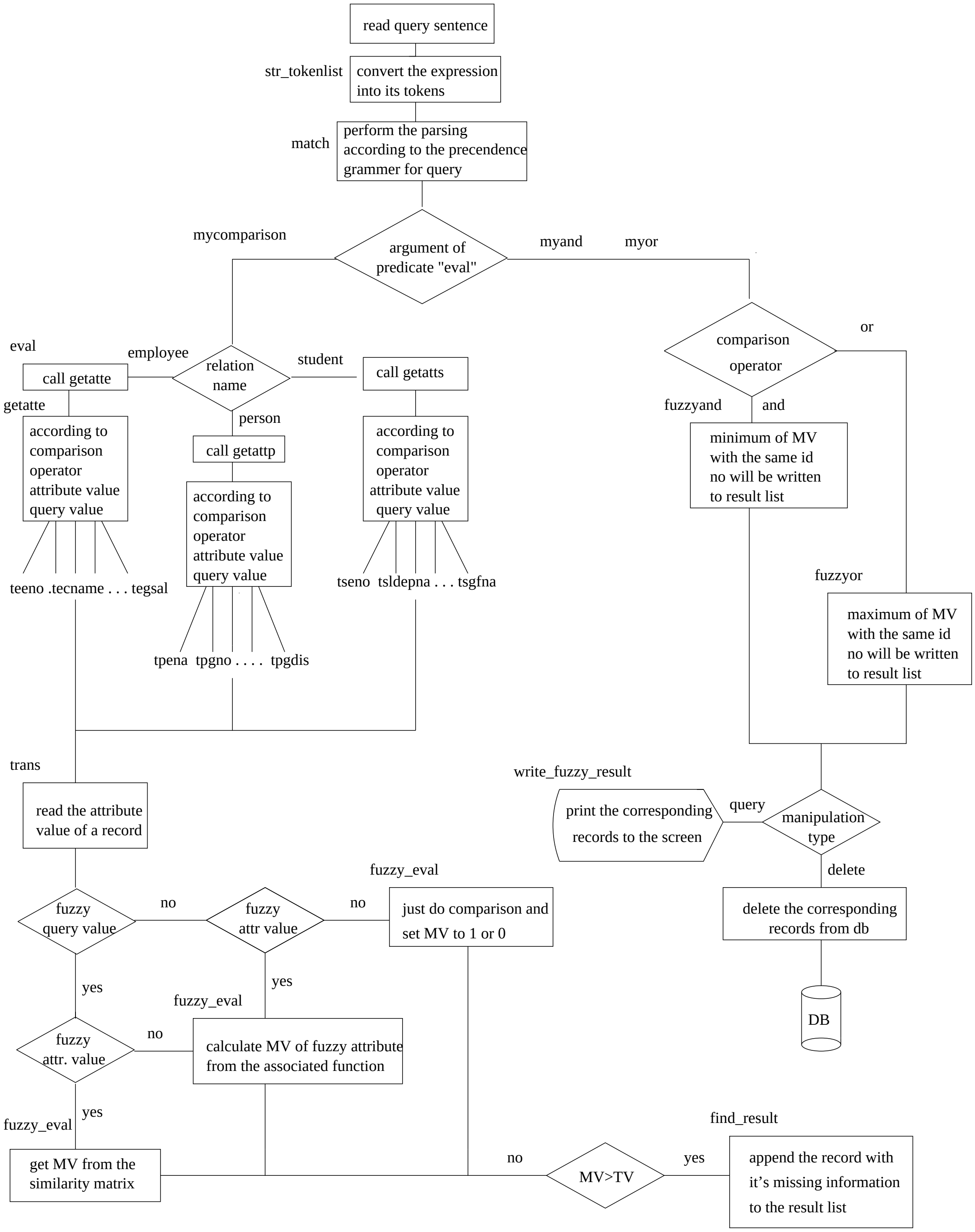


Figure 4.3 : Structure of the Query and Deletion

$\Box_1$ : mycomp(">","name","mehmet")

$\Box_2$ : mycomp("<","age","35")

$\Box_3$ : mycomp(">","age","40")

$\Box_4$ : mycomp("=","height","tall")

$\Box_5$ : mycomp("<","distance","far")

The next subgoal, query attempts, is to evaluate the outcome of the match. This is done by the predicate eval, deep inside the evaluation some recursive call will attempt to evaluate the value of a mycomp predicate. The answer is found by searching through the database, this will be explained in details, in the forthcoming lines, but for a moment let us not look at 'how' it is done but care about 'what' is done. The evaluation of a mycomp predicate boils down to a (hopefully non empty) fuzzy set. By fuzzy we mean a set where each element carries with it an MV, a measure of 'the right of being the member of the set'. If this eval of mycomp was fired (recursively) by the evaluation of an or, for instance, the father call of this recursive eval call will have now two fuzzy sets and the 'or' operation to hand. The next step is to take the 'union' of these two fuzzy sets (since it is an 'or' relation). So, the evaluation proceeds with fuzzy set operations from the deeper levels of the recursion to the outer levels. At the very final end, one expects to have a single fuzzy set to hand, which is the answer to the user's query.

So, now the question is how the solution (which is a fuzzy set) set for a mycomp goal is found. Looking at the above given query example, let us assume, somewhere deep in the recursion the system decides to find the solution set for

mycomp(">","height","tall")

There are three predicates one of which is called to find the solution.

getattp Finds the solution set for any given attribute and value searching through the person predicates in the database.

getatte Finds the solution set for any given attribute and value searching through the employee predicates in the database.

getatts Finds the solution set for any given attribute and value searching through the student predicates in the database.

All getatt$\Box$ predicates, four arguments have to be supplied: namely the field-id (like "height"), the specific value of the attribute that is subject to the query (like "tall"), and a comparison operator (like "="). And, an argument for the return of the result. Here are they all together:

getattp("=","height","tall",R)

There exist definitions for all possible argument combinations in all getatt□ predicates.

Extensive use is made of the findall built-in predicate of Turbo Prolog. This predicate, if called as findall(X,G,L), constructs a list L consisting of all the objects X such that the goal G is satisfied. G is a structure which is treated by findall as a Prolog goal. Also X will appear somewhere inside G. So, G can be instantiated to a Prolog goal of arbitrary complexity.

In our implementation it is not sufficient to have a collection list of a single attribute of the solution predicates in the database. We need the whole predicate. Furthermore, most of the queries are not as simple as 'does it exactly exist in the database or not?' questions. Most of them involve comparison and some of these are even of fuzzy type. This needs additional processing of the possible solution in the database. Hence, it is not satisfactory to have a single and simple findall goal with a second argument instantiated to a predicate like person. Therefore intermediate predicates are used as subjects to findall. In nature all these predicates take in their first $n-1$ arguments which are individual attribute entries (like Name,No,Tel,Age,Address, etc.), then return from its $n$ th argument the predicate that form the found solution (this predicate also includes the 'membership value' ,MV, of the solution). For example tpgno is the special predicate which is used to match persons with id-no's greater than the number instantiated to the first argument. The found solutions are returned from the second argument as a predicate (fperson) which resembles very much the person predicate (but has one more information, namely the MV value). So, a goal as

tpgna(1350,S)

will possibly instantiate to S a fperson predicate with arguments coming from the first match of a person predicate of the database that has an id-no greater than 1350. Upon backtracking S will return alternative fperson predicates with different arguments. Specific to tpgna, we can say that the MV value, returned as the last argument of the 'cooked up' fperson is always 1.0 since this query is none-fuzzy.

When we backtrack over the tpgna call, the second argument will get instantiated to all possible fperson solutions, so a findall call like

findall(S,tpgna(1350,S),L)

will cause L to be instantiated to a list of all the possible fperson solutions. This list is the fuzzy solution list.

The naming convention used in the source for these intermediate predicates is as follows

**1.letter** is always 't'

**2.letter** is one of

- 'p' : means the query is to be done on person predicate.
- 'e' : means the query is to be done on employee predicate.
- 's' : means the query is to be done on student predicate.

**3.letter** is one of

- 'g' : means the comparison is 'greater than'
- 'e' : means the comparison is 'equal to'
- 'l' : means the comparison is 'less than'

**rest letters** an abbreviation of the query attribute field name.

How are the fuzzy evaluations handled? That means what happens when a query contains a mycomp(">","salary","medium"). The first predicate that gets its hands on it is one of the above described 'intermediate' predicate. In particular, for the given example, this is tegsal. Since it is a characteristic piece of the code, let us have a closer look how this predicate is defined.

```
tegsal(Sal,femployee(Eno,Salary,Chl,MV)):-
                    employee(Eno,Salary,Chl),
                    fuzzyeval(Salary,Sal,">",salary,MV).
```

Here the first argument is the query attribute value. For the above given example, the first argument will be instantiated to "medium". The second argument is the output value, and will get its entries instantiated after the sub-goals on the right hand side are satisfied. The first subgoal is a query on the prolog database, upon backtracking it will get arguments instantiated by all the possible employee predicates. The second predicate serves as a kind of filter or acceptor for the particular instantiation that the employee subgoal made. So, fuzzyeval will, for this case, decide whether the value instantiated to Salary is greater than the fuzzy value of "medium" looking at the fuzzy relations defined for salary. It will compute a **membership value** for this instantiation and assign this to the MV argument. One important question remains: will all employee instantiations get a 'pass' through fuzzyeval, just having the MV calculated? Will it never fail, and hence reject the instantiation made by the employee call? The answer is no, the fuzzyeval call, which will be explained in details below, will fail for MV values which are below a certain **threshold** value. So, only those instantiations which have a **membership value** greater or equal than the threshold will have a chance to survive.

The `fuzzyeval` predicate is the heart of the evaluation for the attributes which may have fuzzy values. So, let us have a closer look at it.

First of all an attribute which can admit fuzzy values will not always receive a fuzzy value or will not always questioned with a fuzzy value. Actually the existence of all the four combinations is possible:

- The attribute has a fixed (we will call it '**deterministic**') value in database and the query is done with a **deterministic** value. (One expects that nothing in this case is fuzzy and the MV result is either 1.0 or 0.0, but actually this is **not** true, since the 'equality' is interpreted as 'being about' in this work)

- The attribute has a **deterministic** value in database and the query is done with a **fuzzy** value.

- The attribute has a **fuzzy** value in the database and the query is done with a **deterministic** value.

- The attribute has a **fuzzy** value in the database and the query is also done with a **fuzzy** value.

The `fuzzyeval` predicate makes extensive use of the pattern matching property of Prolog. The predicate acts different for the above stated cases and the rules for these different cases are implemented as different clauses for the `fuzzyeval` predicate. It is guaranteed that one of them will catch the particular case.

Database:**Deterministic,** Query:Deterministic

In this case the query (which is a string of a numerical value) will be converted to the numerical value and then if it is a 'greater than' or 'less than' operation these two numerical values (the one coming from the database:(arg.1) and the one coming form the query:(arg.2)) are mathematically compared. So either the comparison holds and the **MV** value is 1.0 and the `fuzzyeval` goal succeeds, it does not hold and `fuzzyeval` fails. If the comparison is 'equal', then this has a special meaning. An 'equal' query in this thesis is interpreted as the fuzzy operation of 'being about'. So, a special predicate `about` is called, which uses an evaluation function that tells how 'close' two values are, and depending on this, returns a value in the range [0.0,1.0]. This value then is compared to the threshold value and if it is above the threshold `fuzzyeval` goal succeeds with MV instantiated to this value, else the whole `fuzzyeval` goal fails.

Database & Query:**One is Deterministic, the other is fuzzy**

Every fuzzy attribute value has its own defining function. The naming convention for this predicate is *attributevalue*_eval

For example, the numerical evaluations looking at the function of "near" is done by the predicate near_eval. The idea behind is that this particular function shall evaluate the numerical value (the deterministic one) and return a measure of 'fitness', which will become out MV value. But before this, the returned value is checked against the threshold value. If the value calculated cannot pass this 'exam' the whole fuzzyeval goal fails. For the 'greater than' or 'less than' comparisons, the same scheme applies with a slight modification. In these cases the threshold value for acceptance is pushed up/lowered depending on the comparison being 'greater/less than'.

Database & Query:**Both are fuzzy**

This is a simple table look up. Here a predefined **similarity matrix** with its row/column labeled with the various fuzzy values of an attribute is used. The entries of the matrix are the 'membership values'. These values, as usual, are checked against the threshold value and if accepted returned as MV, else the fuzzyeval goal fails.

# CHAPTER V

# CONCLUSION

Current researches on database are strongly interested in extending available models to

- support complex knowledge,
- combine logic programming and databases,
- deal with uncertain (fuzzy) information.

While many attempts have been made to combine any two of the above, no attempt has been made to combine all of them. This work is based on the idea to join and hence utilize the power of all complex objects logic programming and uncertain information.

The designed and implemented knowledge based system proposed here, can be quoted as a fuzzy complex knowledge representation having the capability of dealing with various types of uncertain information. Logic programming language is used in the implementation of the system. An important feature of this logic-based system is that the computational power of logic programming is utilized for representation and manipulation of both the complex structure of extended $NF^2$ data models and uncertain information.

In the implementation the Turbo-Prolog language was chosen. This dialect of Prolog is strongly typed and uses a compilation approach. Both of these features fitted the needs of this work quite well, since the compilation technique performs a global analysis of the rule set prior to the formulation of the target code in the form of a query against the basic relations of database.

In the implementation of complex objects, it can be concluded that logic programming is well suited, especially for the representation of nested relations. Furthermore, other complex types like 'range' and 'set' could easily be modeled and implemented in Prolog without any need for an extension of the language.

Since the main concern of the work was the modeling of combination of the three areas mentioned, no special effort was spent on efficiency issues. The implementation runs on a small database and a limited number of relations and rules without exhibiting any time/memory deficiency. However, when the size of the relations in the database increases, efficiency has to be considered seriously. Also when the size of the main memory is not sufficient to hold all the rules and relations then the usage of external storage has to be considered. This problem can be solved by enhancing logic programming language for better data managment or coupling database and logic systems[38].

Future work:

- The prototype can be developed further to inherit the complete database operations, like 'avg', 'merge','join', etc.
- A database manipulation language can be implemented to allow a dynamic structure.
- The merge/unmerge features of extended $NF^2$ [1] can be added.
- Fuzzy inference and deduction mechanisms of this system can be developed.
- An SQL like querying language can be implemented as user interface.
- Natural language quantifiers like 'very', 'almost', 'nearly', etc. can be included.

# REFERENCES


[1] S. Abiteboul and N. Bidoit. Non-first-normal Relations: An Algebra allowing Data Restructuring. *Journal of Computer System Sciences*, 33(3):361–393, December 1986.

[2] S. Abiteboul, P.C. Fischer, and H.J. Schek, editors. *Nested Relations and Complex Objects in Databases*, volume 361. Springer-Verlag, Berlin, 1989.

[3] S. Abiteboul and S. Grumbach. COL: A Logic-Based Language for Complex Objects.

[4] M. Adiba and G.T. Nguyen. Handling Constraints and Metadata on Generalized Data Management Systems. In *International Workshop on Expert Database Systems*, Kiowah Island, South Carolina, October 1984.

[5] D.S. Batory and W. Kim. Modelling Concepts for VLSI CAD Objects. In *ACM Transactions on Database Systems*, volume 10, September 1985.

[6] O. Bobrorowicz, C. Choulet, A. Haurat, F. Sandoz, and M. Tebaa. A Method to Build Membership Functions. In B. Bouchon-Meunier, R.R. Yager, and L.A. Zadeh, editors, *3rd International Conference on Information Processing and Management of Uncertainity in Knowledge-Based Systems, IPMU'90*, Paris, 1990.

[7] J. Bocca, H. Decker, J. M. Nicolas, L. Vielle, and M. Wallace. Some Steps Toward a DBMS-Based KBMS. In *Proc. IFIP World Conference*, Bublin, 1986.

[8] K. Bowen and R.A. Kowalski. Amalgamating Language and Metalanguage in Logic Programming. In K.L. Clark and S.A. Tarnlund, editors, *Logic Programming*, 1982.

[9] B.P. Buckes and F.E. Petry. Uncertainity Models in Information and Database Systems. *Journal of Information Science*, (11):77–87, 1985.

[10] S. Ceri, G. Gottlab, and L. Tanca. *Logic Programming and Databases.* Springer-Verlag, Surveys in Computer Science, 1990.

[11] S. Ceri, G. Gottlob, and G. Wiederhold. Interfacing Relational Databases and Prolog Efficiently. In *Proc. First Int. Conference on Expert Database Systems*, Charleston, 1986.

[12] S. Ceri, G. Gottlob, and G. Wiederhold. Efficient Database Access in Prolog. *IEEE- Transactions on Software Engineering*, February 1989.

[13] C.L. Chang and A. Walker. PROSQL: A Prolog Programming Interface with SQL/DS. In L. Kerschberg, editor, *Proc. from the First Int. Workshop on Expert Database Systems*, Kiowah Island,SC, October 1984. Benjamin-Cummings.

[14] Quiming Chen and Yahiko Kambayashi. Nested Relation Based Database Knowledge Representation. In *Proc. of the 1991 ACM SIGMOD, International Conference on Management of Data*, volume 20, Denver, Colorado, June 1991.

[15] F. Cuppens and R. Demolombe. A PROLOG-Relational DBMS interface using Delayed Evaluation. In *Workshop on Integration of Logic Programming and Databases*, Venice, December 1986.

[16] P.W. Dart and J. Zobel. Conceptual Schemas Applied to Deductive Databases. *Information Systems*, 13(3):273–287, 1988.

[17] B.C. Desai, P. Goyal, and F. Sadri. Non-first-form Universal Relations: An Application to Information Retrieval Systems. *Information Systems*, 12(1):49–55, 1987.

[18] P. Fischer and D. Van Gucht. Determining when a Structure is a Nested Relation. In *Proceedings of the Eleventh International Conference on Very Large Databases*, 1985.

[19] H. Gallaire. Impacts of Logic and Databases. In *Proc. 7th Internal Conference on Very Large Databases*, Cannes,France, 1981.

[20] H. Gallaire, J. Minker, and J.M. Nicolas. Logic and Databases: A Deductive Approach. In *ACM*, 1984.

[21] G. Gardarin, J.P. Cheincy, G. Kiernan, D. Pastre, and H. Stara. Managing Complex in an Extensible Relational DBMS. In *Proceedings of the Fifteenth International Conference on Very Large Databases*, Amsterdam, 1989.

[22] S. Greco, N. Leone, and P. Rullo. COMPLEX: An Object-Oriented Logic Programming System. *IEEE Transactions on Knowledge and Data Engineering*, 4(4), August 1992.

[23] M.M. Gupta and E. Sanchez. *Fuzzy Information and Decision Processors.* North-Holland, 1982.

[24] M. Gyssens, J. Paredaens, and D. Van Gucht. A Uniform Approach towards Handling Atomic and Structured Information in the Nested Relational Database Model. *Journal of ACM*, 36(4):790–825, October 1989.

[25] R. Haskin and R. Lovie. On Extending the Functions of a Relational Database System. In *ACM SIGMOD International Conference*, Orlando, Florida, June 1982.

[26] C.W Holsapple and A.B. Whinston. Decision Support Systems: Theory and Applicaiton. Springer-Verlag, 1987.

[27] Y.E. Ioannidis, J. Chen, M.A. Friedman, and M.M. Tsangaris. BERMUDA- An Architectural Perspective on Interfacing Prolog to a Database Machine. Technical Report 723, University of Winsconsin, CS Dept., October 1987.

[28] Y. Kambayashi, K. Tanaka, and K. Takeda. Synthesis of Unnormalized Relations Incorporating More Meaning. *Information Sciences Journal*, 29:201–247, 1983.

[29] R. Kowalski. Logic as a Database Language. Technical report, Imperial College, London, July 1981.

[30] G.M. Kuper and M.Y. Vardi. The Logical Data Model. *ACM Transactions on Database Systems*, 18(3):379–413, September 1993.

[31] K.S. Leung and W. Lan. Fuzzy Concepts in Expert Systems. *IEEE Transactions*, pages 43–53, 1988.

[32] M. Levene and G. Loizou. Semantics for Null Extended Relations. In *ACM Transactions on Database Systems*, volume 18, pages 414–459, 1993.

[33] D. Li and D. Liu. *A Fuzzy Prolog Database System.* Research Studies Press Ltd., 1990.

[34] W. Lipski. On Semantic Issues Connected With Incomplete Information Systems. volume 4, pages 262–296, September 1979.

[35] A. Makinouchi. A Consideration on Normal Form at Not-necessarily-normalized Relation in the Relational Data Model. In *Proceedings of 3rd International Conference on Very Large Databases*, pages 447–453, Tokyo, 1977.

[36] N. Malvache and D. Willaeys. The Use Of Fuzzy Sets for the Treatment of Fuzzy Information by Computer. *J.FSS*, (5):323–327, 1981.

[37] T.P. Martin and J.F. Baldwin. An Abstract Mechanism for Handling Uncertainity. In B. Bouchon-Meunier, R.R. Yager, and L.A. Zadeh, editors, *3rd International Conference on Information Processing and Management of Uncertainity in Knowledge-Based Systems, IPMU'90*, Paris, 1990.

[38] B. Napheys and D. Herkimer. A Look at Loosely-Coupled Prolog Database Systems. In *Proc. from the Second Int. Conf. on Expert Database Systems*.

[39] S. Nurcan, L. Lei, and J. Kouloumdjian. Integrating DB Technology and Logic Programming Paradigm. In *ACM*, 1990.

[40] G. Özsoyoğlu, Z.M. Özsoyoğlu, and I. Mata. A Language and a Physical Organization Technique for Summary Tables. In *ACM SIGMOD International Conference*, Austin, Texas, May 1985.

[41] Z.M. Özsoyoğlu and Yuan L.Y. A New Normal form for Nested Relations. In *ACM Transactions Database Systems*, volume 12, pages 111–136, 1987.

[42] D.S. Parker, M. Carey, F. Golshani, M. Jarke.and E. Sciore, and A. Walker. Logic Programming and Database. In *Proc. from the First Int. Workshop on Expert Database Systems*, Kiowah Island,SC, 1986. Benjamin-Cummings.

[43] Quintus Computer Systems Inc., California. *Quintus Prolog Database Interface*, June 29 1987. Version 1.

[44] Moore R.C. and J.C. Hobbs, editors. *Formal Theories at the Commonsense World*. Harwood, NJ : Ablex, 1984.

[45] M.A. Roth and H.F. Korth. The Design of ¬1NF Relational Databases into Nested Normal Form. In *ACM*, pages 143–160, 1987.

[46] M.A. Roth, H.F. Korth, and A. Silberschatz. Extended Algebra and Calculus for Nested Relational Databases. 13(4):389–417, 1988.

[47] M.A. Roth, H.F. Korth, and A. Silberschatz. Null Values in Nested Relational Databases. 13(4):615–642, 1989.

[48] H.J. Schek and M.H. Scholl. The Relational Model with Relation-valued Attributes. pages 137–147, 1986.

[49] E. Sciore and D.S. Warren. Towards an Integrated Database-Prolog System. In *Proc. from the Second International Conference on Expert Database Systems.*

[50] J. Ullman. *Principles of Databases and Knowledge-base Systems.* Computer Science press, 1988.

[51] P. Valduriez, S. Khoshafian, and G. Copeland. Implementation Techniques of Complex Objects. In *Proc. of the Twelfth Int. Conference on Very Large Databases*, Kyoto, August 1986.

[52] A. Yazıcı. A Database Model for Handling Complex and Uncertain Information. In *Proc. of the Seventh Int. Symp. on Computer and Informatin Sciences (ISCIS-VII)*, pages 601–604, Antalya, Türkiye, October 1992.

[53] A. Yazıcı, B.P. Buckles, and F. E. Petry. A Semantic Data Model Approach to Knowledge-Intensive Applications. In *Proc. of Florida AI Research Symposium*, pages 115–119, Florida, USA, April 1992.

[54] A. Yazıcı, R. George, B.P. Buckles, and F.E. Petry. A Survey of Conceptual and Logical Data Models for Uncertainity Management. In L. A. Zadeh and J. Kacprzyk, editors, *Fuzzy Logic for Management of Uncertainity*, pages 607–644. John Wiley and Sons, Inc., New York, USA, 1992.

[55] L.A. Zadeh. Fuzzy Sets. *Information and Control*, (8):338–353, 1965.

[56] L.A. Zadeh. Similarity Relations and Fuzzy Orderings. *Information Science*, (3):177–206, 1970.

[57] L.A. Zadeh. The Treatment of Uncertainity in Knowledge-based Systems Using Fuzzy Sets and Possibility Theory. *International Journal of Intelligent Systems*, 3:141–165, 1988.

[58] L.A. Zadeh. Knowledge Representation in Fuzzy Logic. *IEEE Transactions*, 1(1), March 1989.

[59] C. Zaniolo. The Representation and Deductive Retrieval of Complex Objects. In *International Conference on VLDB*, Stockholm, August 1985.

[60] H.J. Zimmermann. *Fuzzy Set Theory-and Its Applications.* Kluewer-Nijhof Publishing, 1985.

# APPENDIX A

# PROLOG AS A LANGUAGE FOR LOGIC PROGRAMMING

Prolog stands for PROgramming in LOGic. It was implemented as a simplification of general theorem proving techniques in an efficient and programmable form. Use of Prolog and databases (especially relational databases) have become widespread in commercial world especially in Europe and academic and scientific environments throughout the seventies and early eighties [10]. Throughout these years, some important theoretical studies have been conducted which ended up with the preference of Prolog as a programming paradigm to Japanese Fifth Generation Project. This project introduces the development of the computers of a new generation which have powerful deduction capabilities. The project uses relational data model for storing large collections of data. Therefore researches about interfacing Prolog and Relational Databases have been accelerated [10].

Here is a brief description of some of the features of the language Prolog. Prolog language structure based on the notions of atom, variable, functor, predicate, term, literal, and a clause, defined below.

An atom is a finite string of alphanumeric characters starting with a lowercase letter. Examples of atoms are: `book`, `mary`, `'John'`, `y13bg`. A constant is either an atom or a number such as `0`, `1234`, `5`.

A variable is a finite string of alphanumeric characters starting with an uppercase letter. Examples are: `Book`, `X`, `Var`. The special variable "_", called anonymous variable, is thought of as standing for any unnamed variable.

A functor is characterized by its name (which is an atom) and its arity, i.e. the number of arguments. A functor can be an operator, like "+", or a predicate name, like ">" or "ancestor". Each predicate defines a mapping from the domain $\mathcal{D}$ for which the predicate is defined to the set of values $true$, $false$. For instance, we can apply "ancestor" to the pair (bill,joe), and its value will be $true$ if bill is an ancestor of joe, or false otherwise.

Atoms and variables are terms. Structured terms are built recursively by applying functors to the other terms for example ancestor(bill,joe). Thus in Prolog predicates are also viewed as terms. A literal is either a structured term whose outermost symbol is a predicate. Examples of literals are: likes(george,mary), or member(X,L).

A prolog clause is a collection of literals. A Prolog program is a collection of clauses. Variables are local to clauses, i.e., scope of each variable is limited to the clause in which it appears. There are three kinds of clauses:

- **Rules:** A rule is a clause consisting of a head which is one positive literal, and a body, formed by several literals separated by commas. The head and the body are separated by the symbol ":-". A rule expresses a conditional assertion. For example:

  likes(george,Y) :- woman(Y), likes(Y,books).

  says that George likes any woman, provided that she likes books. Notice that variables in the rule head are to be considered as universally quantified, while the variables in the body are to be considered as existentially quantified. The comma between the literals in the rule body represents conjunction. A rule body can also be in disjunctive form. For example,

  likes(mary, Y) :- nice(Y); loves(Y,cats).

  says that Mary likes a person either if this person is nice or if he/she loves cats.

- **Facts:** A fact is a clause consisting of a unique literal. It is an assertion which is an axiom, i.e. it is true independently of any condition. For instance, likes(george,mary). says the fact that George likes Mary is true in the program.

- **Goals, or headless clauses:** these represent the queries. When a goal is presented to a Prolog engine, it tries to prove that the goal is true by using the facts and rules that are presented in the program. A Prolog goal is always preceded by the symbol "?-". For example

```
?- likes(george, lucy)
```

will cause the Prolog Engine to try to prove that George likes Lucy by using facts and rules contained in the program.

A Prolog Engine is an inferential machine which operates on a given Prolog program, aiming to prove the truth of a given goal. A goal is proven if it matches the head of some clause, and each of the literals (if any) of the body of that clause, considered as a goal, is also proven.

In this way, goals are arranged in a proof tree. Computation is performed by the Prolog interpreter by traversing this tree in a top-down, depth-first fashion. In this context, matching of two literals has to be understood informally as the fact that an instantiation of variables of both literals that makes values equal exists.

The most important features of Prolog are recursion and backtracking:

- Prolog allows the definition of recursive predicates. A predicate is recursive if its definition involves the predicate itself. An example is the following program to detect list membership:

  ```
  member(X,[X|_]).
  member(X,[_|Y]) :- member(X,Y).
  ```

  The first argument of the predicate member is a variable, the second argument is a list, denoted by enclosing its elements in square brackets "[]". The first fact says that X is member of a list if it is the head of the list. The second clause says that X is member of a list if it is a member of its tail. Notice the use of the anonymous variable "_", and the list concatenation operator "|".

- If, during the exploration of the proof tree, Prolog encounters a goal that is not satisfiable, it retraces its own course by going backwards along the last tree branch, and resumes traversal by trying to re-satisfy the goal to the left of the one that just failed. This feature is called backtracking.

The aim of backtracking is to enable the exploration of the proof tree with all possible variable instantiations. However, sometimes this makes the computation rather slow. In order to compensate for this and other limitations of Prolog, other constructs have been introduced that introduce procedurality. These are extra logical primitives, which express control information about how the search must be carried out.

The most important of these primitives is the cut "!", used to control backtracking. Its effect is to make the variables occurring to the left of the cut in the current rule irrecoverably instantiated: variable instantiations made by the Prolog engine before reaching the cut cannot be changed any more.

Other extra logical primitives are, for instance, the predicates retract and assert, which, respectively, withdraw and put clauses to the program.

# APPENDIX B

# DICTIONARY

## B.1 Module Dictionary

The program consists of six modules:

1. global
2. path
3. add
4. query_delete
5. parseval
6. toolbox

**Global :** The domains to communicate across module boundries using the predicates defined in the global predicates and global database sections are declared in this file. In every relevant module it is included with an include directive.

**Path :** All modules used should be linked together to create a single executable program.Path is the project definition file containing a list of names of the modules in the project.

It's content is like global+add+query_delete+parseval+toolbox.

**Add :** This module responsible for the addition of person and employee or student record with fuzzy and complex valued attributes.

**Query_delete :** Entered the query/delete sentence with fuzzy and complex values, it finds the satisfying records and lists or delete them.

**Parseval :** Seperates query or delete sentence into its tokens and performs evaluation of the query.

**Toolbox :** is a TURBO PROLOG utility, used to design screen layouts. Contains predicates for menus, windows.

**Datam :** This is the last file containing person, employee and student relations with complex and fuzzy valued attributes.

## B.2 Predicate Dictionary

mainm

| | |
|---|---|
| input arguments | : None |
| output arguments | : None |
| function | : Draws a menu asking the which data manipulation will be done and gets Choice accordingly. |
| calls | : do(Choice) |

do

| | |
|---|---|
| input arguments | : Choice : integer |
| output arguments | : None |
| function | : Calls one of delete, add or query procedures according to the value of Choice. |
| calls | : Choice of<br>1 : add<br>2 : delete<br>3 : query |

add

input arguments : None

output arguments: None

function : Asks the relation name addition will be done.

calls : doperson(Qchoice)

Qchoice of

1 : employee

2 : student

doperson

input arguments : Qchoice : integer

output arguments: None

function : Consults screen layout file padd.scr, gets the values of the attributes being atomic,fuzzy and complex, inserts the record into the database.

calls : gtn,gtt,gth,gtd

doadd(Qchoice,No)

Qchoice of

1 : employee

2 : student

doadd

input arguments : Qchoice : integer

No : real (Since the id number is the same as employee no and student no, inorder not to ask it twice, it is passed to the doadd procedure as a parameter.)

output arguments: None

function : Consults screen layout file stadd.scr or empadd.scr, gets the associated values of the attributes being atomic, fuzzy and complex, inserts the record.

calls : learn_info or learn_course

query

input arguments : None

output arguments: None

function : Entered query expression is parsed, and the outcome is evaluated, answer to the query expression being atomic, fuzzy and complex, is inserted to a result list.

calls : str_tokenlist
match
eval
write_reslist

str_tokenlist

input arguments : String : the whole query expression.

output arguments: L : List of tokens (string list)

function : A string that is read in as the query expression is tokenized. The result is a list of token strings.

calls : str_tokenlist_slave
trim_tokenlist

eval

input arguments : 1.arg : One of the three predicates
1) myand
2) myor
3) mycomp

output arguments: R : A fuzzy set

function : The fuzzy evaluation of the prefix representation of the query given in the input argument. The output is a fuzzy set.

calls : fuzzyand
fuzzyor
getattp
getatte
getatts
findresult

fuzzyand

input arguments : 1.arg and 2.arg : Fuzzy sets

output arguments: 3.arg : Fuzzy set

function : Performs the intersection of two fuzzy sets given in its first and second argument. Result is fuzzy set returned form the third argument.

calls : fuzzy_and_member

fuzzyor

input arguments : 1.arg and 2.arg : Fuzzy sets

output arguments: 3.arg : Fuzzy set

function : Performs the union of two fuzzy sets given in its first and second argument. Result is fuzzy set returned form the third argument.

calls : fuzzy_and_member

findresult

input arguments : list of fperson(...), fstudent(...) or femployee(...)

output arguments: list of fresult(...)

function : Given the fperson, femployee or fstudent, join is done on id-number which is common in three relations. Output is put into a new list consisting of fresult(...)s.

calls : None

# APPENDIX C

# PROLOG SOURCE

## global

```
global domains

/* for toolbox */
  ROW, COL, LEN, ATTR   = INTEGER
  STRINGLIST = STRING*
  INTEGERLIST = INTEGER*
  KEY    = cr; esc; break; tab; btab; del; bdel; ctrlbdel; ins;
      end ; home ; fkey(INTEGER) ; up ; down ; left ; right ;
      ctrlleft; ctrlright; ctrlend; ctrlhome; pgup; pgdn;
      ctrlpgup; ctrlpgdn; char(CHAR) ; otherspec

  FNAME=SYMBOL
  FNAMELIST= FNAME*
  TYPE = int(); str(); real()


/* for our program */


   Gd,Gm,Nd,Nm=integer
   lower,upper=integer
   mv=real
   name,address,dname,fac,ch_name= string
   e_no,s_no,dept_no,c_no,tp=integer

   somefuzzy = k(real);uk(string)  /* obsolate ??? */
   height  =   k(real);uk(string)
   salary  =   k(real);uk(string)
   distan  =   k(real);uk(string)
   Presult =   k(real);uk(string)
   Sresult =   k(real);uk(string)
   Dresult =   k(real);uk(string)
   tel     =   k(real);uk(string)
   Tresult =   k(real);uk(string)

   intel = string
   no= e(integer);s(integer)
   /* person's id# should be same as one of
   the employee or student number */

   Ch = char
```

```
Linepos,Filepos = integer
Line = string

courses=integer*

ch_age=ch_age(lower,upper)
age=age(lower,upper)
/* age is a fuzzy value, with upper and lower
limits */


/* in order to represent fuzzy telephone information
either known or unknown is ued, unknown may be notexist
or no information to explain the reason of not knowing */


list=symbol*
listch =char*

child=child(ch_name,ch_age)
childl=child*
/* child relation is a subrelation of person,
 consisting of child name and child age */


dept_name=dept_name(fac,dname)
/* department name consists of faculty and
department name */

mdept=mdept(dept_name,dept_no)
/* major department is a subrelation of student,
with department name and code. */

per   = person(name,no,tel,age,address,height,distan,tp)
fper  = fperson(name,no,tel,age,address,height,distan,tp,mv)
femp  = femployee(e_no,salary,childl,mv)

emp   = employee(e_no,salary,childl)
st    = student(s_no,courses,mdept,intel)
fst   = fstudent(s_no,courses,mdept,intel,mv)

fgen = fperson(name,no,tel,age,address,height,distan,tp,mv);
       femployee(e_no,salary,childl,mv);
       fstudent(s_no,courses,mdept,intel,mv)


stlist = st*
fstlist = fst*
elist = emp*
felist = femp*
plist = per*
fplist = fper*
rlist = r*

Resultlist = rlist
Perlist,Outper = plist
Emplist = elist

genlist = fgen*
```

```
  res = veya(res,res); ve(res,res);
        kiyas(string,string,string); o(string)
  lstring = string*
  r = fresult1(e_no,salary,childl,
               name,tel,age,address,height,distan,tp,mv);
     fresult2(s_no,courses,mdept,intel,
              name,tel,age,address,height,distan,tp,mv)

global database

  insmode /* Global insertmode */
  actfield(FNAME) /* Actual field */
  screen(SYMBOL,DBASEDOM) /* Saving different screens */
  value(FNAME,STRING)          /* value of a field */
  field(FNAME,TYPE,ROW,COL,LEN) /* Screen definition */
  txtfield(ROW,COL,LEN,STRING)
  windowsize(ROW,COL)
  notopline

  /* DATABASE PREDICATES USED BY VSCRHND */
  windowstart(ROW,COL)
  mycursord(ROW,COL)

  /* Database declarations used in lineinp */
  lineinpstate(STRING,COL)
  lineinpflag



global predicates

/*  for toolbox */

  nondeterministic menu(ROW,COL,ATTR,ATTR,
           STRINGLIST,STRING,INTEGER,INTEGER) - (i,i,i,i,i,i,i,o)

  makestatus(ATTR,STRING) - (i,i)
  changestatus(STRING) - (i)
  scrhnd(SYMBOL,KEY)   - (i,o)
  readkey(KEY)         - (o)

  field_action(FNAME)  - (i)
  field_value(FNAME,STRING) - (i,o)
  noinput(FNAME)       - (i)


/* for our program */

  member(child,childl)            - (i,i) (o,i)
  member(integer,courses)         - (i,i) (o,i)
  member(per,plist)               - (i,i)
  member(emp,elist)               - (i,i)
  member(st,stlist)               - (i,i)
  member(fper,fplist)             - (i,i)
  member(femp,felist)             - (i,i)
  member(fst,fstlist)             - (i,i)
  member(r,rlist)                 - (i,i)
  member(string,lstring)          - (i,i)
```

```
findresult(genlist,rlist) - (i,o)
getatte(string,string,string,genlist) - (i,i,i,o)
getattp(string,string,string,genlist) - (i,i,i,o)
getatts(string,string,string,genlist) - (i,i,i,o)
nondeterm match(symbol,lstring,res) - (i,i,o)
str_tokenlist(string,lstring) - (i,o)
eval(res,rlist) - (i,o)

main  /* is defined in gok3 .. the entry point */
```

**add + query_del**

```
project "tez"

include "global.pro"


 database - owndatabase

  person(name,no,tel,age,address,height,distan,tp)
  employee(e_no,salary,childl)
  /* employee is a nested relation with atomic
   attributes e_no and salary, and anon atomic
   attribute which is alist of child relation */

  student(s_no,courses,mdept,intel)
  /* student is a two level nested relation
  consisting of an atomic attribute s_no, list
  of taken courses and another nested relation
  major department */

  isa(string,string)
  /* this predicate is used to simulate IFO's
  isa relation */

  accept_level(real)/* This his a treshold value (which may be updated
                       at run time and serves the purpose of acceptance/
                       refutation of a calculated MV result. */


predicates

  query
  delete
  delete_slave(rlist)
  dodelete(rlist)

  get(symbol)

  mainm

  isas
  isae

  do(integer)

  fuzzyeval(somefuzzy,string,string,symbol,mv)
  fuzzyeval(height,string,string,symbol,mv)
  fuzzyeval(salary,string,string,symbol,mv)
  fuzzyeval(distan,string,string,symbol,mv)
  fuzzyeval(somefuzzy,string,string,mv)
  fuzzyeval(height,string,string,mv)
  fuzzyeval(salary,string,string,mv)
  fuzzyeval(distan,string,string,mv)

  /* arguments:
    1.     somefuzzy: either k(real) or uk(some_fuzzy_keyword_string)
    2.     string   : comes from query: it is either
                          a) some_fuzzy_keyword_string
                          b) some real value which is exact
```

```
    3.     string   : one of "<", ">", "="
    4.     symbol   : to clasify which type of fuzzy relation (only
                      used in (=) of two deterministic values)
                          a) distance
                          b) height
                          c) telefon
    5.     mv       : The famous "Membership Value" calculated according
                      the 1. and 2. argument

 */

  about(symbol,real,real,real)
/*   This predicate calculates a MV (which is returned from the 3rd slot)
     for the first two slots. The idea is to use the function:
                  -c(x-y)
                 e
     the coefficient (c) controls the broadness of the hill,
     it will live in a predicate inserted in the database as like:
            about_c(1.2)
     the symbol is a keyword (like 'height') which is used as an index
     to the correct (c) value.;
*/


  about_c(mv)
  accept(mv)

  in_omega_range(string,real,integer,integer)
  in_omega_range(string,real,integer)

  membership_fn(real,string,mv)
  membership_fn1(real,string,mv)
  membership_fn2(real,string,mv)
  membership_fn3(real,string,mv)
  integro(real,string,string,mv)
  integro1(real,string,string,mv)
  integro2(real,string,string,mv)
  integro3(real,string,string,mv)
  ff1(real,string,real)
  ff2(real,string,real)
  f1(real,string,real)
  f2(real,string,real)
  f3(real,string,real)
  g(real,string,integer,real)
  alpha(string,real)
  alpha(string,integer,real)
  beta(string,real)
  c_constant(string,real)
  omega(string,integer,real)
  ksi(string,real)
  eta(string,real)
  rho(string,real)
  delta(string,real)
  lambda(string,real)
  nu(string,real)
  sigma(string,real)



  transp(name,no,tel,age,address,height,distan,tp,fgen)
```

```
transp2(name,no,tel,string,address,height,distan,tp,fgen)

transe(e_no,string,fgen)
transe2(e_no,string,string,fgen)
transe3(e_no,string,fgen)

transs2(s_no,integer,string,string,integer,string,fgen)


tsldna(string,fgen)
tsldno(string,fgen)
tslfac(string,fgen)
tsgdna(string,fgen)
tsgdno(string,fgen)
tsgfac(string,fgen)


tpgna(string,fgen)
tpgno(string,fgen)
tpgage(string,fgen)
tpgadd(string,fgen)
tpghe(string,fgen)
tpgdis(string,fgen)

tpehe(string,fgen)
tpedis(string,fgen)
teesal(string,fgen)

tegsal(string,fgen)
tegcna(string,fgen)
tegcage(string,fgen)


tplna(string,fgen)
tplno(string,fgen)
tplage(string,fgen)
tpladd(string,fgen)
tplhe(string,fgen)
tpldis(string,fgen)

telsal(string,fgen)
telcna(string,fgen)
telcage(string,fgen)


add
/* ask the relation to which the record will be added */

doadd(integer,integer)
/* menu for adding a record to employee or student */

doperson(integer)



 learn_info(real,childl)
 /* recursively getting child's information for addition */
 learn_child(childl)
 ask_single_child(child)
```

```
   /* this predicate is used for recursion of deleting a student, testing
   is the last child is entered or not */
   process(symbol,child)

  /*recursively getting taken courses for addition and query*/
   learn_course(courses)
   ask_single_course(integer)

  /* this predicate is used for recursion, testing is the last course
  is entered or not */
   processc(integer)

  gt(string,height)
  gtd(string,distan)
  gtt(string,tel)
  gtn(integer,integer,no)
  gts(string,salary)



/* Here start the fancy output predicates */

write_fuzzy(height)
write_fuzzy(distan)
write_fuzzy(salary)
write_tel(tel)
draw_line
wait

write_perlist(fplist)
write_perlist_slave(fplist,integer)

write_reslist(rlist)
write_reslist_slave(rlist,integer)

confirm(string)


 clauses

/* That is the entry point  and is defined as a global predicate  */

main :-
    consult("datam.dba",owndatabase),
    mainm.




  member(X,[X|_]) :- !.
  member(X,[_|T]) :- member(X,T).


  /* inorder to query nested relations, memberhip test
  is required */


/**************************************************************************/
```

```
/*                    F U Z Z Y   E V A L                          */
/******************************************************************/


about_c(0.162).    /* the broadness control of the below function */

about(Type,X,Y,Z) :- about_c(C), Z = exp(-C*(X-Y)).

accept(MV) :- accept_level(AL), MV>=AL, MV<=1.0,!.

/*********************/
/*   G E N E R A L   */
/*********************/

/* D-base:Deterministic,   Query:Deterministic <decision is still fuzzy> */
fuzzyeval(k(R),Query,"=",Type,MV) :-  str_real(Query,Q),
                                      about(Type,R,Q,MV),!,accept(MV).
fuzzyeval(k(R),Query,"<",1.0) :- str_real(Query,Q),!, R < Q.
fuzzyeval(k(R),Query,">",1.0) :- str_real(Query,Q),!, R > Q.

/* D-base:Deterministic,   Query:Fuzzy */
fuzzyeval(k(R),Fuzzy,"=",MV)  :- membership_fn(R,Fuzzy,MV), !, accept(MV).
fuzzyeval(k(R),Fuzzy,Sign,MV) :- integro(R,Sign,Fuzzy,MV), !, accept(MV).

/* D-base:Fuzzy,   Query:Deterministic */
fuzzyeval(uk(Fuzzy),S,"=",MV) :- str_real(S,R), !,
                                 membership_fn(R,Fuzzy,MV),
                                 accept(MV).
fuzzyeval(uk(Fuzzy),S,Sign,MV):- str_real(S,R), !,
                                 integro(R,Sign,Fuzzy,MV),
              writef("GENERAL#1 ==> MV value=%5.2f\n",MV),readchar(_),
                                 accept(MV).


/* DISTANCE */
/* D-base:Fuzzy,   Query:Fuzzy <the similary matrix>*/

fuzzyeval(uk("near"),"near","=",1.0):- !.
fuzzyeval(uk("near"),"far","=",0.1) :- !, accept(0.1).
fuzzyeval(uk("far"),"near","=",0.1) :- !, accept(0.1).
fuzzyeval(uk("far"),"far","=",1.0)  :- !.

fuzzyeval(uk("near"),"near","<",0.5):- !, write("###"),accept(0.5).
fuzzyeval(uk("near"),"far","<",1.0) :- !.
fuzzyeval(uk("far"),"far","<",0.5)  :- !, accept(0.5).

fuzzyeval(uk("near"),"near",">",0.5):- !, write("@@@"),accept(0.5).
fuzzyeval(uk("far"),"near",">",1.0) :- !.
fuzzyeval(uk("far"),"far",">",0.5)  :- !, accept(0.5).

/* SALARY */
/* D-base:Fuzzy,   Query:Fuzzy <the similary matrix>*/

fuzzyeval(uk("low"),"low","=",1.0).
fuzzyeval(uk("low"),"medium_salary","=",0.6) :- accept(0.6).
fuzzyeval(uk("low"),"high","=",0.1) :- accept(0.1).
fuzzyeval(uk("high"),"low","=",0.1) :- accept(0.1).
fuzzyeval(uk("high"),"high","=",1.0).
fuzzyeval(uk("high"),"medium_salary","=",0.7):- accept(0.7).
fuzzyeval(uk("medium_salary"),"low","=",0.6) :- accept(0.6).
```

```
fuzzyeval(uk("medium_salary"),"high","=",0.7):- accept(0.7).
fuzzyeval(uk("medium_salary"),"medium_salary","=",1.0).

fuzzyeval(uk("low"),"low","<",0.5):- accept(0.5).
fuzzyeval(uk("low"),"high","<",1.0).
fuzzyeval(uk("low"),"medium_salary","<",1.0).
fuzzyeval(uk("high"),"high","<",0.5):- accept(0.5).
fuzzyeval(uk("high"),"medium_salary","<",0.1):- accept(0.1).
fuzzyeval(uk("medium_salary"),"low","<",1.0).
fuzzyeval(uk("medium_salary"),"high","<",0.1):- accept(0.1).
fuzzyeval(uk("medium_salary"),"medium_salary","<",0.5):- accept(0.1).

fuzzyeval(uk("low"),"low",">",0.5):- accept(0.5).
fuzzyeval(uk("low"),"high",">",1.0).
fuzzyeval(uk("low"),"medium_salary",">",1.0).
fuzzyeval(uk("high"),"high",">",0.5):- accept(0.5).
fuzzyeval(uk("high"),"medium_salary",">",0.1):- accept(0.1).
fuzzyeval(uk("medium_salary"),"low",">",1.0).
fuzzyeval(uk("medium_salary"),"high",">",0.1):- accept(0.1).
fuzzyeval(uk("medium_salary"),"medium_salary",">",0.5):- accept(0.1).

/* HEIGHT */
/* D-base:Fuzzy,   Query:Fuzzy <the similary matrix>*/

fuzzyeval(uk("short"),"short","=",1.0).
fuzzyeval(uk("short"),"tall","=",0.1) :- accept(0.1).
fuzzyeval(uk("short"),"medium_height","=",0.6) :- accept(0.6).
fuzzyeval(uk("tall"),"short","=",0.1) :- accept(0.1).
fuzzyeval(uk("tall"),"tall","=",1.0).
fuzzyeval(uk("tall"),"medium_height","=",0.7):- accept(0.7).
fuzzyeval(uk("medium_height"),"short","=",0.6) :- accept(0.6).
fuzzyeval(uk("medium_height"),"tall","=",0.7):- accept(0.7).
fuzzyeval(uk("medium_height"),"medium_height","=",1.0).

fuzzyeval(uk("short"),"short","<",0.5):- accept(0.5).
fuzzyeval(uk("short"),"tall","<",1.0).
fuzzyeval(uk("short"),"medium_height","<",1.0).
fuzzyeval(uk("tall"),"tall","<",0.5):- accept(0.5).
fuzzyeval(uk("tall"),"medium_height","<",0.1):- accept(0.1).
fuzzyeval(uk("medium_height"),"short","<",1.0).
fuzzyeval(uk("medium_height"),"tall","<",0.1):- accept(0.1).
fuzzyeval(uk("medium_height"),"medium_height","<",0.5):- accept(0.1).

fuzzyeval(uk("short"),"short",">",0.5):- accept(0.5).
fuzzyeval(uk("short"),"tall",">",1.0).
fuzzyeval(uk("short"),"medium_height",">",1.0).
fuzzyeval(uk("tall"),"tall",">",0.5):- accept(0.5).
fuzzyeval(uk("tall"),"medium_height",">",0.1):- accept(0.1).
fuzzyeval(uk("medium_height"),"short",">",1.0).
fuzzyeval(uk("medium_height"),"tall",">",0.1):- accept(0.1).
fuzzyeval(uk("medium_height"),"medium_height",">",0.5):- accept(0.1).

/*****************************************************************/

     in_omega_range(Fuzzy,X,0,R) :- !, omega(Fuzzy,R,WU),
                                   0<=X, X<WU.

     in_omega_range(Fuzzy,X,R1,R2) :- !,
             omega(Fuzzy,R1,WL), omega(Fuzzy,R2,WU),
             WL<=X, X<WU.
```

```
     in_omega_range(Fuzzy,X,R) :- !, omega(Fuzzy,R,WL),
                                     WL<=X.


/*********************************************************************/
/*   membership_fn()     is the predicate that returns the mu-value    */
/*   for a fuzzy keyword evaluted at a given point (X).                */
/*   for example  membership_fn(1.6,"short",MV) will assign MV <- 0.65 */
/*********************************************************************/

     membership_fn(X,Fuzzy,MV) :- member(Fuzzy,["short","near","low"]),
                                  !, membership_fn1(X,Fuzzy,MV).

     membership_fn(X,Fuzzy,MV) :- member(Fuzzy,["tall","far","high"]),
                                  !, membership_fn2(X,Fuzzy,MV).

     membership_fn(X,Fuzzy,MV) :- member(Fuzzy,["medium_salary",
                                  "medium_height"]),
                                  !, membership_fn3(X,Fuzzy,MV).


     membership_fn1(X,Fuzzy,1.0):- in_omega_range(Fuzzy,X,0,1),!.

     membership_fn1(X,Fuzzy,MV) :- in_omega_range(Fuzzy,X,1,2),!,
                                   beta(Fuzzy,B),
                                   c_constant(Fuzzy,C),
                                   alpha(Fuzzy,A),
                                   MV = B/(1+(C/(X-A))*(C/(X-A))).

     membership_fn1(X,Fuzzy,0.0):- in_omega_range(Fuzzy,X,2),!.


     membership_fn2(X,Fuzzy,0.0):- in_omega_range(Fuzzy,X,0,1),!.

     membership_fn2(X,Fuzzy,0.0):- omega(Fuzzy,1,T),
                                   T=X, !. /* to prevent singularity */

     membership_fn2(X,Fuzzy,MV) :- in_omega_range(Fuzzy,X,1,2),!,
                                   beta(Fuzzy,B),
                                   c_constant(Fuzzy,C),
                                   alpha(Fuzzy,A),
                                   MV = B/(1+(C/(X-A))*(C/(X-A))).

     membership_fn2(X,Fuzzy,1.0):- in_omega_range(Fuzzy,X,2),!.


     membership_fn3(X,Fuzzy,0.0):- in_omega_range(Fuzzy,X,0,1),!.

     membership_fn3(X,Fuzzy,MV) :- in_omega_range(Fuzzy,X,1,2),!,
                                   c_constant(Fuzzy,C),
                                   alpha(Fuzzy,1,A),
                                   MV = C*(X-A)*(X-A).

     membership_fn3(X,Fuzzy,MV) :- in_omega_range(Fuzzy,X,2,3),!,
                                   c_constant(Fuzzy,C),
                                   alpha(Fuzzy,2,A),
                                   MV = 1-C*(X-A)*(X-A).

     membership_fn3(X,Fuzzy,MV) :- in_omega_range(Fuzzy,X,3,4),!,
```

```
                              c_constant(Fuzzy,C),
                              alpha(Fuzzy,3,A),
                              MV = C*(X-A)*(X-A).
membership_fn3(X,Fuzzy,0.0):- in_omega_range(Fuzzy,X,4),!.


/*********************************************************************/

integro(X,Sign,Fuzzy,MV) :- member(Fuzzy,["short","near","low"]),
                            !, integro1(X,Sign,Fuzzy,MV).

integro(X,Sign,Fuzzy,MV) :- member(Fuzzy,["tall","far","high"]),
                            !, integro2(X,Sign,Fuzzy,MV).

integro(X,Sign,Fuzzy,MV) :- member(Fuzzy,["medium_salary",
                            "medium_height"]),
                            !, integro3(X,Sign,Fuzzy,MV).

/*********************************************************************/
/*    integro1 will function if Fuzzy is "short"|"near"|"low"        */

integro1(X,"<",Fuzzy,MV) :-
                         in_omega_range(Fuzzy,X,0,1),!,
                         ff1(X,Fuzzy,F1),
                         ksi(Fuzzy,Ksi),
                         Area = -F1+Ksi,
                         sigma(Fuzzy,Sigma),
                         MV = Area/Sigma.

integro1(X,"<",Fuzzy,MV) :-
                         in_omega_range(Fuzzy,X,1,2),!,
                         ff2(X,Fuzzy,F2),
                         eta(Fuzzy,Eta),
                         Area = -F2+Eta,
                         sigma(Fuzzy,Sigma),
                         MV = Area/Sigma.

integro1(X,"<",Fuzzy,0.0) :-
                         in_omega_range(Fuzzy,X,2),!.

integro1(X,">",Fuzzy,MV) :-
                         in_omega_range(Fuzzy,X,0,1),!,
                         ff1(X,Fuzzy,Area),
                         sigma(Fuzzy,Sigma),
                         MV = Area/Sigma.

integro1(X,">",Fuzzy,MV) :-
                         in_omega_range(Fuzzy,X,1,2),!,
                         ff2(X,Fuzzy,F2),
                         rho(Fuzzy,Rho),
                         Area = F2+Rho,
                         sigma(Fuzzy,Sigma),
                         MV = Area/Sigma.

integro1(X,">",Fuzzy,1.0) :-
                         in_omega_range(Fuzzy,X,2),!.


/*********************************************************************/
/*    integro2 will function if Fuzzy is "tall","far","high"         */
```

```
     integro2(X,"<",Fuzzy,1.0) :-
                              in_omega_range(Fuzzy,X,0,1),!.

     integro2(X,"<",Fuzzy,MV) :-
                              in_omega_range(Fuzzy,X,1,2),!,
                              ff2(X,Fuzzy,F2),
                              ksi(Fuzzy,Ksi),
                              Area = -F2+Ksi,
                              sigma(Fuzzy,Sigma),
                              MV = Area/Sigma.

     integro2(X,"<",Fuzzy,MV) :-
                              in_omega_range(Fuzzy,X,2,3),!,
                              ff1(X,Fuzzy,F1),
                              eta(Fuzzy,Eta),
                              Area = -F1+Eta,
                              sigma(Fuzzy,Sigma),
                              MV = Area/Sigma.

     integro2(X,">",Fuzzy,0.0) :-
                              in_omega_range(Fuzzy,X,0,1),!.

     integro2(X,">",Fuzzy,MV) :-
                              in_omega_range(Fuzzy,X,1,2),!,
                              ff2(X,Fuzzy,F2),
                              rho(Fuzzy,Rho),
                              Area = F2+Rho,
                              sigma(Fuzzy,Sigma),
                              MV = Area/Sigma.

     integro2(X,">",Fuzzy,MV) :-
                              in_omega_range(Fuzzy,X,2,3),!,
                              ff1(X,Fuzzy,F1),
                              delta(Fuzzy,Delta),
                              Area = F1 + Delta,
                              sigma(Fuzzy,Sigma),
                              MV = Area/Sigma.


/************************************************************************/
/*    integro3 will function if Fuzzy is "medium_salary"|"medium_height" */

     integro3(X,"<",Fuzzy,1.0) :-
                              in_omega_range(Fuzzy,X,0,1),!.

     integro3(X,"<",Fuzzy,MV) :-
                              in_omega_range(Fuzzy,X,1,2),!,
                              f1(X,Fuzzy,F1),
                              ksi(Fuzzy,Ksi),
                              Area = -F1+Ksi,
                              sigma(Fuzzy,Sigma),
                              MV = Area/Sigma.

     integro3(X,"<",Fuzzy,MV) :-
                              in_omega_range(Fuzzy,X,2,3),!,
                              f2(X,Fuzzy,F2),
                              eta(Fuzzy,Eta),
                              Area = -F2+Eta,
                              sigma(Fuzzy,Sigma),
```

```
                        MV = Area/Sigma.

integro3(X,"<",Fuzzy,MV) :-
                        in_omega_range(Fuzzy,X,3,4),!,
                        f3(X,Fuzzy,F3),
                        rho(Fuzzy,Rho),
                        Area = -F3+Rho,
                        sigma(Fuzzy,Sigma),
                        MV = Area/Sigma.

integro3(X,"<",Fuzzy,0.0) :-
                        in_omega_range(Fuzzy,X,4),!.


integro3(X,">",Fuzzy,0.0) :-
                        in_omega_range(Fuzzy,X,0,1),!.


integro3(X,">",Fuzzy,MV) :-
                        in_omega_range(Fuzzy,X,1,2),!,
                        f1(X,Fuzzy,F1),
                        delta(Fuzzy,Delta),
                        Area = F1+Delta,
                        sigma(Fuzzy,Sigma),
                        MV = Area/Sigma.

integro3(X,">",Fuzzy,MV) :-
                        in_omega_range(Fuzzy,X,2,3),!,
                        f2(X,Fuzzy,F2),
                        lambda(Fuzzy,Lambda),
                        Area = F2+Lambda,
                        sigma(Fuzzy,Sigma),
                        MV = Area/Sigma.

integro3(X,">",Fuzzy,MV) :-
                        in_omega_range(Fuzzy,X,3,4),!,
                        f3(X,Fuzzy,F3),
                        nu(Fuzzy,Nu),
                        Area = F3+Nu,
                        sigma(Fuzzy,Sigma),
                        MV = Area/Sigma.

integro3(X,">",Fuzzy,1.0) :-
                        in_omega_range(Fuzzy,X,4),!.

/*************************************************************************/

ff1(X,Fuzzy,R) :- X=R.
ff2(X,Fuzzy,R) :- alpha(Fuzzy,A), beta(Fuzzy,B), c_constant(Fuzzy,C),
                  R = B*(X-C*arctan((X-A)/C)).
f1(X,Fuzzy,R) :- g(X,Fuzzy,1,R).
f2(X,Fuzzy,R) :- g(X,Fuzzy,2,G), R = X - G.
f3(X,Fuzzy,R) :- g(X,Fuzzy,3,R).
g(X,Fuzzy,N,G):- alpha(Fuzzy,N,A), c_constant(Fuzzy,C),
                 G = C*(X*X*X/3.0-A*X*X+A*A*X).

alpha("short",1.7).
alpha("near",7.0).
alpha("low",4.0).
alpha("tall",1.65).
```

```
alpha("far",3.5).
alpha("high",5.5).
alpha("medium_salary",1,3.0).
alpha("medium_salary",2,5.0).
alpha("medium_salary",3,7.0).
alpha("medium_height",1,1.4).
alpha("medium_height",2,1.7).
alpha("medium_height",3,2.0).
beta("short",1.05).
beta("near",1.083).
beta("low",1.087).
beta("tall",1.05).
beta("far",1.02).
beta("high",1.02).
c_constant("short",0.08).
c_constant("near",2.0).
c_constant("low",1.2).
c_constant("tall",0.08).
c_constant("far",2.0).
c_constant("high",2.0).
c_constant("medium_salary",0.5).
c_constant("medium_height",22.222).
omega("short",1,1.35).
omega("short",2,1.7).
omega("near",1,0.0).
omega("near",2,7.0).
omega("low",1,0.0).
omega("low",2,4.0).
omega("tall",1,1.65).
omega("tall",2,2.0).
omega("tall",3,2.4).
omega("far",1,3.5).
omega("far",2,15.0).
omega("far",3,30.0).
omega("high",1,5.5).
omega("high",2,20.0).
omega("high",3,40.0).
omega("medium_salary",1,3.0).
omega("medium_salary",2,4.0).
omega("medium_salary",3,6.0).
omega("medium_salary",4,7.0).
omega("medium_height",1,1.4).
omega("medium_height",2,1.55).
omega("medium_height",3,1.85).
omega("medium_height",4,2.0).
ksi("short",1.60443).
ksi("near",4.78145).
ksi("low",2.67923).
ksi("tall",2.38693).
ksi("far",27.4468).
ksi("high",37.4752).
ksi("medium_salary",6.4492).
ksi("medium_height",20.6257).
eta("short",1.785).
eta("near",7.581).
eta("low",4.348).
eta("tall",2.4).
eta("far",30.0).
eta("high",40.0).
eta("medium_salary",-14.8334).
```

```
    eta("medium_height",-34.5422).
    rho("short",-0.18057).
    rho("near",-2.79955).
    rho("low",-1.66877).
    rho("tall",-1.7325).
    rho("far",-3.57).
    rho("high",-5.61).
    rho("medium_salary",57.1666).
    rho("medium_height",59.2587).
    delta("tall",-1.74557).
    delta("far",-6.1232).
    delta("high",-8.1348).
    delta("medium_salary",-4.5).
    delta("medium_height",-20.3257).
    lambda("medium_salary",16.8326).
    lambda("medium_height",34.8417).
    nu("medium_salary",-55.1674).
    nu("medium_height",-58.9592).
    sigma("short",1.60443).
    sigma("near",4.78145).
    sigma("low",2.67923).
    sigma("tall",0.65443).
    sigma("far",23.8768).
    sigma("high",31.8652).
    sigma("medium_salary",1.92).
    sigma("medium_height",0.2995).


/*******************************************************************
Field action
*******************************************************************/

  field_action(_):-fail.

/*******************************************************************
 Field_value
*******************************************************************/

  field_value(FNAME,VAL):-value(FNAME,VAL),!.

/*******************************************************************
 noinput
*******************************************************************/

  noinput(_):-fail.
  noinput("Total").


/* window for asking the main choice */
  mainm:-makewindow(2,21,70,"",0,0,25,80),
         makewindow(1,56,113,"",0,0,24,80),
         cursor(20,0),
         makestatus(23,"Non-First Normal Form"),
         menu(10,25,21,19,["Add record","Delete record","Query database"],
              "Please enter your choice",1,Choice),
         do(Choice).

/* according to the choice returned it will call one
of add, delete or query predicates */
  do(1):- retractall(field(_,_,_,_,_)),
```

```
          retractall(txtfield(_,_,_,_)),
          retractall(value(_,_)),
          makewindow(5,56,113," ",0,0,25,80),
         add,!.
/*  do(2):-delete,!.  */
  do(3):-query,!.


/* window for relation type selection for addition */
  add:-makewindow(2,21,70,"",0,0,25,80),
       makewindow(1,56,113,"",0,0,24,80),
       cursor(20,0),
       makestatus(23,"Record Addition into Non-First Normal Form Database"),
       menu(10,25,21,19,["Add Employee Record","Add Student Record"],
              "Please enter your choice",1,Qchoice),
       doperson(Qchoice).



learn_info(No,Children):-
          cursor(0,0),
          consult("empadd.scr"),
          changestatus(" F10 to End "),
          scrhnd(off,KEY1),
          not(KEY1=esc),
          value("E_s",Es),
          gts(Es,Sresult),
          learn_child(Children),
          assertz(employee(No,Sresult,Children),owndatabase).

gts(Es,uk(Es)):-frontchar(Es,Ch,_),
                            char_int(Ch,Ascii4),
                            Ascii4>97,!.
gts(Es,k(E2)):-str_real(Es,E2).

learn_child([H|T]):-ask_single_child(H),
                    learn_child(T).

learn_child([]).
ask_single_child(Outcome):-
          retractall(field(_,_,_,_,_)),
          retractall(txtfield(_,_,_,_)),
          retractall(value(_,_)),
          makewindow(5,56,113," ",0,0,25,80),
          consult("childn.scr"),
          changestatus(" F10 to End "),
          scrhnd(off,KEY2),
          not(KEY2=esc),
          get(Name),
          write(Name),
          process(Name,Outcome),
          write(Outcome).

get(N):-value("Cname",N).

process(last,_):-!,fail.

process(N,child(N,ch_age(L_age,U_age))):-
          retractall(field(_,_,_,_,_)),
          retractall(txtfield(_,_,_,_)),
```

```
          retractall(value(_,_)),
          makewindow(5,56,113," ",0,0,25,80),
          consult("childa.scr"),
   changestatus(" F10 to End "),
          scrhnd(off,KEY3),
          not(KEY3=esc),
          value("La",Lage),
          str_int(Lage,L_age),
          value("Ua",Uage),
          str_int(Uage,U_age).


/* according to the Qchoice returned it will prepare a window
for adding a record to employee ( Qchoice=1 ) or to
student ( Qchoice =2 ) */


  doadd(1,No):-
            retractall(field(_,_,_,_,_)),
            retractall(txtfield(_,_,_,_)),
            retractall(value(_,_)),
            makewindow(5,56,113," ",0,0,25,80),
            learn_info(No,Children),!.



  doadd(2,No):-
          retractall(field(_,_,_,_,_)),
          retractall(txtfield(_,_,_,_)),
          retractall(value(_,_)),
          makewindow(5,56,113," ",0,0,25,80),
          consult("stadd.scr"),
          changestatus(" F10 to End "),
          scrhnd(off,KEY4),
          not(KEY4=esc),
          value("Fn",F_name),
          value("Dn",D_name),
          value("Dc",Dco),
          str_int(Dco,D_code),
          value("Intel",In),
          learn_course(L2),
          assertz(student(No,L2,mdept(dept_name(F_name,D_name),D_code),In),
                         owndatabase),!.

   doperson(Qchoice):-
          makewindow(5,56,113," ",0,0,25,80),
          consult("padd.scr"),
          changestatus(" F10 to End "),
          scrhnd(off,KEY4),
          not(KEY4=esc),
          value("Namep",Pname),
          value("Nop",Pno),
          str_real(Pno,No),
          write(No),
          readkey(_),
          gtn(Qchoice,No,Nresult),
          write(Nresult),nl,
          value("Telp",Tno),
          gtt(Tno,Tresult),
          write(Tresult),nl,
```

```
        value("Agepl",Pagel),
        str_int(Pagel,Agel),
        write(Agel),nl,
        value("Agepu",Pageu),
        str_int(Pageu,Ageu),
        write(Ageu),nl,
        value("Addp",Paddress),
        write(Paddress),nl,
        value("Heightp",Pheight),
        gt(Pheight,Presult),
        write(Presult),nl,
        value("Dis",Distance),
        gtd(Distance,Dresult),
        write(Dresult),nl,
        assertz(person(Pname,Nresult,Tresult,
                age(Agel,Ageu),Paddress,Presult,Dresult,Qchoice),
                owndatabase),
        doadd(Qchoice,No).


gtn(1,Nop,e(Nop)):-!.
gtn(2,Nop,s(Nop)).

gt(Pheight,uk(Pheight)):-frontchar(Pheight,Ch2,_),
                            char_int(Ch2,Ascii2),
                            Ascii2>97,!.
gt(Pheight,k(Phe)):-str_real(Pheight,Phe).

gtd(Distance,uk(Distance)):-frontchar(Distance,Ch1,_),
                            char_int(Ch1,Ascii1),
                            Ascii1>97,!.
gtd(Distance,k(Pdis)):-str_real(Distance,Pdis).

gtt(Tno,uk(Tno)):-frontchar(Tno,Ch,_),
                            char_int(Ch,Ascii),
                            Ascii>97,!.
gtt(Tno,k(T_no)):-str_real(Tno,T_no).


/* if the entered values does not correspond to the last
course it will ask for other informations about the course */
  learn_course([H|T]):-ask_single_course(H),
        learn_course(T).

  learn_course([]).

  /* for getting course information */
  ask_single_course(C_no):-
        retractall(field(_,_,_,_,_)),
        retractall(txtfield(_,_,_,_)),
        retractall(value(_,_)),
        makewindow(5,56,113," ",0,0,25,80),
        consult("coursen.scr"),
        changestatus(" F10 to End "),
        scrhnd(off,KEY5),
        not(KEY5=esc),
        value("Cn",Cno),

        str_int(Cno,C_no),
        processc(C_no).
```

```
/* Checks if the entered course number is 0 or not */
  processc(0):-!,fail.
  processc(_).



/************************************************/

  query:-makewindow(2,21,70,"",0,0,25,80),
         makewindow(1,56,113,"",0,0,24,80),
         cursor(20,0),
         makestatus(23,"Quering Non-First Normal Form Database"),
         write("Enter query relation:"),
         readln(S),
         str_tokenlist(S,L),
         match(exp,L,Q),
         eval(Q,Reslist),
         write_reslist(Reslist),
         readchar(_),
         removewindow,
         removewindow.

  delete:-makewindow(2,21,70,"",0,0,25,80),
         makewindow(1,56,113,"",0,0,24,80),
         cursor(20,0),
         makestatus(23,"Deleting From Non-First Normal Form Database"),
         write("Enter deletion relation:"),
         readln(S),
         str_tokenlist(S,L),
         match(exp,L,Q),
         eval(Q,Reslist),
         write_reslist(Reslist),
         delete_slave(Reslist),
         readchar(_),
         removewindow,
         removewindow.

  delete_slave(Reslist) :-
         confirm("Will be deleted..."),
         dodelete(Reslist),
         write("DELETION IS DONE."),nl,!.

  delete_slave(_) :- write("NO DELETION IS PERFORMED."),nl.


getattp("=","name",Name,Perlist) :-
        findall(X,transp(Name,_,_,_,_,_,_,_,X),Perlist).

getattp("=","no",N,Perlist) :-
       str_real(N,N1),
       gtn(1,N1,No),
       findall(X,transp(_,No,_,_,_,_,_,_,X),Perlist).

getattp("=","tel",T,Perlist) :-
       gtt(T,Tel),
       findall(X,transp(_,_,Tel,_,_,_,_,_,X),Perlist).

/************************/
```

```
getattp("=","age",Age,Perlist) :-
        findall(X,transp2(_,_,_,Age,_,_,_,_,X),Perlist).

getattp("=","address",Address,Perlist) :-
       findall(X,transp(_,_,_,_,Address,_,_,_,X),Perlist).


getattp("=","height",Height,Perlist) :-
       findall(X,tpehe(Height,X),Perlist).

getattp("=","distance",Distance,Perlist) :-
       findall(X,tpedis(Distance,X),Perlist).

getattp("<","name",Name,Perlist):-
       findall(X,tplna(Name,X),Perlist).

getattp("<","no",No,Perlist):-
       findall(X,tplno(No,X),Perlist).

getattp("<","age",Age,Perlist):-
       findall(X,tplage(Age,X),Perlist).

getattp("<","address",Address,Perlist):-
       findall(X,tpladd(Address,X),Perlist).


getattp("<","height",Height,Perlist):-
       findall(X,tplhe(Height,X),Perlist).

getattp("<","distance",Distance,Perlist):-
       findall(X,tpldis(Distance,X),Perlist).


getattp(">","name",Name,Perlist):-
       findall(X,tpgna(Name,X),Perlist).

getattp(">","no",No,Perlist):-
       findall(X,tpgno(No,X),Perlist).

getattp(">","age",Age,Perlist):-
       findall(X,tpgage(Age,X),Perlist).

getattp(">","address",Address,Perlist):-
       findall(X,tpgadd(Address,X),Perlist).

getattp(">","height",Height,Perlist):-
       findall(X,tpghe(Height,X),Perlist).

getattp(">","distance",Distance,Perlist):-
       findall(X,tpgdis(Distance,X),Perlist).

/*********/
tpehe(He,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,MV)):-
       person(Name,No,Tel,Age,Address,Height,Distance,Tp),
       fuzzyeval(Height,He,"=",height,MV).

tpedis(Dis,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,MV)):-
       person(Name,No,Tel,Age,Address,Height,Distance,Tp),
       fuzzyeval(Distance,Dis,"=",distance,MV).
```

```
teesal(Sal,femployee(Eno,Salary,Chl,MV)):-
       employee(Eno,Salary,Chl),
       fuzzyeval(Salary,Sal,"=",salary,MV).

/*********/


transp(Name,No,Tel,Age,Address,Height,Distance,Tp,
       fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,1)):-
         person(Name,No,Tel,Age,Address,Height,Distance,Tp).

transp2(Name,No,Tel,Age,Address,Height,Distance,Tp,
        fperson(Name,No,Tel,age(Age1,Age2),Address,Height,Distance,Tp,100)):-
         str_int(Age,Ag),
         person(Name,No,Tel,age(Age1,Age2),Address,Height,Distance,Tp),
         Ag>=Age1,
         Ag>=Age2.

tpgna(Na,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,1.0)):-
         person(Name,No,Tel,Age,Address,Height,Distance,Tp),
         Name>Na.

tpghe(He,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,MV)):-
       person(Name,No,Tel,Age,Address,Height,Distance,Tp),
       fuzzyeval(Height,He,">",MV).

tpgdis(Dis,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,MV)):-
       person(Name,No,Tel,Age,Address,Height,Distance,Tp),
       fuzzyeval(Distance,Dis,">",MV).



tegcage(Ca,femployee(Eno,Salary,Chl,1.0)):-
       str_real(Ca,Ca1),
       employee(Eno,Salary,Chl),
       member(X,Chl),
       X = child(Cname,ch_age(Al,Au)),
       Au>Ca1.

tegsal(Sal,femployee(Eno,Salary,Chl,MV)):-
       employee(Eno,Salary,Chl),
       fuzzyeval(Salary,Sal,">",MV).

tegcna(Cna,femployee(Eno,Salary,Chl,1.0)):-
       employee(Eno,Salary,Chl),
       member(X,Chl),
       X = child(Cname,Cage),
       Cname>Cna.



tpgadd(Add,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,1.0)):-
        person(Name,No,Tel,Age,Address,Height,Distance,Tp),
        Address>Add.

tpgage(A,fperson(Name,No,Tel,age(Age1,Age2),
                 Address,Height,Distance,Tp,1.0)):-
       str_int(A,Ag),
       person(Name,No,Tel,age(Age1,Age2),Address,Height,Distance,Tp),
```

```
        Age2>Ag.

tpgno(N,fperson(Name,e(No),Tel,Age,Address,Height,Distance,1,1.0)):-
        str_int(N,N1),
        person(Name,e(No),Tel,Age,Address,Height,Distance,1),
        No>N1.

tpgno(N,fperson(Name,s(No),Tel,Age,Address,Height,Distance,2,1.0)):-
        str_int(N,N1),
        person(Name,s(No),Tel,Age,Address,Height,Distance,2),
        No>N1.

tplna(Na,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,1.0)):-
          person(Name,No,Tel,Age,Address,Height,Distance,Tp),
          Name>Na.

tplhe(He,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,MV)):-
        person(Name,No,Tel,Age,Address,Height,Distance,Tp),
        fuzzyeval(Height,He,">",MV).



tpldis(Dis,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,MV)):-
        person(Name,No,Tel,Age,Address,Height,Distance,Tp),
        fuzzyeval(Distance,Dis,"<",MV).


telsal(Sal,femployee(Eno,Salary,Chl,MV)):-
        employee(Eno,Salary,Chl),
        fuzzyeval(Salary,Sal,"<",MV).


telcage(Ca,femployee(Eno,Salary,Chl,1.0)):-
        str_real(Ca,Ca1),
        employee(Eno,Salary,Chl),
        member(X,Chl),
        X = child(Cname,ch_age(Al,Au)),
        Al<Ca1.


telcna(Cna,femployee(Eno,Salary,Chl,1.0)):-
        employee(Eno,Salary,Chl),
        member(X,Chl),
        X = child(Cname,Cage),
        Cname<Cna.



tpladd(Add,fperson(Name,No,Tel,Age,Address,Height,Distance,Tp,1.0)):-
         person(Name,No,Tel,Age,Address,Height,Distance,Tp),
         Address<Add.

tplage(A,fperson(Name,No,Tel,
                age(Age1,Age2),Address,Height,Distance,Tp,1.0)):-
        str_int(A,Ag),
        person(Name,No,Tel,age(Age1,Age2),Address,Height,Distance,Tp),
        Age1<Ag.

tplno(N,fperson(Name,e(No),Tel,Age,Address,Height,Distance,1,1.0)):-
        str_int(N,N1),
```

```
        person(Name,e(No),Tel,Age,Address,Height,Distance,1),
        No<N1.

tplno(N,fperson(Name,s(No),Tel,Age,Address,Height,Distance,2,1.0)):-
        str_int(N,N1),
        person(Name,s(No),Tel,Age,Address,Height,Distance,2),
        No<N1.


transe(Eno,S,femployee(Eno,Salary,Chl,MV)):-
        employee(Eno,Salary,Chl),
        fuzzyeval(Salary,S,"=",salary,MV).

transe2(Eno,Cname,Cha,femployee(Eno,Salary,Chl,1.0)):-
        str_int(Cha,Ca),
        employee(Eno,Salary,Chl),
        member(X,Chl),
        X = child(Cname,ch_age(Ca1,Ca2)),
        Ca>Ca1,
        Ca>=Ca2.

transe3(Eno,Cname,femployee(Eno,Salary,Chl,1.0)):-
        employee(Eno,Salary,Chl),
        member(X,Chl),
        X = child(Cname,_).


dodelete([fresult1(E_no,_,_,_,_,_,_,_,_,_,_)|T]) :-
           retract(person(_,e(E_no),_,_,_,_,_,_)),
           retract(employee(E_no,_,_)),
           dodelete(T).
dodelete([fresult2(S_no,_,_,_,_,_,_,_,_,_,_,_)|T]) :-
           retract(person(_,s(S_no),_,_,_,_,_,_)),
           retract(student(S_no,_,_,_)),
           dodelete(T).

dodelete([]).

/* findresult:
      1.Arg:  is a list of one of the following
                 a) fperson(...)
                 b) femployee(...)
                 c) fstudent(...)
      2.Arg: is intended to return   fresult(...) which
             packes togather all the missing info, the
             link is over IP number.
*/

findresult([],[]).

findresult([femployee(E_no,Salary,Childl,Mv)|T],
           [fresult1(E_no,Salary,Childl,Name,Tel,
                    Age,Adress,Height,Distan,1,Mv)
                 |Result]) :-
        person(Name,e(E_no),Tel,Age,Adress,Height,Distan,1), !,
        findresult(T,Result).

findresult([fstudent(S_no,Courses,Mdept,Intel,Mv)|T],
           [fresult2(S_no,Courses,Mdept,Intel,Name,Tel,
                    Age,Adress,Height,Distan,2,Mv)
```

```
                 |Result]) :-
       person(Name,s(S_no),Tel,Age,Adress,Height,Distan,2), !,
       findresult(T,Result).

findresult([fperson(Name,e(E_no),Tel,Age,Adress,Height,Distan,1,Mv)|T],
           [fresult1(E_no,Salary,Childl,Name,Tel,
                     Age,Adress,Height,Distan,1,Mv)
                   |Result]) :-
       employee(E_no,Salary,Childl), !,
       findresult(T,Result).

findresult([fperson(Name,s(S_no),Tel,Age,Adress,Height,Distan,2,Mv)|T],
           [fresult2(S_no,Courses,Mdept,Intel,Name,Tel,
                     Age,Adress,Height,Distan,2,Mv)
                   |Result]) :-
       student(S_no,Courses,Mdept,Intel), !,
       findresult(T,Result).

getatte("=","salary",Salary,Emplist):-
       findall(Y,transe(Eno,Salary,Y),Emplist).

getatte("=","cname",Cname,Emplist):-
       findall(Y,transe3(Eno,Cname,Y),Emplist).

getatte("=","cage",Cage,Emplist):-
       findall(Y,transe2(Eno,Cname,Cage,Y),Emplist).



getatte("<","salary",Salary,Emplist):-
       findall(Y,telsal(Salary,Y),Emplist).

getatte("<","cname",Cname,Emplist):-
       findall(Y,telcna(Cname,Y),Emplist).

getatte("<","cage",Cage,Emplist):-
       findall(Y,telcage(Cage,Y),Emplist).

/***   GREATER THAN ****/


getatte(">","salary",Salary,Emplist):-
       findall(Y,tegsal(Salary,Y),Emplist).

getatte(">","cname",Cname,Emplist):-
       findall(Y,tegcna(Cname,Y),Emplist).

getatte(">","cage",Cage,Emplist):-
       findall(Y,tegcage(Cage,Y),Emplist).



/*****   STUDENT   ******/



getatts("=","courses",Co,Stlist):-
       str_int(Co,Courses),
       findall(Y,transs2(_,Courses,_,_,_,_,Y),Stlist).
```

```
getatts("=","facname",Facname,Stlist):-
       findall(Y,transs2(_,_,Facname,_,_,_,Y),Stlist).

getatts("=","depname",Depname,Stlist):-
       findall(Y,transs2(_,_,_,Depname,_,_,Y),Stlist).

getatts("=","depno",Depn,Stlist):-
       str_int(Depn,Depno),
       findall(Y,transs2(_,_,_,_,Depno,_,Y),Stlist).

getatts("=","intel",Intel,Stlist):-
       findall(Y,transs2(_,_,_,_,_,Intel,Y),Stlist).


/*less*/


getatts("<","facname",Facname,Stlist):-
       findall(Y,tslfac(Facname,Y),Stlist).

getatts("<","depname",Depname,Stlist):-
       findall(Y,tsldna(Depname,Y),Stlist).

getatts("<","depno",Depno,Stlist):-
       findall(Y,tsldno(Depno,Y),Stlist).




/***   GREATER THAN ****/

getatts(">","facname",Facname,Stlist):-
       findall(Y,tsgfac(Facname,Y),Stlist).

getatts(">","depname",Depname,Stlist):-
       findall(Y,tsgdna(Depname,Y),Stlist).

getatts(">","depno",Depno,Stlist):-
       findall(Y,tsgdno(Depno,Y),Stlist).


transs2(Sno,Courses,Facname,Depname,Depno,Intel,
        fstudent(Sno,Coursel,mdept(dept_name(Facname,Depname),
                                   Depno),Intel,1.0)):-
       student(Sno,Coursel,mdept(dept_name(Facname,Depname),
                                   Depno),Intel),
       member(Courses,Coursel).

tsldna(Depn,fstudent(Sno,Coursel,mdept(dept_name(Facname,Depname),
                                       Depno),Intel,1.0)):-
       student(Sno,Coursel,mdept(dept_name(Facname,Depname),
                                       Depno),Intel),
       Depname<Depn.

tsldno(Dno,fstudent(Sno,Coursel,mdept(dept_name(Facname,Depname),
                                      Depno),Intel,1.0)):-
       str_int(Dno,Dn),
       student(Sno,Coursel,mdept(dept_name(Facname,Depname),Depno),Intel),
       Depno<Dn.
```

```
tslfac(Facn,fstudent(Sno,Coursel,mdept(dept_name(Facname,Depname),
                                       Depno),Intel,1.0)):-
       student(Sno,Coursel,mdept(dept_name(Facname,Depname),Depno),Intel),
       Facname<Facn.

tsgdna(Depn,fstudent(Sno,Coursel,mdept(dept_name(Facname,Depname),
                                       Depno),Intel,1.0)):-
       student(Sno,Coursel,mdept(dept_name(Facname,Depname),Depno),Intel),
       Depname>Depn.

tsgdno(Dn,fstudent(Sno,Coursel,mdept(dept_name(Facname,Depname),
                                       Depno),Intel,1.0)):-
       str_int(Dn,Dno),
       student(Sno,Coursel,mdept(dept_name(Facname,Depname),Depno),Intel),
       Depno>Dno.


tsgfac(Facn,fstudent(Sno,Coursel,mdept(dept_name(Facname,Depname),
                                       Depno),Intel,1.0)):-
       student(Sno,Coursel,mdept(dept_name(Facname,Depname),Depno),Intel),
       Facname>Facn.


/*  Various Writes for pretty printing */

write_perlist(L) :-
                   makewindow(5,3,7,"Person Output Screen",2,2,20,76),
                   write_perlist_slave(L,0),
                   readchar(_),
                   removewindow.


/* below the scond argument integer is
   the maximal count of lines of output
   before a clear window occures */

write_perlist_slave(L,10) :- readchar(_),clearwindow,!,
                             write_perlist_slave(L,0).

write_perlist_slave([fperson(Name,No,Tel,age(Lower,Upper),
                     Address,Height,Dist,Tp,MV)|Tail],
             Already_printed) :-
    writef("%-16s ",Name),
    write_tel(Tel),
    writef(" [%2d,%2d] %12s",Lower,Upper,Address),
    write_fuzzy(Height),
    writef(" "),
    write_fuzzy(Dist),
    writef(" %3d %4.2f",Tp,MV),
    nl,
    A = Already_printed+1,!,
    write_perlist_slave(Tail,A).

write_perlist_slave([],_).

write_fuzzy(k(F)) :- bound(F),writef("%8.2f",F),!.
write_fuzzy(uk(S)) :- bound(S),write(S).
write_tel(k(F)) :- bound(F),writef("%8.0f",F),!.
write_tel(uk(S)):- bound(S),write(S),!.
```

```
draw_line :- write(
  "----------------------------------------------------------------------"),
      nl.

wait :- cursor(Row,_), Row>=16,readchar(_),!,  clearwindow.
wait.

/**********************************************************************/


write_reslist(L) :-
                   makewindow(6,30,7,"Result Output Screen",0,2,24,76),
                   write_reslist_slave(L,0),
                   readchar(_),
                   removewindow.

/* below the second argument integer is the maximal count of lines of output
   before a clear window occures */

write_reslist_slave(L,10) :- readchar(_),clearwindow,!,
                             write_reslist_slave(L,0).

write_reslist_slave([fresult1(E_no,Salary,Childl,
                              Name,Tel,age(Lower,Upper),
                              Address,Height,Distan,Tp,MV)
                        |Tail],Already_printed) :-
    writef("Name:%-30s     Employee no:%5d     Age: [%2d,%2d] ",
              Name,E_no,Lower,Upper),nl,
    writef("Child(ren): "),
    write(Childl), nl,
    writef("Address: %-40s",Address),
    write("    Telephone: "),
    write_tel(Tel), nl,
    write("Height: "),
    write_fuzzy(Height),
    write("    Distance: "),
    write_fuzzy(Distan),
    write("    Salary: "),
    write_fuzzy(Salary),    nl,
    writef("Membership value: %4.2f",MV), nl,
    draw_line,
    wait,
    A = Already_printed+1,!,
    write_reslist_slave(Tail,A).

write_reslist_slave(
   [fresult2(S_no,Courses,mdept(dept_name(Fac,Dname),Dno),
             Intel,Name,Tel,age(Lower,Upper),Address,
             Height,Distan,Tp,MV)
       | Tail],
                   Already_printed) :-
    writef("Name:%-30s     Student no:%5d     Age: [%2d,%2d]",
              Name,S_no,Lower,Upper),nl,
    writef("Address: %-40s",Address),
    write("    Telephone: "),
    write_tel(Tel), nl,
    write("Height: "),
    write_fuzzy(Height),
    write("    Distance: "),
    write_fuzzy(Distan), nl,
```

```
    writef("Faculty: %12s   Department: %10s   Dept. no: %2d",
              Fac,Dname,Dno),nl,
    write("Courses: "),
    write(Courses),
    write("      Intelligence:",Intel), nl,
    writef("     Membership value: %4.2f",MV), nl,
    draw_line,
    wait,
    A = Already_printed+1,!,
    write_reslist_slave(Tail,A).

write_reslist_slave([],_).

confirm(S) :- concat(S," [y/n] :",R),
              makewindow(20,78,12,S,12,30,15,50),
              readchar(K), removewindow, upper_lower(K,'y').
```

**parseval**

```
project "tez"

include "global.pro"

predicates


split(lstring,lstring,lstring)
str_tokenlist_slave(string,lstring)
trim_tokenlist(lstring,lstring)
glue(string,string)

fuzzyand(rlist,rlist,rlist)
fuzzyor(rlist,rlist,rlist)
fuzzy_or_member(r,rlist)
fuzzy_and_member(r,rlist,r)
delete_ip(r,rlist,rlist)

max_real(real,real,real)
min_real(real,real,real)

clauses

/*************************************************************************/
/*                         P  A  R  S  E  R                              */
/*************************************************************************/


    match(exp,L,veya(R1,R2)) :-
            split(L,L1,["or"|L2]),
            match(term1,L1,R1), match(term1,L2,R2).

    match(exp,L,R)          :- match(term1,L,R).

    match(term1,L,ve(R1,R2)) :-
            split(L,L1,["and"|L2]),
            match(term2,L1,R1), match(term2,L2,R2).

    match(term1,L,R)        :- match(term2,L,R).

    match(term2,["("|L],R)  :- split(L,Exp,[")"]),
                               match(exp,Exp,R).

    match(term2,L,R)        :- match(kiyas,L,R).

    match(kiyas,[Left,Sign,Right],kiyas(Sign,Left,Right)) :-
            member(Sign,["<",">","=","<=",">="]).

    match(_,[X],o(X)).


    split(X,[],X).
    split([A|D],[A|B],C) :- split(D,B,C).

/*************************************************************************/

    str_tokenlist(String,L) :- str_tokenlist_slave(String,L1),
                               trim_tokenlist(L1,L).
```

```
    str_tokenlist_slave("",[]) :- !.
    str_tokenlist_slave(" ",[]) :- !.

    str_tokenlist_slave(String,[Token|R]) :-
            fronttoken(String,Token,Reststring),
            str_tokenlist_slave(Reststring,R).

    trim_tokenlist([],[]) :- !.
    trim_tokenlist([X,Y|R],[XY|Q]) :-
            glue(X,Y),
            !,concat(X,Y,XY),trim_tokenlist(R,Q).
    trim_tokenlist([X|R],[X|Q]) :- trim_tokenlist(R,Q).

    glue("<","=").
    glue(">","=").

/***************************************************************************/
/*                  E  V  A  L  U  A  T  O  R                             */
/***************************************************************************/

    eval(ve(E1,E2),R)   :-
            !, eval(E1,R1), eval(E2,R2), fuzzyand(R1,R2,R).

    eval(veya(E1,E2),R) :-
            !, eval(E1,R1), eval(E2,R2), fuzzyor(R1,R2,R).

    eval(kiyas(Comparision,FieldId,Attr_value) ,R) :-
            getattp(Comparision,FieldId,Attr_value,Perlist),!,
            findresult(Perlist,R).
    eval(kiyas(Comparision,FieldId,Attr_value) ,R) :-
            getatte(Comparision,FieldId,Attr_value,Emplist),!,
            findresult(Emplist,R).
    eval(kiyas(Comparision,FieldId,Attr_value) ,R) :-
            getatts(Comparision,FieldId,Attr_value,Stlist),!,
            findresult(Stlist,R).

/*-------------------------------------------------------------------------*/

fuzzyand([],_,[]) :- !.

fuzzyand([X|T],S,[Y|Q]) :-
                        fuzzy_and_member(X,S,Y), !,
                        fuzzyand(T,S,Q).

fuzzyand([_|T],S,Q) :- fuzzyand(T,S,Q).

/* in fuzzy_and_member :
       1.Arg: Candidate element (some packet info)
       2.Arg: A fuzzy set of the same type of the Candidate element.
       3.Arg: Is one of these:
              * The 1.Arg (if a member in the set (2.Arg) exist with
                the same IP, and 1.Arg has better MV)
              * A member of the set (2.Arg) (which matched the
                IP with the candidate but has a better MV)
  this predicate FAILs if the none of the above two conditions hold */

fuzzy_and_member(fresult1(IP,X1,X2,X3,X4,X5,X6,X7,X8,X9,MV),
                 [fresult1(IP,_,_,_,_,_,_,_,_,_,MV2)|_],
                 fresult1(IP,X1,X2,X3,X4,X5,X6,X7,X8,X9,MVR)) :- !,
```

```
                 min_real(MV,MV2,MVR).
fuzzy_and_member(fresult2(IP,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,MV),
                 [fresult2(IP,_,_,_,_,_,_,_,_,_,_,MV2)|_],
                 fresult2(IP,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,MVR)) :- !,
                 min_real(MV,MV2,MVR).
fuzzy_and_member(X,[_|T],MVR) :- fuzzy_and_member(X,T,MVR).

/*-----------------------------------------------------------------*/

fuzzyor([],S,S) :- !.
fuzzyor([X|T],S,[X|Q]) :-
                 fuzzy_or_member(X,S),!,
                 delete_ip(X,S,S2), /* if we are here:above succeeeded*/
                 fuzzyor(T,S2,Q).

fuzzyor([_|T],S,Q) :- fuzzyor(T,S,Q).


/* in fuzzy_or_member :
       1.Arg: Candidate element (some packet info)
       2.Arg: A fuzzy set of the same type of the Candidate element.
       predicate succeeds if one of the following holds:
             * The 1.Arg (if a member in the set (2.Arg) exist with
               the same IP, and 1.Arg has lower MV (that is favourable))
             * The 1.Arg does not exist in the set (2.Arg).  */

fuzzy_or_member(X,[]).
fuzzy_or_member(fresult1(IP,_,_,_,_,_,_,_,_,_,MV),
                [fresult1(IP,_,_,_,_,_,_,_,_,_,MV2)|_]) :- !, MV<MV2.
fuzzy_or_member(fresult2(IP,_,_,_,_,_,_,_,_,_,_,MV),
                [fresult2(IP,_,_,_,_,_,_,_,_,_,_,MV2)|_]) :- !, MV<MV2.
fuzzy_or_member(X,[_|T]) :- fuzzy_or_member(X,T).


delete_ip(_,[],[]).
delete_ip(fresult1(IP,_,_,_,_,_,_,_,_,_,_),
         [fresult1(IP,_,_,_,_,_,_,_,_,_,_)|T],T) :- !.
delete_ip(fresult2(IP,_,_,_,_,_,_,_,_,_,_,_),
         [fresult2(IP,_,_,_,_,_,_,_,_,_,_,_)|T],T) :- !.
delete_ip(X,[H|T],[H|R]) :- delete_ip(X,T,R).

max_real(N1,N2,N1) :- N1>N2,!.
max_real(_,N2,N2).

min_real(N1,N2,N1) :- N1<N2,!.
min_real(_,N2,N2).
```

```
project "tez"
include "global.pro"


/***************************************************************
     Turbo Prolog Toolbox
     (C) Copyright 1987 Borland International.

 This module includes some routines which are used in nearly
 all menu and screen tools.
***************************************************************/

/***************************************************************/
/* repeat */
/***************************************************************/

PREDICATES
  nondeterm repeat

CLAUSES
  repeat.
  repeat:-repeat.


/***************************************************************/
/* miscellaneous */
/***************************************************************/

PREDICATES
  maxlen(STRINGLIST,COL,COL)    /* The length of the longest string */
  listlen(STRINGLIST,ROW)       /* The length of a list             */
  writelist(ROW,COL,STRINGLIST) /* used in the menu predicates      */
  reverseattr(ATTR,ATTR)        /* Returns the reversed attribute   */
  min(ROW,ROW,ROW)
  min(COL,COL,COL)
  min(LEN,LEN,LEN)
  min(INTEGER,INTEGER,INTEGER)
  max(ROW,ROW,ROW)
  max(COL,COL,COL)
  max(LEN,LEN,LEN)
  max(INTEGER,INTEGER,INTEGER)

CLAUSES
  maxlen([H|T],MAX,MAX1) :-
str_len(H,LENGTH),
LENGTH>MAX,!,
maxlen(T,LENGTH,MAX1).
  maxlen([_|T],MAX,MAX1) :- maxlen(T,MAX,MAX1).
  maxlen([],LENGTH,LENGTH).

  listlen([],0).
  listlen([_|T],N):-
listlen(T,X),
N=X+1.

  writelist(_,_,[]).
  writelist(LI,ANTKOL,[H|T]):-
field_str(LI,0,ANTKOL,H),
```

```
LI1=LI+1,
writelist(LI1,ANTKOL,T).

  min(X,Y,X):-X<=Y,!.
  min(_,X,X).

  max(X,Y,X):-X>=Y,!.
  max(_,X,X).

  reverseattr(A1,A2):-
bitand(A1,$07,H11),
bitleft(H11,4,H12),
bitand(A1,$70,H21),
bitright(H21,4,H22),
bitand(A1,$08,H31),
A2=H12+H22+H31.


/**************************************************************/
/* Find letter selection in a list of strings */
/*      Look initially for first uppercase letter. */
/*      Then try with first letter of each string. */
/**************************************************************/

PREDICATES
  upc(CHAR,CHAR)  lowc(CHAR,CHAR)
  try_upper(CHAR,STRING)
  tryfirstupper(CHAR,STRINGLIST,ROW,ROW)
  tryfirstletter(CHAR,STRINGLIST,ROW,ROW)
  tryletter(CHAR,STRINGLIST,ROW)

CLAUSES
  upc(CHAR,CH):-
CHAR>='a',CHAR<='z',!,
char_int(CHAR,CI), CI1=CI-32, char_int(CH,CI1).
  upc(CH,CH).

  lowc(CHAR,CH):-
CHAR>='A',CHAR<='Z',!,
char_int(CHAR,CI), CI1=CI+32, char_int(CH,CI1).
  lowc(CH,CH).

  try_upper(CHAR,STRING):-
frontchar(STRING,CH,_),
CH>='A',CH<='Z',!,
CH=CHAR.
  try_upper(CHAR,STRING):-
frontchar(STRING,_,REST),
try_upper(CHAR,REST).

  tryfirstupper(CHAR,[W|_],N,N) :-
try_upper(CHAR,W),!.
  tryfirstupper(CHAR,[_|T],N1,N2) :-
N3 = N1+1,
tryfirstupper(CHAR,T,N3,N2).

  tryfirstletter(CHAR,[W|_],N,N) :-
frontchar(W,CHAR,_),!.
  tryfirstletter(CHAR,[_|T],N1,N2) :-
N3 = N1+1,
```

```
tryfirstletter(CHAR,T,N3,N2).

  tryletter(CHAR,LIST,SELECTION):-
upc(CHAR,CH),tryfirstupper(CH,LIST,0,SELECTION),!.
  tryletter(CHAR,LIST,SELECTION):-
lowc(CHAR,CH),tryfirstletter(CH,LIST,0,SELECTION).



/****************************************************************/
/* adjustwindow takes a windowstart and a windowsize and adjusts */
/* the windowstart so the window can be placed on the screen.  */
/* adjframe looks at the frameattribute: if it is different from */
/* zero, two is added to the size of the window  */
/***************************************************************/

PREDICATES
  adjustwindow(ROW,COL,ROW,COL,ROW,COL)
  adjframe(ATTR,ROW,COL,ROW,COL)

CLAUSES
  adjustwindow(LI,KOL,DLI,DKOL,ALI,AKOL):-
LI<25-DLI,KOL<80-DKOL,!,ALI=LI,AKOL=KOL.
  adjustwindow(LI,_,DLI,DKOL,ALI,AKOL):-
LI<25-DLI,!,ALI=LI,AKOL=80-DKOL.
  adjustwindow(_,KOL,DLI,DKOL,ALI,AKOL):-
KOL<80-DKOL,!,ALI=25-DLI, AKOL=KOL.
  adjustwindow(_,_,DLI,DKOL,ALI,AKOL):-
ALI=25-DLI, AKOL=80-DKOL.

  adjframe(0,R,C,R,C):-!.
  adjframe(_,R1,C1,R2,C2):-R2=R1+2, C2=C1+2.


/***************************************************************/
/*  Readkey */
/* Returns a symbolic key from the KEY domain         */
/***************************************************************/

PREDICATES
/*   readkey(KEY) moved to global */
  readkey1(KEY,CHAR,INTEGER)
  readkey2(KEY,INTEGER)

CLAUSES
  readkey(KEY):-readchar(T),char_int(T,VAL),readkey1(KEY,T,VAL).

  readkey1(KEY,_,0):-!,readchar(T),char_int(T,VAL),readkey2(KEY,VAL).
  readkey1(cr,_,13):-!.
  readkey1(esc,_,27):-!.
  readkey1(break,_,3):-!.
  readkey1(tab,_,9):-!.
  readkey1(bdel,_,8):-!.
  readkey1(ctrlbdel,_,127):-!.
  readkey1(char(T),T,_) .

  readkey2(btab,15):-!.
  readkey2(del,83):-!.
  readkey2(ins,82):-!.
  readkey2(up,72):-!.
```

```
  readkey2(down,80):-!.
  readkey2(left,75):-!.
  readkey2(right,77):-!.
  readkey2(pgup,73):-!.
  readkey2(pgdn,81):-!.
  readkey2(end,79):-!.
  readkey2(home,71):-!.
  readkey2(ctrlleft,115):-!.
  readkey2(ctrlright,116):-!.
  readkey2(ctrlend,117):-!.
  readkey2(ctrlpgdn,118):-!.
  readkey2(ctrlhome,119):-!.
  readkey2(ctrlpgup,132):-!.
  readkey2(fkey(N),VAL):- VAL>58, VAL<70, N=VAL-58, !.
  readkey2(fkey(N),VAL):- VAL>=84, VAL<104, N=11+VAL-84, !.
  readkey2(otherspec,_).

/*****************************************************************

     Turbo Prolog Toolbox
     (C) Copyright 1987 Borland International.

  menu
  Implements a popup menu with at most 23 possible choices.
  For more than 23 possible choices use longmenu.

  The up and down arrow keys can be used to move the bar.
  RETURN or F10  will select an indicated item.
  Pressing Esc aborts menu selection and returns zero.

  The arguments to menu are:

  menu(ROW,COL,WINDOWATTR,FRAMEATTR,
       STRINGLIST,HEADER,STARTCHOICE,SELECTION)

ROW and COL determines the position of the window
WATTR and FATTR determine the attributes for the window
and its frame - if FATTR is zero there
will be no frame around the window.
STRINGLIST is the list of menu items
HEADER is the text to appear at the top of the menu window
STARTCHOICE determines where the bar should be placed.

  Ex:  menu(5,5,7,7,[this,is,a,test],"select word",0,CHOICE)

*****************************************************************/

PREDICATES
/*  menu(ROW,COL,ATTR,ATTR,STRINGLIST,STRING,INTEGER,INTEGER) */
  menuinit(ROW,COL,ATTR,ATTR,STRINGLIST,STRING,ROW,COL)
  menu1(SYMBOL,ROW,ATTR,STRINGLIST,ROW,COl,INTEGER)
  menu2(KEY,STRINGLIST,ROW,ROW,ROW,SYMBOL)

CLAUSES
  menu(ROW,COL,WATTR,FATTR,LIST,HEADER,STARTCHOICE,CHOICE) :-
menuinit(ROW,COL,WATTR,FATTR,LIST,HEADER,NOOFROW,LEN),
        ST1=STARTCHOICE-1,max(0,ST1,ST2),
        MAX=NOOFROW-1,min(ST2,MAX,STARTROW),
menu1(cont,STARTROW,WATTR,LIST,NOOFROW,LEN,CHOICE),
removewindow.
```

```
  menuinit(ROW,COL,WATTR,FATTR,LIST,HEADER,NOOFROW,NOOFCOL):-
maxlen(LIST,0,MAXNOOFCOL),
str_len(HEADER,HEADLEN),
HEADL1=HEADLEN+4,
max(HEADL1,MAXNOOFCOL,NOOFCOL),
listlen(LIST,LEN), LEN>0,
NOOFROW=LEN,
adjframe(FATTR,NOOFROW,NOOFCOL,HH1,HH2),
adjustwindow(ROW,COL,HH1,HH2,AROW,ACOL),
makewindow(81,WATTR,FATTR,HEADER,AROW,ACOL,HH1,HH2),
writelist(0,NOOFCOL,LIST).

  menu1(cont,ROW,ATTR,LIST,MAXROW,NOOFCOL,CHOICE):-!,
reverseattr(ATTR,REV),
field_attr(ROW,0,NOOFCOL,REV),
cursor(ROW,0),
readkey(KEY),
field_attr(ROW,0,NOOFCOL,ATTR),
menu2(KEY,LIST,MAXROW,ROW,NEXTROW,CONT),
menu1(CONT,NEXTROW,ATTR,LIST,MAXROW,NOOFCOL,CHOICE).
  menu1(esc,ROW,_,_,_,_,CHOICE):-!,CHOICE=ROW+1.
  menu1(_,ROW,ATTR,_,_,NOOFCOL,CHOICE):-
CHOICE=ROW+1,
reverseattr(ATTR,REV),
field_attr(ROW,0,NOOFCOL,REV).

  menu2(esc,_,_,_,-1,esc):-!.
  menu2(fkey(10),_,_,ROW,ROW,stop):-!.
  menu2(char(C),LIST,_,_,CH,selection):-tryletter(C,LIST,CH),!.
/*menu2(fkey(1),_,_,ROW,ROW,cont):-!,help.  If a help system is used */
  menu2(cr,_,_,ROW,CH,selection):-!,CH=ROW.
  menu2(up,_,_,ROW,NEXTROW,cont):-ROW>0,!,NEXTROW=ROW-1.
  menu2(down,_,MAXROW,ROW,NEXTROW,cont):-NEXTROW=ROW+1,NEXTROW<MAXROW,!.
  menu2(end,_,MAXROW,_,NEXT,cont):-!,NEXT=MAXROW-1.
  menu2(pgdn,_,MAXROW,_,NEXT,cont):-!,NEXT=MAXROW-1.
  menu2(home,_,_,_,0,cont):-!.
  menu2(pgup,_,_,_,0,cont):-!.
  menu2(_,_,_,ROW,ROW,cont).


/***************************************************************/
/*  menu_leave */
/* As menu but the window is not removed on return. */
/***************************************************************/

PREDICATES
  menu_leave(ROW,COL,ATTR,ATTR,STRINGLIST,STRING,INTEGER,INTEGER)

CLAUSES
  menu_leave(ROW,COL,WATTR,FATTR,LIST,HEADER,STARTCHOICE,CHOICE) :-
menuinit(ROW,COL,WATTR,FATTR,LIST,HEADER,NOOFROW,NOOFCOL),
        ST1=STARTCHOICE-1,max(0,ST1,ST2),
        MAX=NOOFROW-1,min(ST2,MAX,STARTROW),
menu1(cont,STARTROW,WATTR,LIST,NOOFROW,NOOFCOL,CHOICE).



/**************************************************************
```

```
  menu_mult
 Implements a popup-menu which allows a multiple number of
 selections.

 Each selection is made by pressing RETURN. All selections are
 then activated by pressing F10.

  The arguments to menu_mult are:

  menu(ROW,COL,WINDOWATTR,FRAMEATTR,STRINGLIST,HEADER,STARTLIST,NEWLIST)

ROW and COL determine the position of the window
WATTR and FATTR determine the attributes for the window
and its frame - if FATTR is zero there
will be no frame around the window.
STRINGLIST is the list of menu items
HEADER is the text to appear at the top of the menu window
STARTLIST determines the items to be highlighted when
  the menu is first displayed
NEWLIST   contains the list of selections

  Ex:  menu_mult(5,5,7,7,[this,is,a,test],"select words",[1],NEWLIST)

*************************************************************/

PREDICATES
  menu_mult(ROW,COL,ATTR,ATTR,STRINGLIST,STRING,INTEGERLIST,INTEGERLIST)
  multmenu1(SYMBOL,ROW,ATTR,STRINGLIST,ROW,COL,INTEGERLIST,INTEGERLIST)
  highlight_selected(INTEGERLIST,COL,ATTR)
  handle_selection(INTEGER,INTEGERLIST,INTEGERLIST,COL,ATTR)
  try_del(INTEGER,INTEGERLIST,INTEGERLIST,COL,ATTR)

CLAUSES
  menu_mult(ROW,COL,WATTR,FATTR,LIST,HEADER,STARTCHLIST,CHLIST) :-
menuinit(ROW,COL,WATTR,FATTR,LIST,HEADER,NOOFROW,NOOFCOL),
multmenu1(cont,0,WATTR,LIST,NOOFROW,NOOFCOL,STARTCHLIST,CHLIST),
removewindow.

  multmenu1(stop,_,_,_,_,_,CHL,CHL):-!.
  multmenu1(esc,_,_,_,_,_,_,[]):-!.
  multmenu1(selection,ROW,ATTR,LIST,MAXROW,NOOFCOL,OLDCHLIN,CHLOUT):-
CHOICE=1+ROW,
handle_selection(CHOICE,OLDCHLIN,NEWCHLIN,NOOFCOL,ATTR),
multmenu1(cont,ROW,ATTR,LIST,MAXROW,NOOFCOL,NEWCHLIN,CHLOUT).
  multmenu1(cont,ROW,ATTR,LIST,MAXROW,NOOFCOL,CHLIN,CHLOUT):-
reverseattr(ATTR,REV),
highlight_selected(CHLIN,NOOFCOL,REV),
cursor(ROW,0),
readkey(KEY),
menu2(KEY,LIST,MAXROW,ROW,NEXTROW,CONT),
multmenu1(CONT,NEXTROW,ATTR,LIST,MAXROW,NOOFCOL,CHLIN,CHLOUT).

  highlight_selected([],_,_).
  highlight_selected([H|T],L,ATTR):-
ROW=H-1,
field_attr(ROW,0,L,ATTR),
highlight_selected(T,L,ATTR).

  try_del(SELECTION,[SELECTION|REST],REST,LEN,ATTR):-
ROW=SELECTION-1,
```

```
field_attr(ROW,0,LEN,ATTR),!.
  try_del(SELECTION,[H|REST],[H|REST1],LEN,ATTR):-
try_del(SELECTION,REST,REST1,LEN,ATTR).

  handle_selection(SELECTION,OLDCHIN,NEWCHIN,LEN,ATTR):-
try_del(SELECTION,OLDCHIN,NEWCHIN,LEN,ATTR),!.
  handle_selection(SELECTION,OLDCHIN,[SELECTION|OLDCHIN],_,_).


 /*********************************************************************

     Turbo Prolog Toolbox
     (C) Copyright 1987 Borland International.

           Status Line
            Uses window number 83 for a status line.
 *********************************************************************/

PREDICATES
/*  makestatus(ATTR,STRING)  moved to global */
  nondeterm tempstatus(ATTR,STRING)
  removestatus
  refreshstatus
/*  changestatus(STRING)  moved to global */

CLAUSES
  makestatus(ATTR,TXT):-
shiftwindow(OLD),
makewindow(83,ATTR,0,"",24,0,1,80),
field_str(0,0,80,TXT),
shiftwindow(OLD).

  tempstatus(ATTR,TXT):-makestatus(ATTR,TXT).
  tempstatus(_,_):-removestatus,fail.

  removestatus:-
shiftwindow(OLD),
shiftwindow(83), removewindow,
shiftwindow(OLD).

  refreshstatus:-
shiftwindow(OLD),
shiftwindow(83),
shiftwindow(OLD).

  changestatus(TXT):-
shiftwindow(OLD),
shiftwindow(83),
field_str(0,0,80,TXT),
shiftwindow(OLD).

 /**************************************************/
 /*                                                */
 /*    Turbo Prolog Toolbox                        */
 /*    (C) Copyright 1987 Borland International.   */
 /*                                                */
 /*             LINE INPUT DRIVER                  */
 /**************************************************/

/* These two database predicates must be declared somewhere
```

```
in the program.
 */


PREDICATES
  lineinput(ROW,COL,LEN,ATTR,ATTR,STRING,STRING,STRING)
  lineinput_leave(ROW,COL,LEN,ATTR,ATTR,STRING,STRING,STRING)
  nondeterm lineinput_repeat(ROW,COL,LEN,ATTR,ATTR,
                             STRING,STRING,STRING)
  nondeterm lineinput_repeat1(ROW,COL,LEN,ATTR,ATTR,
                              STRING,STRING,STRING,STRING)
  lineinput1(ROW,COL,LEN,ATTR,ATTR,STRING,STRING,STRING,KEY)
  inpkeyact(KEY,COL,LEN,COL,COL,STRING,STRING)
  inpendkey(KEY)
  changemode
  inpstate(DBASEDOM)
  setlineinpflag(KEY)
  disp_str(COL,COL,LEN,STRING)
  lineinpcursor(COL,LEN,COL,COL,STRING)
  lin(KEY,COL,STRING,STRING)
  myfrontstr(COL,STRING,STRING,STRING)
  dropchar(STRING,STRING)

CLAUSES
/* Lineinput creates an input line at ROW and COL of LENGTH with a prompt
   CONSTTXT and an starting value OLDTXT that will be returned as NEWTXT
   if no new text is entered   */
  lineinput(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,NEWTXT):-
lineinput1(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,NEWTXT,KEY),
removewindow,     /* Removes window */
not(KEY=esc).

  lineinput_leave(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,NEWTXT):-
lineinput1(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,NEWTXT,KEY),
not(KEY=esc),!.
  lineinput_leave(_,_,_,_,_,_,_,_):-removewindow,fail.

/* A fail will cause backtracking to lineinp_repeat and a new lineinput
   will be displayed */
  lineinput_repeat(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,NEWTXT):-
lineinput1(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,NEWTXT1,KEY),
not(KEY=esc),!,
        lineinput_repeat1(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,
                          NEWTXT1,NEWTXT).
  lineinput_repeat(_,_,_,_,_,_,_,_):-removewindow,fail.

  lineinput_repeat1(_,_,_,_,_,_,_,TXT,TXT).
  lineinput_repeat1(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,_,NEWTXT):-
removewindow,
lineinput_repeat(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,NEWTXT).


  lineinput1(_,_,_,_,_,_,_,_,_):-retract(lineinpflag),fail.
  lineinput1(ROW,COL,LEN,WATTR,FATTR,CONSTTXT,OLDTXT,NEWTXT,KEY):-
MAXCOL=LEN,
adjframe(FATTR,1,MAXCOL,DR,DC),
adjustwindow(ROW,COL,DR,DC,ROW1,COL1),
makewindow(81,WATTR,FATTR,"",ROW1,COL1,DR,DC),
str_len(CONSTTXT,MINCOL),
write(CONSTTXT),
```

```
inpstate(lineinpstate(OLDTXT,0)),
REST=LEN-MINCOL,
repeat,
lineinpstate(CURTEXT,OFFSET),
disp_str(OFFSET,MINCOL,REST,CURTEXT),
cursor(_,CC),
        readkey(KEY),  /* In TPREDS.PRO returns the tokenized input */
setlineinpflag(KEY),
inpkeyact(KEY,CC,MAXCOL,MINCOL,OFFSET,CURTEXT,OLDTXT),
inpendkey(KEY),
lineinpstate(NEWTXT,_),!.

  inpendkey(fkey(10)):-!.
  inpendkey(cr):-!.
  inpendkey(esc).

/* assert lineinpflag when control character is entered */
  setlineinpflag(char(_)):-!.
  setlineinpflag(_):-lineinpflag,!.
  setlineinpflag(_):-assert(lineinpflag).

  disp_str(_,_,_,_):-keypressed,!.
  disp_str(0,MINCOL,LEN,TEXT):-!,
field_str(0,MINCOL,LEN,TEXT).
  disp_str(OFFSET,MINCOL,LEN,TEXT):-
frontstr(OFFSET,TEXT,_,TXT2),!,
field_str(0,MINCOL,LEN,TXT2).
  disp_str(_,MINCOL,LEN,_):-
field_str(0,MINCOL,LEN,"").

  inpstate(_):-retract(lineinpstate(_,_)),fail.
  inpstate(T):-assert(T).

  lineinpcursor(COL,MAXCOL,MINCOL,_,_):-
COL<MAXCOL,COL>=MINCOL,!,cursor(0,COL).
  lineinpcursor(COL,MAXCOL,_,OFFSET,TXT):-
COL>=MAXCOL,!,
OFFSET1=1+OFFSET+COL-MAXCOL,
inpstate(lineinpstate(TXT,OFFSET1)),
COL1=MAXCOL-1,
cursor(0,COL1).
  lineinpcursor(COL,_,MINCOL,OFFSET,TXT):-
COL<MINCOL,
OFFSET1=OFFSET-(1+MINCOL-COL),
max(0,OFFSET1,OFFSET2),
inpstate(lineinpstate(TXT,OFFSET2)),
cursor(0,MINCOL).

  myfrontstr(N,STR,S1,S2):-frontstr(N,STR,S1,S2),!.
  myfrontstr(N,STR,S1,""):-
str_len(STR,NN),
LEN=N-NN,
str_len(SS,LEN),
concat(STR,SS,S1).

  changemode:-retract(insmode),!.
  changemode:-assert(insmode).

  lin(char(T),POS,STR,STR1):-
insmode,!,
```

```
myfrontstr(POS,STR,S1,S2),
frontchar(S22,T,S2),
concat(S1,S22,STR1).

  lin(char(T),POS,STR,STR1):-
myfrontstr(POS,STR,S1,S2),
dropchar(S2,S21),
frontchar(S22,T,S21),
concat(S1,S22,STR1).

  lin(del,POS,STR,STR1):-
frontstr(POS,STR,S1,S2),
frontchar(S2,_,S22),!,
concat(S1,S22,STR1).
  lin(del,_,S,S).

  dropchar(S,S1):-frontchar(S,_,S1),!.
  dropchar(S,S).


 /*****************************************************************
 inpkeyact carries out associated action for user input. Handles all
 standard input keys: arrow keys, insert, delete, etc.
 *****************************************************************/

  inpkeyact(char(T),COL,MAXCOL,MINCOL,OFFSET,TXT,OLDTXT):-
lineinpflag,!,
POS=OFFSET+COL-MINCOL,
lin(char(T),POS,TXT,TXT1),
inpstate(lineinpstate(TXT1,OFFSET)),
inpkeyact(right,COL,MAXCOL,MINCOL,OFFSET,TXT1,OLDTXT).

  inpkeyact(char(T),_,MAXCOL,MINCOL,_,_,OLDTXT):-
assert(lineinpflag),
str_char(TXT1,T),
inpstate(lineinpstate(TXT1,0)),
inpkeyact(right,MINCOL,MAXCOL,MINCOL,0,TXT1,OLDTXT).

  inpkeyact(del,COL,_,MINCOL,OFFSET,TXT,_):-
POS=OFFSET+COL-MINCOL,
lin(del,POS,TXT,TXT1),
inpstate(lineinpstate(TXT1,OFFSET)).

  inpkeyact(bdel,COL,MAXCOL,MINCOL,OFFSET,TXT,OLDTXT):-
COL>MINCOL,
POS=OFFSET+COL-1-MINCOL,
lin(del,POS,TXT,TXT1),
inpstate(lineinpstate(TXT1,OFFSET)),
inpkeyact(left,COL,MAXCOL,MINCOL,OFFSET,TXT1,OLDTXT).

  inpkeyact(cr,_,_,_,_,_,_).

  inpkeyact(fkey(10),_,_,_,_,_,_).

/*inpkeyact(fkey(1),_,_,_,_,_,_):-help.  If a help system is used  */

  inpkeyact(esc,_,_,_,_,_,_).

  inpkeyact(ins,_,_,_,_,_,_):-changemode.
```

```
  inpkeyact(home,_,_,MINCOL,_,TXT,_):-
inpstate(lineinpstate(TXT,0)),
cursor(0,MINCOL).

  inpkeyact(end,_,MAXCOL,MINCOL,_,TXT,_):-
str_len(TXT,STRLEN),
COL=MINCOL+STRLEN,
COL<MAXCOL,!,
inpstate(lineinpstate(TXT,0)),
cursor(0,COL).

  inpkeyact(end,_,MAXCOL,MINCOL,_,TXT,_):-
str_len(TXT,STRLEN),
OFFSET=(MINCOL+STRLEN+1)-MAXCOL,
inpstate(lineinpstate(TXT,OFFSET)),
COL=MAXCOL-1,
cursor(0,COL).

  inpkeyact(right,COL,MAXCOL,MINCOL,OFFSET,TXT,_):-
COL1=COL+1,
lineinpcursor(COL1,MAXCOL,MINCOL,OFFSET,TXT).

  inpkeyact(ctrlright,COL,MAXCOL,MINCOL,OFFSET,TXT,_):-
COL1=COL+5,
lineinpcursor(COL1,MAXCOL,MINCOL,OFFSET,TXT).

  inpkeyact(left,COL,MAXCOL,MINCOL,OFFSET,TXT,_):-
COL1=COL-1,
lineinpcursor(COL1,MAXCOL,MINCOL,OFFSET,TXT).

  inpkeyact(ctrlleft,COL,MAXCOL,MINCOL,OFFSET,TXT,_):-
COL1=COL-5,
lineinpcursor(COL1,MAXCOL,MINCOL,OFFSET,TXT).

  inpkeyact(ctrlbdel,_,_,MINCOL,_,_,_):-
inpstate(lineinpstate("",0)),
cursor(0,MINCOL).

  inpkeyact(fkey(8),_,_,MINCOL,_,_,OLDTXT):-
OLDTXT><"",
inpstate(lineinpstate(OLDTXT,0)),
cursor(0,MINCOL).

/*****************************************************************

     Turbo Prolog Toolbox
     (C) Copyright 1987 Borland International.

SCRHND
======

 This module implements a screen handler called by:

                 scrhnd(TOPLINE,ENDKEY)

TOPLINE = on/off  - determines if there should be a top line
ENDKEY            - Esc or F10 used to return values
*****************************************************************/

PREDICATES
```

```
  /* SCREEN DRIVER */
/*   scrhnd(SYMBOL,KEY)   moved to global */
  endkey(KEY)
  scr(KEY)
  writescr
  showcursor
  mkheader
  showoverwrite

  ass_val(FNAME,STRING)
  valid(FNAME,TYPE,STRING)
  typeerror
  chng_actfield(FNAME)
/*  field_action(FNAME)
  field_value(FNAME,STRING)
  noinput(FNAME)     moved to global */
  types(INTEGER,TYPE,STRING) /* Definition of the known types */


 /*********************************************************************/
 /* Create the window */
 /* This can be used to create the window automatically from the */
 /* windowsize predicate. */
 /*********************************************************************/

PREDICATES
  createwindow(SYMBOL)

CLAUSES
  createwindow(off):-
windowsize(R,C),!,
R1=R+3, C1=C+3,
makewindow(81,23,66,"",0,0,R1,C1).
  createwindow(on):-
windowsize(R,C),!,
R1=R+3, C1=C+3,
makewindow(85,112,0,"",0,0,1,C1),
makewindow(81,23,66,"",1,0,R1,C1).

 /***************************************************************/
 /*            Intermediate predicates                          */
 /***************************************************************/

PREDICATES
  ttrunc(LEN,STRING,STRING)
  oldstr(FNAME,STRING)
  settopline(SYMBOL)

CLAUSES
  endkey(fkey(10)):-!.
  endkey(esc).

  ttrunc(LEN,STR1,STR2):-str_len(STR1,L1),L1>LEN,
                        !,frontstr(LEN,STR1,STR2,_).
  ttrunc(_,STR,STR).

  settopline(_):-retract(notopline),fail.
  settopline(off):-!,assert(notopline).
  settopline(_).
```

```
  oldstr(FNAME,S):- value(FNAME,S),!.
  oldstr(_,"").

  ass_val(FNAME,_):- retract(value(FNAME,_)),fail.
  ass_val(FNAME,VAL):-VAL><"",assert(value(FNAME,VAL)),fail.
  ass_val(_,_).

  chng_actfield(_):-typeerror,!,fail.
  chng_actfield(_):-
retract(actfield(_)),fail.
  chng_actfield(FNAME):-
assert(actfield(FNAME)).

  typeerror:-
actfield(FNAME),
field(FNAME,TYPE,_,_,_),
value(FNAME,VAL),
not(valid(FNAME,TYPE,VAL)),
beep,!.

  valid(_,str,_).
  valid(_,int,STR):-str_int(STR,_).
  valid(_,real,STR):-str_real(STR,_).

  /* The known types */
  types(1,int,"integer").
  types(2,real,"real").
  types(3,str,"string").


 /*****************************************************************/
 /*             SCREEN DRIVER                                     */
 /* Screen definition/input is repeated until F10 is pressed      */
 /*****************************************************************/

  scrhnd(STATUSON,KEY):-
settopline(STATUSON),
mkheader,
writescr,
field(FNAME,_,R,C,_),!,cursor(R,C),
chng_actfield(FNAME),
showcursor,
repeat,
writescr,
keypressed,/*Continuation until keypress means that time dependent
     user functions can be updated*/
readkey(KEY),
scr(KEY),
showcursor,
endkey(KEY),!.

 /*******************************************************************/
 /*   Find the next field     */
 /*******************************************************************/

PREDICATES
  /* The predicates should be called with:
    ACTROW, ACTCOL, MAXROW, MAXCOL, NEWROW, NEWCOL   */
  best_right(ROW,COL,ROW,COL,ROW,COL)
  best_left(ROW,COL,ROW,COL,ROW,COL)
```

```
  best_down(ROW,COL,ROW,COL,LEN,ROW,COL)
  best_up(ROW,COL,ROW,COL,LEN,ROW,COL)
  better_right(ROW,COL,ROW,COL,ROW,COL)
  better_left(ROW,COL,ROW,COL,ROW,COL)
  better_field(ROW,COL,ROW,COL,LEN,ROW,COL,LEN)
  calcdist(ROW,COL,ROW,COL,LEN,LEN)
  move_left
  move_right
  nextfield(ROW,COL)
  gtfield(ROW,ROW,COL,COL)
  prevfield(ROW,COL)
  chk_found(FNAME,ROW,COL,ROW,COL)
  setlastfield

CLAUSES
  best_right(R0,C0,R1,C1,ROW,COL):-
field(_,_,R2,C2,_), C2>C0,
better_right(R0,C0,R1,C1,R2,C2),!,
best_right(R0,C0,R2,C2,ROW,COL).
  best_right(_,_,R,C,R,C).

  better_right(R0,_,R1,_,R2,_):-abs(R2-R0)<abs(R1-R0),!.
  better_right(R0,_,R1,C1,R2,C2):-abs(R2-R0)=abs(R1-R0),C2<C1.

  best_left(R0,C0,R1,C1,ROW,COL):-
field(_,_,R2,C2,_), C2<C0,
better_left(R0,C0,R1,C1,R2,C2),!,
best_left(R0,C0,R2,C2,ROW,COL).
  best_left(_,_,R,C,R,C).

  better_left(R0,_,R1,_,R2,_):-abs(R2-R0)<abs(R1-R0),!.
  better_left(R0,_,R1,C1,R2,C2):-abs(R2-R0)=abs(R1-R0),C2>C1.

  best_down(R0,C0,R1,C1,L1,ROW,COL):-
field(_,_,R2,C2,L2), R2>R0,
better_field(R0,C0,R1,C1,L1,R2,C2,L2),!,
best_down(R0,C0,R2,C2,L2,ROW,COL).
  best_down(_,_,R,C,_,R,C).

  best_up(R0,C0,R1,C1,L1,ROW,COL):-
field(_,_,R2,C2,L2), R2<R0,
better_field(R0,C0,R1,C1,L1,R2,C2,L2),!,
best_up(R0,C0,R2,C2,L2,ROW,COL).
  best_up(_,_,R,C,_,R,C).

  better_field(R0,C0,R1,C1,L1,R2,C2,L2):-
calcdist(R0,C0,R1,C1,L1,DIST1),
calcdist(R0,C0,R2,C2,L2,DIST2),
DIST2<DIST1.

  calcdist(R0,C0,R1,C1,L1,DIST):-
C11=C1+L1,
max(C0,C1,H1),
min(H1,C11,H2),
DIST=3*abs(R1-R0)+abs(H2-C0).

  move_left:-
not(typeerror),
actfield(FNAME),
field(FNAME,_,R,C,_),!,
```

```
best_left(R,C,-100,-100,ROW,COL),
field(F1,_,ROW,COL,_),
chng_actfield(F1),!,
cursor(ROW,COL).

  move_right:-
not(typeerror),
actfield(FNAME),
field(FNAME,_,R,C,_),!,
best_right(R,C,-100,-100,ROW,COL),
field(F1,_,ROW,COL,_),
chng_actfield(F1),!,
cursor(ROW,COL).

  prevfield(_,_):-typeerror,!,fail.
  prevfield(R,C):-
field(FNAME,_,ROW,COL,_),
chk_found(FNAME,R,C,ROW,COL),!,
actfield(F1),
field(F1,_,RR,CC,_),!,
cursor(RR,CC).

  chk_found(_,R,C,R,C):-!.
  chk_found(FNAME,_,_,_,_):-chng_actfield(FNAME),fail.


  nextfield(_,_):-typeerror,!,fail.
  nextfield(R,C):-
field(FNAME,_,ROW,COL,_),gtfield(ROW,R,COL,C),
chng_actfield(FNAME),!,
cursor(ROW,COL).
  nextfield(_,_).

  gtfield(R1,R2,_,_):-R1>R2,!.
  gtfield(R,R,C1,C2):-C1>C2.

  setlastfield:-
field(FNAME,_,_,_,_),
chng_actfield(FNAME),
fail.
  setlastfield.


 /****************************************************************/
 /*            scr                                               */
 /****************************************************************/

  /* Insert a new character in a field */
  scr(char(T)):-actfield(FNAME),
not(noinput(FNAME)),
cursor(_,C),
field(FNAME,_,ROW,COL,LEN),!,
POS=C-COL,
oldstr(FNAME,STR),
lin(char(T),POS,STR,STR1),
ttrunc(LEN,STR1,STR2),
ass_val(FNAME,STR2),
field_str(ROW,COL,LEN,STR2),
scr(right).
```

```
  /* Delete character under cursor */
  scr(del):- actfield(FNAME),
not(noinput(FNAME)),
cursor(_,C),
field(FNAME,_,ROW,COL,LEN),!,
POS=C-COL,
oldstr(FNAME,STR),
lin(del,POS,STR,STR1),
ass_val(FNAME,STR1),
field_str(ROW,COL,LEN,STR1).

  /* Delete character before cursor and move cursor to the left */
  scr(bdel):- actfield(FNAME),
not(noinput(FNAME)),
cursor(_,C),
field(FNAME,_,ROW,COL,LEN),!,
POS=C-COL-1,
oldstr(FNAME,STR),
lin(del,POS,STR,STR1),
ass_val(FNAME,STR1),
field_str(ROW,COL,LEN,STR1),
scr(left).

 /*If there is an action - do it. Otherwise, go to next field*/
  scr(cr):-
actfield(FNAME),
field_action(FNAME),
cursor(RR,CC),cursor(RR,CC),!.
  scr(cr):-cursor(RR,CC),cursor(RR,CC),scr(tab).


  /* Change between insertmode and overwritemode */
  scr(ins):-changemode,showoverwrite.

  /* escape */
  scr( esc ).

  /* F10: end of definition */
  scr( fkey(10) ):-not(typeerror).

  scr(right):-
actfield(FNAME),
not(noinput(FNAME)),
field(FNAME,_,_,C,L),
cursor(ROW,COL), COL<C+L-1,!,
COL1=COL+1,
cursor(ROW,COL1).

  scr(right):-move_right.

  scr(ctrlright):-
actfield(FNAME),
not(noinput(FNAME)),
field(FNAME,_,_,C,L),
cursor(ROW,COL),
COL1=COL+5, COL1<C+L-1,!,
cursor(ROW,COL1).

  scr(ctrlright):-move_right.
```

```
  scr(left):-
actfield(FNAME), field(FNAME,_,_,C,_),
cursor(ROW,COL),
COL>C,!,
COL1=COL-1,
cursor(ROW,COL1).

  scr(left):-move_left.

  scr(ctrlleft):-
actfield(FNAME), field(FNAME,_,_,C,_),
cursor(ROW,COL),
COL1=COL-5, COL1>C,!,
cursor(ROW,COL1).

  scr(ctrlleft):-move_left.

  scr(tab):-
cursor(R,C),
nextfield(R,C).

  scr(btab):-
cursor(R,C),
prevfield(R,C).

  scr(up):-
not(typeerror),
cursor(R,C),
best_up(R,C,-100,-100,1,ROW,COL),
field(F1,_,ROW,COL,_),
chng_actfield(F1),!,
cursor(ROW,COL).

  scr(down):-
not(typeerror),
cursor(R,C),
best_down(R,C,100,100,1,ROW,COL),
field(F1,_,ROW,COL,_),
chng_actfield(F1),!,
cursor(ROW,COL).

  scr(home):-
not(typeerror),
field(F1,_,ROW,COL,_),
chng_actfield(F1),!,
cursor(ROW,COL).

  scr(end):-
not(typeerror),
setlastfield,
actfield(FNAME),
field(FNAME,_,ROW,COL,_),!,
cursor(ROW,COL).

/* scr(fkey(1)):-help.  If helpsystem is used. */


 /***********************************************************/
 /*     Predicates maintaining the top messages line        */
```

```
/*************************************************************/

  mkheader:-notopline,!.
  mkheader:-
   shiftwindow(OLD),
gotowindow(85),
field_str(0,0,30,"ROW:      COL:"),
gotowindow(OLD).

PREDICATES
  get_overwritestatus(STRING)
  show_str(COL,LEN,STRING)
  showfield(ROW,COL)

CLAUSES
  get_overwritestatus(insert):-insmode,!.
  get_overwritestatus(overwrite).

  show_str(C,L,STR):-
windowsize(_,COLS),
C<COLS,!,
MAXL=COLS-C,
min(L,MAXL,LL),
field_str(0,C,LL,STR).
  show_str(_,_,_).

  showoverwrite:-notopline,!.
  showoverwrite:-
shiftwindow(OLD),
gotowindow(85),
get_overwritestatus(OV),
show_str(20,9,OV),
gotowindow(OLD).

  showfield(_,_):-keypressed,!.
  showfield(R,C):-
field(FNAME,TYP,ROW,COL,LEN),
ROW=R, COL<=C, C<COL+LEN,
types(_,TYP,TYPE),!,
show_str(30,8,TYPE),
STR=FNAME, show_str(38,42,STR).
  showfield(_,_):-keypressed,!.
  showfield(R,C):-
txtfield(ROW,COL,LEN,TXT),
ROW=R, COL<=C, C<=COL+LEN,!,
show_str(30,1,"\""),
show_str(31,49,TXT).
  showfield(_,_):-show_str(30,50,"").

  showcursor:-keypressed,!.
  showcursor:-notopline,!.
  showcursor:-
shiftwindow(OLD),
cursor(R,C),
str_int(RSTR,R), str_int(CSTR,C),
gotowindow(85),
show_str(4,4,RSTR), show_str(14,4,CSTR),
showfield(R,C),
gotowindow(OLD),
cursor(R,C).
```

```
/**************************************************************/
/*     update all fields on the screen                        */
/**************************************************************/

  writescr:-
field(FNAME,_,ROW,COL,LEN),
field_attr(ROW,COL,LEN,112),
field_value(FNAME,STR),
field_str(ROW,COL,LEN,STR),
keypressed,!.
  writescr:-
txtfield(ROW,COL,LEN,STR),
field_str(ROW,COL,LEN,STR),
keypressed,!.
  writescr.
```

$\boxed{\textbf{main}}$

```
project "tez"

include "global.pro"

goal main.
```