EXPERT SYSTEM SHELL FOR ARCHITECTURAL AND BUILDING DESIGN

A DISSERTATION

submitted in partial fulfilment of the
requirements for the award of the degree
of
MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE AND TECHNOLOGY

By

TEJPAL SINGH

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE-247 667 (INDIA)

JANUARY, 1995

CANDIDATES DECLARATION

I hereby certify that the work which is being presented in the dissertation entitled **"EXPERT SYSTEM SHELL FOR ARCHITECTURAL AND BUILDING DESIGN"** *in the partial fulfillment of the degree of MASTER OF TECHNOLOGY in COMPUTER SCIENCE & TECHNOLOGY in the Department of* ELECTRONICS & COMPUTER ENGINEERING, University Of Roorkee, *is an authentic record of my own work carried out,* 1495 *under the guidance of* **Dr. PADAM KUMAR**, *Department of* ELECTRONICS & COMPUTER ENGINEERING, University Of Roorkee. *The matter embodied in this dissertation has not been submitted by me for the award of any other degree or diploma.*

Date : 17th January, 1995

Place : ROORKEE

[TEJPAL SINGH]

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date : 17th January, 1995

Place : ROORKEE

[Dr. PADAM KUMAR]

Department Of Electronics & Computer Engg.

ACKNOWLEDGEMENT

It is my proud privilege to express a profound sense of gratitude to my guide **Dr. PADAM KUMAR** *for his liberal guidance, inspiration and encouragement which I have received on my hands so spontaneously and lavishly throughout my dissertation.*

I wish to thank **Dr. R.C. JOSHI** *especially for providing all the facilities (from Systems, Softwares to Concepts), without which the completion of the work would have been impossible.*

I would like to thank Head of the Deptt. **Dr. R.P. Aggarwal,** *for providing facilities in the deptt., helpful in completion of the project.*

I would also link to thank all the staff members of the Dept. for their kind co-operation.

My friends in the hostel and outside, have proved what friendship is. When I look back on this period of my life, I shall not be able to forget the support my family provided me.

[**TEJPAL SINGH**]

CONTENTS

ABSTRACT

Design is a process of producing a description of a system or process to satisfy a set of requirement. To support the designer in the identification and composition of components of a design solutions requires both synthesis and evaluation methods. Such methods can provide a systematic approach to design allowing the designer to pursue more alternatives and evaluate the alternatives based on a discourse of criteria and value. The use of knowledge based techniques for the exploration of synthesis and evaluation methods maintains a separation of method and knowledge, allowing the designer to guide the methods with qualitative or empirical knowledge without sacrificing the benefits of a systematic approach.

In this work an expert system shell (ESS) named SHELL has been developed which will help the user in taking decisions and provide him an expert advice based on the facts provided by the user during the interaction. The ESS have a rule based inference engine (IE) which provides decisions based on the rules applicable for the session, which in turn depend upon the facts that hold for the project. The knowledge representation (KR) in the ESS is also with the semantic networks which basically form a fact base for the inference engine module. However it can be used independently to provide answers to the queries which can be answered with the help of a semantic network exploiting the inheritance properties of the object present in it. Two ESS modules were developed sharing a common semantic network. The modules are rule based with a difference in their learning abilities. One is a non-learning module which requires the complete set of rules to provided beforehand to get complete decisions or to cover all sorts of situations. This module assumes that all the knowledge in the form or rule base or semantic net is complete and correct. The following cases are envisaged in this module :

The system cannot reach a decision because the facts provided to it were conflicting ;

The system cannot provide a decision because the rule base has no rule(s) to reach that decision ;

The system provides conflicting decisions because the facts provided to it were not coherent ;

The system is able to provide the decisions assuming that rules provided to it are complete and correct and so are the facts provided. No confirmation is taken from the user i.e. the system is closed for any feedback.

In all these cases the system never tries to learn any new knowledge because it was never made to do so. A new situation has to handled explicitly by adding new rules in it's rule base.

The other module capable of acquiring, automatically & by explanation, the know how of an Architect to help him in the design process. The module using an incomplete description, provided by the designer, of the construction project to be designed, the system tries to complete the project using the knowledge at it's disposal which the designer has taught it. The following cases are envisaged :

The system cannot solve a problem because it has never learnt how to ;

The system cannot solve a problem because it lacks information ;

The system claims to know how to solve a problem and submits a solution to the designer for approval.

In all three cases the system learns the knowledge which it will be able to use at a future date. The learning mechanism resembles the learning methodology of a human being who also learns by experience.

The two modules share the semantic net to provide the various facts. The learning module may also be used to generate more facts for the non-learning module or may be used to complete a partial semantic, to be used by both of the modules at a future date and providing a means of interaction between the two modules.

SHELL is developed not with the aim of replacing an architect or the designer, but with the aim of providing him a tool capable of

simulating the work of an architect or helper who he has trained himself by providing him with rules and knowledge for handling situations. The model of the system developed is built around the High Level Language (HLL) 'C' with an automatic interpretation, knowledge acquisition & explanation system.

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Computers are rapidly penetrating almost all contemporary modern and sophisticated professions. In some fields their influence has been readily acceptable, but in others a restructuring of disciplines seems unavoidable.

Since computer technology continues to advance rapidly due to ever-growing & ever-absorbing consumer market, it is difficult to start a relevant discussion based on definite and proven assertions. Nevertheless, one dimension of such discussion is their potential to simulate the human mind and imagination to the opening of new phases.

In the field of Architecture, computers have been used to produce drawings and other graphical representations to communicate ideas using packages like Autocad, etc.. Further use of computers in this field has proven to be more than a mere drafting assistance, due to which the field of Architecture is undergoing a drastic change as a discipline.

Architecture is concerned with concept, ideas, judgment & experience. All of these appear to be outside the realm of traditional computing. Human beings discourse with each other using models of their worlds largely unrelated to either mathematical descriptions or procedural representations. They make use of knowledge about objects, events & processes and make declarative statements about them. These are often written down symbolically. The limits of traditional computing are that it is unable to represent and manipulate knowledge in an explicit and coherent form and that it is unable to perform symbolic computation. The knowledge based systems (KBS), however are trying to overcome these limitations and promise to be powerful

tools in this field.

There are considerable arguments in favor of developing KBSs for this field. The first and probably the major impetus in developing KBS is the potential for increasing productivity and improving the quality of design if low-level design decisions and procedures can be automated by encoding the knowledge on which they are based, since more time can now be allocated for more complex decisions. Other reasons, though may be just as significant. The great majority of failures in building construction are due to non-application of existing, recorded, knowledge, and that there is much to be gained from making that knowledge available at the point where it is most valuable ; that is within the medium of production of building construction information.

There are two ways to implement a KBS. Firstly, creating a computer program which has the required knowledge implemented in the form of its code. Secondly creating an expert system (ES) or ESS which store knowledge separately. Conventional computer approaches from the view point of complex applications like design suffer from serious drawbacks. Firstly they are usually complex and difficult for anyone other than their designers to understand. Secondly as they embody their knowledge of the subject area in terms designed for computational efficiency, such that this knowledge is implicit in the programs in such a way which makes it difficult to alter or change. Thirdly, they cannot suggest their users why they need a particular fact nor justify their results, a very important factor for opting for an expert system in field like Design to boost confidence of the potential users in the decisions provided by them.

1.2 STATEMENT OF THE PROBLEM

The ES are capable of managing the coherence of the project more satisfactorily, of conversing with the Architect in a language

closer to his own, and of establishing relations with the technical calculation programs or engineering ESs which thus allow a technical evaluation of the project to be made in very early stages of design. But careful not to limit the creative capacities of the Architects the automated procedures installed in these systems can only incorporate knowledge relating to general Architecture and Building know how, useful to any designer for any building project. In fact, there is no optimal solution, unique to a building design problem, an Architect who solves a problem manipulates strong constraints (regulatory, technical, etc...) but also weak constraints of more subjective character (economical, sociological, aesthetic, etc.) related to working habits and to influence of similar problems met in the past. That is why it seemed important to create an ESS which incorporates general Architecture and Building know how and which may also be personalized by the user according to the type of project to be carried out, his know how and his working habits. Also at present, an Architect from say Roorkee may not be able to design foundation of a building in Delhi due to lack of soil information of the site. Similarly he may not be able to decide upon the type of superstructure of the house in any other country just due to lack of information about sociological and environmental constraints. The aim was to gather all the information at one place and allowing an Architect from place X in country Y to be able to design a house in city A of country B without any problem. Moreover as he at one place is able to perform the job of an Architect, Builder, Structural engineer, etc. this will reduce the time of communication of the facts from one person to other, thus speeding up the initial design process leading to increased productivity as explained earlier with no compromise on quality. As the ESS SHELL is designed to highly interactive, there is no compromise on creative capability of the Architect, nor any rigidity in changing the design due to requirements of or

constraints laid by the client based on his sociological, economical or aesthetic needs.

An ESS design having KR in the form of both semantic networks and production rules was chosen to form the Knowledge base (KB). The rule based model was chosen as it closely resembles the thinking mechanisms of human beings. The Architect or Building designer would not only have much convenience in using such a system., but also will have full confidence in the decisions provided by the ESS. This is a very important factor in deciding whether the ESS will be used extensively by him as otherwise the whole purpose of developing such a system gets defeated. HLL 'C' was chosen as a tool to implement the expert system shell because of the language's ability to manipulate complex structures (used extensively in these kind of work), to provide customized interfacing to the external programs called by the ESS and lastly because programs written in this language have relatively smaller executable file size, saving on RAM usage and providing faster execution and loading. Keeping in view the applications which were to built around SHELL, need for incorporating various means of representing rules (like mathematical, probabilistic, external program & objects from semantic nets) was felt, so that much can done by the SHELL itself without the need to call external programs for these purposes.

A need for learning ESS was felt to provide each Architect with customized system. This system targeted as an assistant of the architect, was expected to learn to handle new situations with experiences, so that the Architect can train it by his own concept, means and ease. This system would also use the semantic net to reach decisions. This system was expected to update the semantic net through its decisions thus also providing one with a means to complete a partial semantic net intelligently.

1.3 ORGANIZATION OF DISSERTATION

The dissertation is divided into seven chapters. **Chapter 1** provided a basic introduction to the situation prevalent and introduced the problem to be handled. The rest of the dissertation is organized as follows. **Chapter 2** provides a review of the various ES and ESS developed in this direction and introduces various topics related to and incorporated in the dissertation. **Chapter 3** provides the description of the ESS at the user's level like various facilities incorporated in the work and provides information about the characteristic features of the ESS finally developed. **Chapter 4** & **chapter 5** provide sample extensive examples & rules and test run result from the fields of Architecture and Building design. **Chapter 6** discusses the software implementation of the ESS. Finally in **chapter 7** conclusions and various directions for future extensions on this work are discussed.

TABLE-2.1A

NAME	K.R. SCHEME	INTERFERENCING MECHANISM	LANGUAGE	MACHINES	EXPLANATION FACILITY	EDITOR/USER INTERFACE	GRAPHICS	[illegible]
ART, Automated Reasoning Tool	Hybrid "Tool Kit" with frames, goals and production rules	forward & backward chaining, pattern-matching & logic programming	LISP C	Symbolics, Suns VAX, LISP Machine, IBM-PC/RT, TI Explorer	Excellent graphics Browser	Excellent Graphics icon editor	Excellent [illegible]	[illegible]
KEE Knowledge Engineering Environment	Object-oriented frames with multiple inheritance, production rules	forward & backward chaining, provides a Truth Maintenance System.	LISP	Xerox, Lambda, Suns, Symbolics, IBM-RT, TI EXPLORER, VAX, HP	Excellent Graphical & highly interactive	Strong for rules, representation and interface	Excellent-object oriented oriented	[illegible]
KES II Knowledge Engineering System	Production rules, Baysian hypothesize & test	abductive reasoning	LISP-> C	Cyber 180, Xerox, VAX, Apollo, Sun, IBM-PC	Explanation, help, justify,& "WHY"	Line-oriented	Windows & Menus or dialog	[illegible]
KNOWLEDGE CRAFT	Frame based with inheritance & procedural attachment Integrated production system	rule - based forward chaining, & logic programming in Prolog	COMMON LISP CRL+/OPS5/ Prolog	VAX, TI Explorer, Symbolics, Suns HP 9320, Apollo	Graphics oriented trace facility	Knowledge base, frame & rule editors, OPS & Prolog workbench	Excellent Windows oriented	
S.1	Rule-based, frame-based, procedure oriented	Procedural programming is done by control strategies	LISP. LISP -> C C -> Ada	AT & T, Micro VAX, Sun, Apollo, VAX, Xerox,	English text help and Explanation	Knowledge base editor	Mouse and menu driven	[illegible]
HOC	Logic programming in a LISP environment	-	NISP, Franz LISP, Common LISP.	Lambda, VAXs TI Explorer, Xerox, Apollo, Symbolics	First order predicate calculus	Varies among LISP environments	Through LISP environment	[illegible]
LLOOS	Rule, access, object & procedure oriented programming.	-	INTERLISP-D	Xerox 1100 Series	Not evident, Since this is a research tool.	Excellent Graphics oriented	Original Xerox Star inspired Windows.	[illegible]
OPS5 AND OPS83	Rule Based, OPS 83 adds imperative programming	-	C and Exper, LISP	Macintosh, Apollo, IBM-PC, VAX, Micro VAX, Xerox	Must program More a language than a shell	Varies among several dialects	Uses mouse, Windows & Menu	[illegible]

M.1	Production Rule system	-	Prolog -> C	IBM-PC	Custom text (HOW and WHY)	May use regular word	Windows for tracing and	Run-time Version
Personal Consultant Series	Production Rule with frames, meta-rules and mapping	-	PC scheme	TI Business-Pro, IBM-PC/AT	"HOW","WHY" and Review	Pop-up menus with integrated graphics	Incorporates captured pictures	Runtime Copy
Super-Expert	Infers decision tree rules from examples with support ical structure	-	Forth for PC version Pascal for Mac version	IBM-PC, Macintosh	Help available Shows logic tables used	Standard Spread-sheet, read ahead	Spreadsheet Screens	Is own delivery system
ESP Advisor	Logic-based KRL Knowledge Representation Language	-	Prolog-2 Knowledge base in KRL	IBM-PC/XT/AT Data General/ One VAX, MicroVAX II	User interacts through scripts plus Help	Menu-driven screen, "text animation"	LCD Screenor good use of colour graphics	Consultation shell plus compiled knowledge base
EXSYS	IF-THEN-ELSE, Production Rules, math capabilities	-	C (Can link to data programs)	IBM-PC, VAX NEC 9801 Macintosh	User Controlled suntime report generator	Menu-driven full screen text editor	Colour-coded text	Run-time sun-time license
TIMM	Frame-like, multi dimensional state space rules inferred from user examples	-	FORTRAN	IBM-PC, Prime, VAX 11/780, Zenith, IBM Mainframes	Identifies all rules used in consultation	Query based dialog to build rules, system checks consistency	Menu-driven text screen only	Runtime Module
GURU	Production rules with natural language interface	-	C	IBM-PC, VAX11/780, IBM Mainframes	Recites rules on rules upon "WHY" or "HOW" request	Windows oriented NL interface	Menu driven windows	Run-time version
Extran	Hierarchies of decision-tree rules	-	Fortran	IBM MVS/TSO Vax VMS and Ultrix Unix workstations	Responds to "WHY" and "HOW" queries	Built-in interactive editor	Interface to Suntools windows	Run-time "Driver"
CLIPS C LANGUAGE Integrated Production System	Rules written in Lisp-Like syntax (if-then-else, while) (Disjunction and procedural programming)	Forward chaining control strategy	C	IBM-PC, Macintosh-II, Sun, Apollo, H-P, DEC 3100 RISC M/CB, DEC VAXs	Capabilities like cross-referencing of relations, style-checking, and semantic error checking	Graphical development interface		Same as machines
AKT-1M Automated Reasoning Tool for Information Management	Frames as Schemata	Forward chaining	C	IBM Mainframes, IBM-PC ATs, IBM Ps/2s, and DEC VAXs	-	Built -in editor, dialogue boxes, debug Menu		Same as machines
Level 5	Facts & Rules expressed in Level 5 Production Rule language (PRL)	Backward chaining control strategy	Pascal	IBM PCs, Machintoses, DEC VAXs, & IBM mainframes	-	-		Same as machines
Kappa-PC	Physical entities or abstract concepts are grouped into a hierarchical structure of classes, subclasses, and	Inference mechanisms available in object-oriented and rule-based programming Rules:-Forward & Backward	C	286 or 386-PC with a 640-K byte RAM under MS-DOS with MS-Win-with MS-Windows 286/386	-	Main Window, Object Browser, Knowledge Tools windows.	Directly accessing CAD Packages	Same as machines

TABLE-2.1B

NAME	PROBABILITY	HAND CRAFT	USER SOPHISTICATION	REFERENCES
ART	User-defined certainty factors	Very flexible tool-kit for Customizing	Powerful system integrating frames, rules, & goals Development tool requires experieced AI Programmers.	[17]
KEE	User -defined certainty factors	Flexible environment.	A flexible and powerful environment to assist experienced programmers in constructing expert systems	[17]
KES II	Certainty factors, Symbolic Probability	May be embedded in other program	Knowledge engineer may select one of three KR schemes for building user-friendly expert systems.	[17]
KNOWLEDGE CRAFT		Yes, with OPS5 & Prolog integration	Offers experienced programmers a choice of control strategies & knowledge representation techniques.	[17]
S.1	Certainty factors	Through links to UNIX System languages.	Shell for computer professionals to develop and deliver efficient expert systems for diagnosis, design & planning.	[17]
DUCK	Must Program	Through LISP environment	Experienced AI Programmers have four modes: logic, rule-based, non-monotonic reasoning, & deduction.	[17]
LOOPS	Not evident, Since this is a research tool.	Very flexible tool	Powerful Knowledge engineering language to assist experienced AI Programmers in developing systems.	[17]
OPS5 & OPS83	Must program in	General KR & Ctrl Structure	Widely used Knowledge engineering providing experienced programmers a good environment for buildings ES.	[17]

M.1	Certainty factors (-100-100)	Restricted, rule-based tool	Knowledge-based rules written in English-like language helpless experienced programmers write ES.	[17]
Personal Consultant Series	Certainty factors	Can link to external programs	Less experienced programmers can use this EMYCIN-like shell for writing ES upto 2000 rules.	[17]
Super-Expert	Not indicated	Rigid spread-sheet format	Domain experts & less experienced AI programmers may enter examples from which system infers rules.	[17]
ESP Advisor	Must program in Prolog	Very good through Prolog Link	Power & flexibility of KRL, requires some programming experience, easy for small scale systems	[17]
EXSYS	Several types including threshold	Fairly rigid rule format	Very easy to use with excellent tutorials, user interface & 5000 rule capacity.	[17]
TDM	Certainty values reliability (o-100)	Fairly rigid example format	Easy to use shell with good dialog to guide domain experts in linking system into decision networks (embeddable).	[17]
GURU	Variety of Certainty factors	System development tool	Powerful ES shell integrated with database, word processing etc. Complexity requires experts only.	[17]
Extran	ITL-Knowledge Link 36 George House Glasgow G12AB, UK (44-41-552-1353)	Standard rule format	Good development and operation environment, but requires knowledge engineer to program.	[17]
CLIPS	High Probability	Can link to external programs.	Development interfaces that support pull-down menus, an integrated editor, and multiple windows.	[19]
ART-IM		Integration with external programs.	Overlapping Windows & Pull-down menus. Reference manual contains some tutorial information & examples helpful to new users.	[19]
Level 5	Confidence factors range between (o & 100)	Fairly rigid rule format	Self-study guide is written as a tutorial to help new users.	[19]
Kappa-PC	Must program in C or KAL	Ways to access external data files & integrated external programs	Two manuals 1) User guide 2) Reference manuals	[18]

CHAPTER 2

REVIEW AND GENERAL CONSIDERATION

2.1 INTRODUCTION

The earliest attempts to use computers to aid architectural and building design commenced nearly three decades ago. However, unlike areas like medicine or geology, design does not have any theoretical core that can be reduced to a fixed set of mathematical and hence traditional computing were of no help. The result has been that Computer Aided Architectural Design (CAAD) has received little attention in the past. Computers when compliment the performance of human designer become a particularly useful design aid. They provide capabilities that are not readily available to designers who rely on traditional manual methods. For instance, the ability to systematically enumerate alternative solutions to a design problem or the ability to take, at the same time, a broad spectrum of design criteria or concerns into account. The underlying assumption is that the human cognitive apparatus does not perform particularly well in such tasks [1]. Knowledge Based Expert systems offer the possibility of capturing and manipulating the human knowledge, such as heuristics or rules of thumb accumulated by the expert or a consultant after years of problem solving.

2.2 REVIEW

A tabulated summary of the detailed features of a number of commercially available ES and ESS is given in Table 2.1. The list is not exhaustive but merely suggestive of the kind of tools. available. Besides, it is important to realize that they are dynamic in nature with capabilities continually expanding in response to customers needs and competitive pressures.

The study of various ESS revealed the following points :

(i) A trend of moving from mainframes to personal computers (PCs).

(ii) A trend of using languages like 'C' as compared to LISP or PROLOG for building these tools. The primary motivation being improved performance and portability.

(iii) A trend towards laying emphasis on embedability was apparent. The ESSs as embedded in user application far exceeded their potential as stand alone programs. This was achieved by the use of conventional programming languages as against typically Artificial Intelligence (AI) languages.

A survey of the work in the field of ESS and various ES tools for CAAD over the past decades is given below :

The two main features of the ESS are its KR technique and its Inferencing mechanism or the IE. The emphasis during the study was therefore, laid on these two points only.

Bedard and Gowri [2] examined the characteristics of engineering design, and in particular, the building design process in an attempt to identify the most important features to implement in a realistic computer-based design system. The authors proposed a prototype system as applied to the preliminary design of building envelope systems. The KR technique used was frames and inferencing was done by implementing a constrained-based depth first strategy. KnowledgeCraft was used as a programming environment.

Pohl and Chapman [3] reported a conceptual design of an Intelligent Computer-Aided Design System (ICADS) as an integration of an intelligent CAD database management system (DBMS), with a knowledge based design generator and a multimedia presentation facility for architectural design. The expert design generator stored the required knowledge as rules.

Fenves et.al. [4] proposed an integrated environment that had

Table 2.2

EXPERT SYSTEM	SYSTEM INPUT	SYSTEM OUTPUT	KNOWLEDGE STRUCTURE	TOOL
HICOST	Preliminary design alternatives	Cost estimate based on input	-	LISP C
BERT	Design of brickwork	Best design solution	-	AutoCAD
ELSIE	Client's project scope needs, aesthetics & design flexibility	Initial budget, profitability of project	-	Saviour shell
SITEPLAN	Available space	Site Layout Updation of site plan	-	KEE
CONSTRUC -TION PLANEX	Project data	Project duration, cost estimates & schedules	Frames	Know-ledge Craft
Expert System for Multi-criteria Dwelling Design Problems	Representation of the building project	Design parameters related to Architecture Acoustics Economics	Rules	Prolog

features required for an effective automation of the building design environment (IBDE). IBDE was implemented in the form of five knowledge based processes that communicated with each other by means of a message blackboard and a project data store.

Hendrickson and Morse [5] presented a conceptual model for automated interactive engineering design, considering both the process architecture and semantic modeling aspects of the problem. A prototype system called FLEX on that model operating in the domain of floor and equipment layout was also presented. It was developed using KEE. KEE was selected because its frame-based object data representation was conducive to the development of the object oriented blackboard process architecture, its facilities for building rule-based systems, and its facilities for evaluation of hypothetical conflict-resolution alternatives.

Flemming [1] considered the 2-D layout problem from the perspective of an ES application. The ES uses a generate and test approach in which generalized rules serve to generate alternative layouts and domain specific rules check the layout at various levels of completeness for semantic integrity.

Rosenman [7] introduced a knowledge based shell called BUILD which was used to build three ES for design evaluation, namely; PREDIKT for interpreting kitchen designs, CODE for checking compliance with the building code and SOLAREXPERT for evaluating passive solar energy systems. The BUILD ESS was developed to provide an environment for developing rule based ESs and include capabilities for explanation, three inferencing strategies : forward, backward and mixed backward & forward chaining, access to external programs and dependency updating.

Evans [8] described the development, implementation and use of rule based calculator, checking structural component designs.

Garret [9] proposed the use of KB techniques to provide a representation that could easily incorporate change as well as serve two purposes : design and design checking. The author

implemented the same in the program called SPEX which used blackboard architecture, in which the knowledge sources were represented by decision tables, classifier trees, heuristic rules for design focus and optimization techniques for constraint based satisfaction.

Chang and Ibbs [10] described the use of probability theory, fuzzy sets, and generalized modus ponens logic for priority ranking in resource allocation.

Mohan [11] commented on the lack of operational ES in the field of construction and presented the characteristics of the state-of-the-art ES in the field of construction management and engineering.

Guene and Zriek [12] designed an intelligent CAAD : an apprentice system capable of acquiring, automatically & by explanation, the know how of an architect to help him in the design process. The system was built around an object language and an automatic interpretation and knowledge acquisition system. They used a learning apprentice system (LAS) named DISCIPLE. KR was by Production rules and semantic nets.

Lee et.al. [13] created a rule based expert system used in the stages of building design synthesis, building defect diagnosis and building contract interpretation. A rule-based ESS was developed to implement the three applications concurrently. An ESS XI Plus from Expertech was used for the development work. It has english like syntax for implementing if-then and when-then rule structures. It also had 'HOW' and 'WHY' explanation facilities.

Out of the above ESs those relevant to this work are given in table 2.2 along with their salient features.

To summarize, it may be stated that a majority of the ES developed so far are available on the PCs, have the rule based KR and are implemented using commercial ESSs. However, a trend was

noticed towards hybrid KR techniques i.e. rules, frames, semantic nets, etc., for design related problems. The use of blackboard architecture, object oriented programming, fuzzy logic etc. was being explored. Nevertheless, rules and inferencing techniques for the same were popular in the situations where the knowledge was of a simple nature as rules were easy to understand and inferencing from them closely resembled the human reasoning.

2.3 EXPERT SYSTEMS

Expert Systems (ES) are a class of computer programs that can advice, analyze, categorize, communicate, consult, design, diagnose, explain, explore, forecast, form concepts, identify,

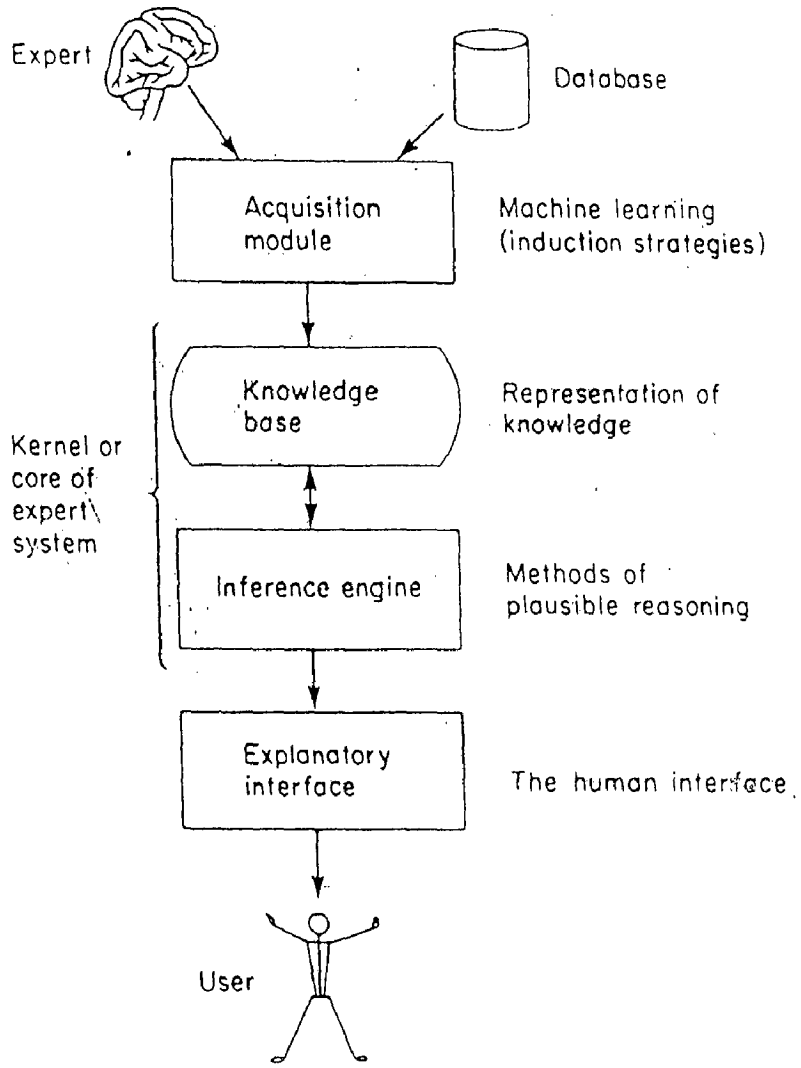


Fig.2.1 Expert systems: a ground plan

interpret, justify, learn, manage, monitor, plan, present, retrieve, schedule, test and tutor [14]. In short, ES are designed to codify and automate the knowledge and heuristics (based on experience and guessing ability) of human experts in a variety of specific problem domains to enable that knowledge to be available for others to use efficiently.

Fig. 2.1 gives an top level architecture of an ES [15].

A brief description of each of these components is given below.

(i) **User** : The end user of the system which includes knowledge engineers, experts from the same or related fields.

(ii) **Explanatory Interface** : It allows for human-machine interaction. It should be as simple and as natural as possible. It can be anything from a simple text editor to standard spreadsheets to menu driven screens to a natural language interface. It is this module which allows the user to interrogate the system, normally by posing 'HOW' and/or 'WHY' questions. 'HOW' questions ask the system to justify is line of reasoning and 'WHY' questions ask it to explain why it requires some piece of information. Both facilities help make the system more usable; but human-machine interaction is still the weakest link in the expert systems technology [15].

(iii) **Knowledge Acquisition Facility** : It is with this facility that knowledge acquired from the domain expert is represented in the computer. This determine the way in which knowledge may be represented. Advanced systems use inductive methods to create rules from examples, thereby discovering new knowledge [16].

(iv) **Inference Engine** : This controls the reasoning process of the ES. It uses the facts and rules in the knowledge base to derive new conclusions leading to a recommendation or a diagnosis. It is basically made up of an interpreter that uses the given knowledge to infer new knowledge and

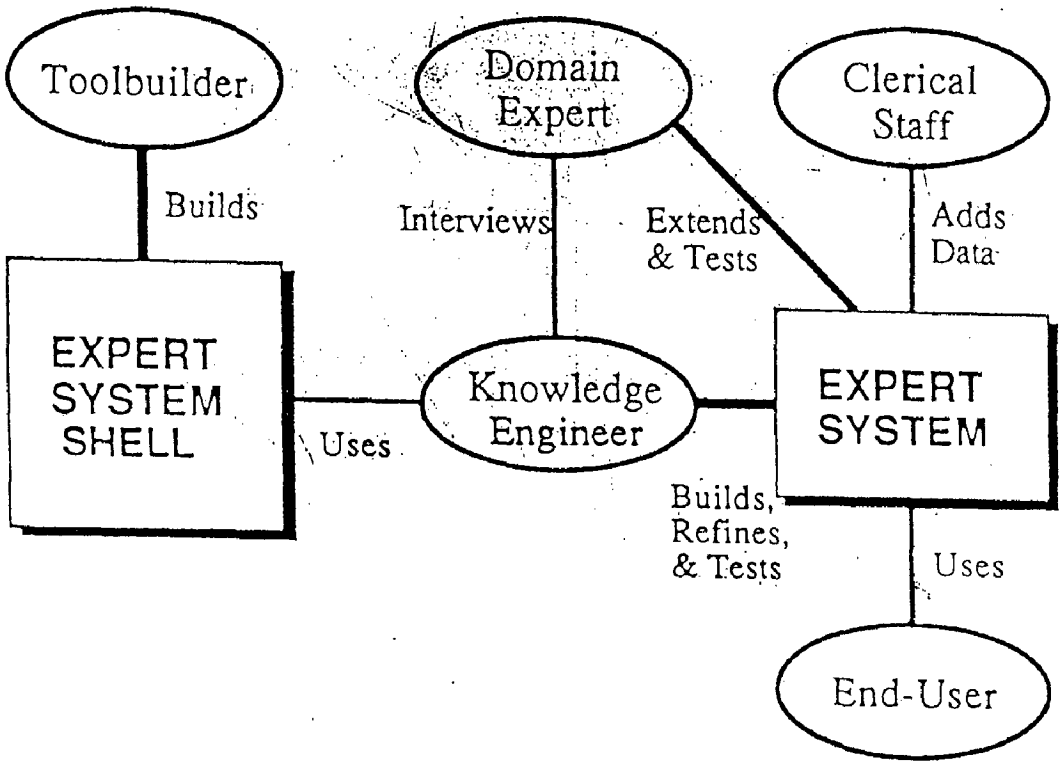

Toolbuilder
Domain Expert
Clerical Staff
Builds
Interviews
Extends & Tests
Adds Data
EXPERT SYSTEM SHELL
Uses
Knowledge Engineer
EXPERT SYSTEM
Builds, Refines, & Tests
Uses
End-User

fig 2.2

scheduler that decides the order in which the knowledge representations are applied. Forward, backward & hybrid of these two are some of the commonly used inferencing techniques [17].

(v) **Knowledge Base** : The knowledge base consists of information structures for encoding expertise. Usually this is elicited from a human expert & reformulated as a collection of rules, a network of facts or a frame based structure. A knowledge base differs from a database in several ways, in particular it is more active. That is, it contains rules for deducing facts that are not stored explicitly.

(vi) **External Programs** : Expert systems cannot operate as stand alone programs when used in complex applications like design. Therefore interfacing with external programs like databases, procedural programs, graphical interface etc. is must [17].

2.4 EXPERT SYSTEM SHELL

An expert system shell (ESS) is an ES stripped up of its domain specific knowledge. An ES is thus domain specific, but ESS is independent of any domain (though they may have features which facilitate design of an ES for a specific domain). Any system in which the interpreter can be separated from domain specific knowledge can be converted into an ESS. Thus interpreter along with a few support facilities is called an ESS [18]. ESS may also be defined as a programming system that simplifies the job of constructing an ES. Fig. 2.2 shows the use of ESS to create an ES and it also shows the programs & players in an ES [14].

2.5 PRODUCTION RULE SYSTEM

The term production rule system refers to several different knowledge representation schemes based on the general underlying

idea of condition-action pairs, which are also called if-then pairs, situation-action pairs, production rule, or just plain productions [19]. Production rules were inspired by the attempts to model the human cognitive process. Production rule system have been shown to be capable of modeling any computable procedure. A production rule base system consists of the following parts :

(i) The rule base, which is composed of set of production rules.

(ii) One or more databases(DBs), which contain whatever information is appropriate for the particular task. Some parts of the database are permanent, whereas others only pertain to the current problem.

(iii) A short-term memory buffer, which represents, the context or current focus of attention of the production rules.

(iv) The interpreter, which determines the task to do next or the production rule to fire next. Its operation is generally closely related to the resolution process.

2.6 SEMANTIC NETWORK

A semantic network is a structured form of Knowledge Representation (KR). It is a declarative form of KR. It is general enough to be able to describe both events & objects. They give a simple, structural picture of a body of facts. Thus a semantic net is a collection of predicate calculus expressions that can be represented by a graph structure. A semantic net has two parts :

(*a*) **NODES** : Entities - Instances, class, subclass e.g. John, Mammal, Home, Floor, Foundation, Rock, etc.

(*b*) **LABELLED ARCS** : These represent relationships among nodes .e.g. isa, owner_of, has, does, covering, are, is, etc. The arrow on the lines point from an object to its value along the corresponding attribute line. The direction of the arrow is very important.

A sample semantic net is shown in fig 2.3.

A semantic net is a collection of frames. The individual frames represent the collection of attributes & values associated with a particular node. Frame add intelligence to the data representation and allow objects to inherit values from other objects. Thus an inheritance network can be created as a special case of semantic nets. The software provides support for creation of semantic networks & hence inheritance networks.

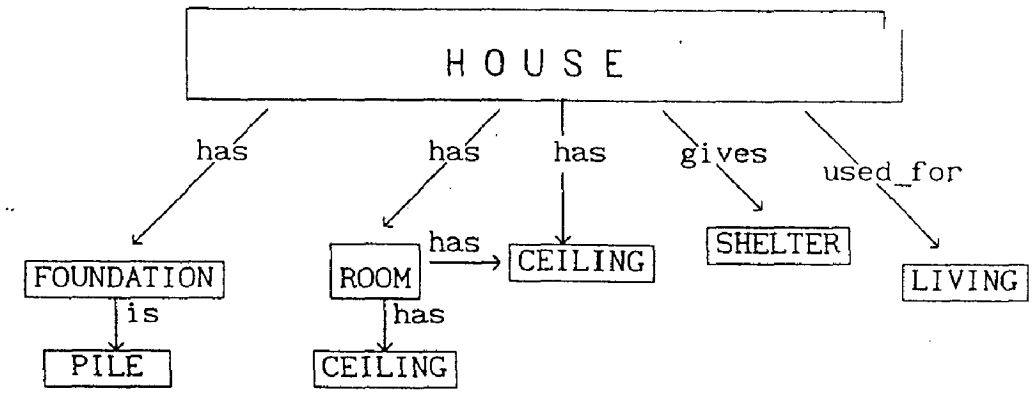


Fig 2.3

A SAMPLE SEMANTIC NETWORK

2.7 DYNAMIC FRAME STRUCTURE

A semantic net is a collection of objects related to each other & forming a graph like structure. Each object represented as a frame consists of no. of attributes associated with it. Take the case of the object "Home" shown in the semantic net of fig 2.3. It has relations "Has", "Gives" & "Used_for". Now suppose we need some new attribute(s) also to be associated with this object "Home". Let the new relation be "isa" with attributes "Building", "temple", & "Tent". The addition of the new attribute should be done without altering the already present structures of other objects. The new attributes are associated with the already present object "Home", dynamically increasing it's size & without affecting the already present database representation. Care has to be taken while adding attributes which are already present, as new relations not clearly visible will have to be created. After adding the relation "isa" with three attributes in the semantic net (see fig 2.4), each attribute gets associated with the object "Home" in turn, thus preserving the consistency of the semantic network. This shows the dynamics of the semantic net structure in

terms of the no. of relations. The structure is also dynamic in terms of the no. of attributes with each relationships. As seen in the fig 2.4, the object "Home" has three occurrences of relation "Isa" & "Has" while one occurrence of relation "Gives" & "Used_for". Thus there are two dimensions of dynamics in this kind of semantic net representation. This approach makes updations much easier with no (or little) overhauling of already present database.

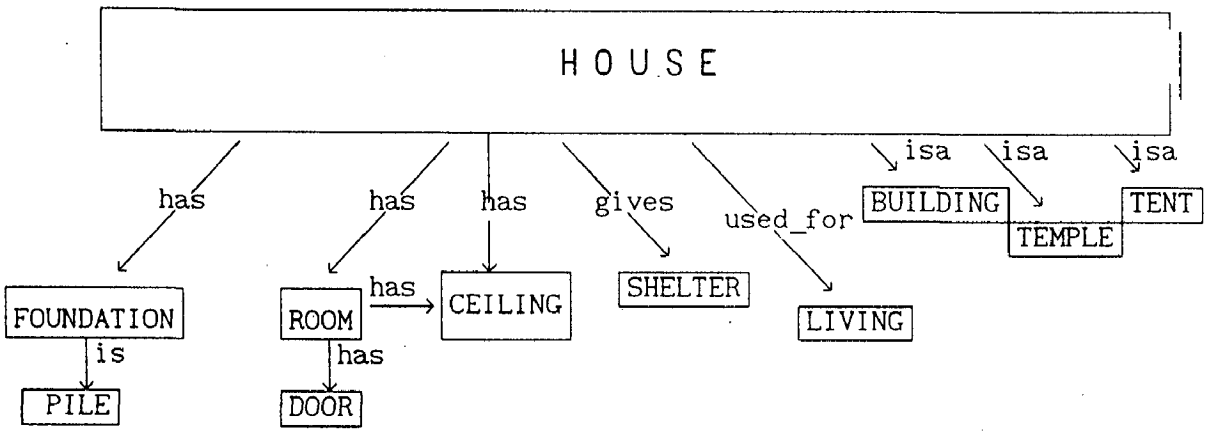


Fig 2.4

SEMANTIC NET OF FIG.- 2.3 AFTER ADDING NEW RELATIONS "ISA" TO OBJECT "HOME"

CHAPTER 3
DESCRIPTION OF EXPERT SYSTEM SHELL

3.1 SOFTWARE DESCRIPTION

This section describes the ESS SHELL from the users viewpoint. For this this ESS can be broken up into three main modules :

(a) Module for creation of semantic net with dynamic frame structure.

(b) Non-learning Rule based Inferencing module.

(c) Learning Rule based Inferencing Module.

(a) Module For Creation Of Semantic Net With Dynamic frame structure :

module creates the DB of facts used by the two modules. The Semantic net as a form of KR for SHELL was chosen because it is very easy to communicate facts with the help of a semantic net. Besides semantic nets can themselves be used independently to provide much information to user based on queries by exploiting inheritance feature of the objects in it. Various facilities have been incorporated which exploit the inheritance feature of the semantic net. There is no restriction on the type of knowledge that can be stored in the semantic [18]. The knowledge from several independent domains can be stored in one file & thus each isolated semantic nets form a sub-domain with a complete knowledge being the integration of all the sub-domains. Facility is provided which can inform one of the head objects (i.e. the objects which have no parents but only children), so that one can separate each sub-domain for his purpose. The various facilities incorporated in this module are ;

(i) Create / Update the KB.

(ii) View the objects with their attributes, called simply qualifiers here.

(iii) View all or only those objects which do not occur as attribute(s) of any other object(s).

(iv) Traverse the KB in any direction or randomly and view various relations to/from each object.

(v) Find the path between two objects w.r.t. certain relation(s) or with no fixed relations.

(vi) Have queries answered for finding a value of an object w.r.t. a given relation.

(vii) With respect to a certain relation find objects which have children w.r.t. that relation or find objects which are children of some objects w.r.t. to that relation.

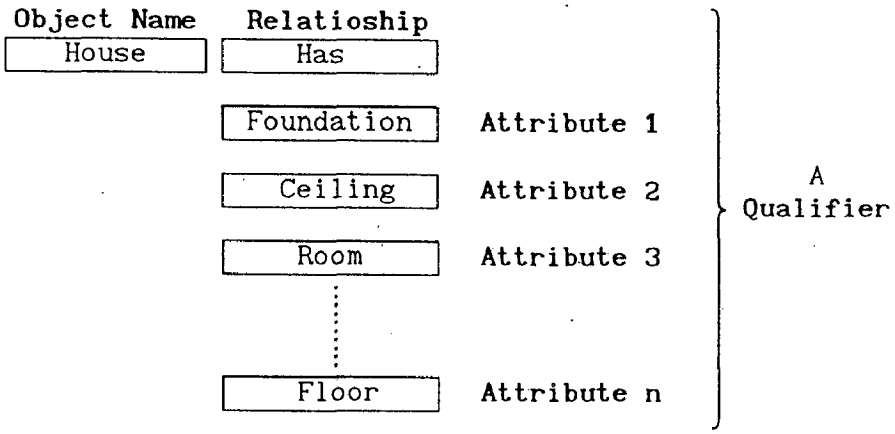


Fig. - 3.1

SAMPLE DATA I/P FORMAT

(i) Creation / Updation Of Knowledge Base :

The data is entered in the format shown in the fig 3.1. As seen from the fig a the information is entered as relationship of an object (here "House") w.r.t. other objects called attributes (here "Foundation", "Ceiling", etc.) with a certain relation (here "has"). At this stage care has to be taken so that new data entered does not result in KB inconsistency. If such a qualifier already exists in the KB, the user is prompted, similarly if the

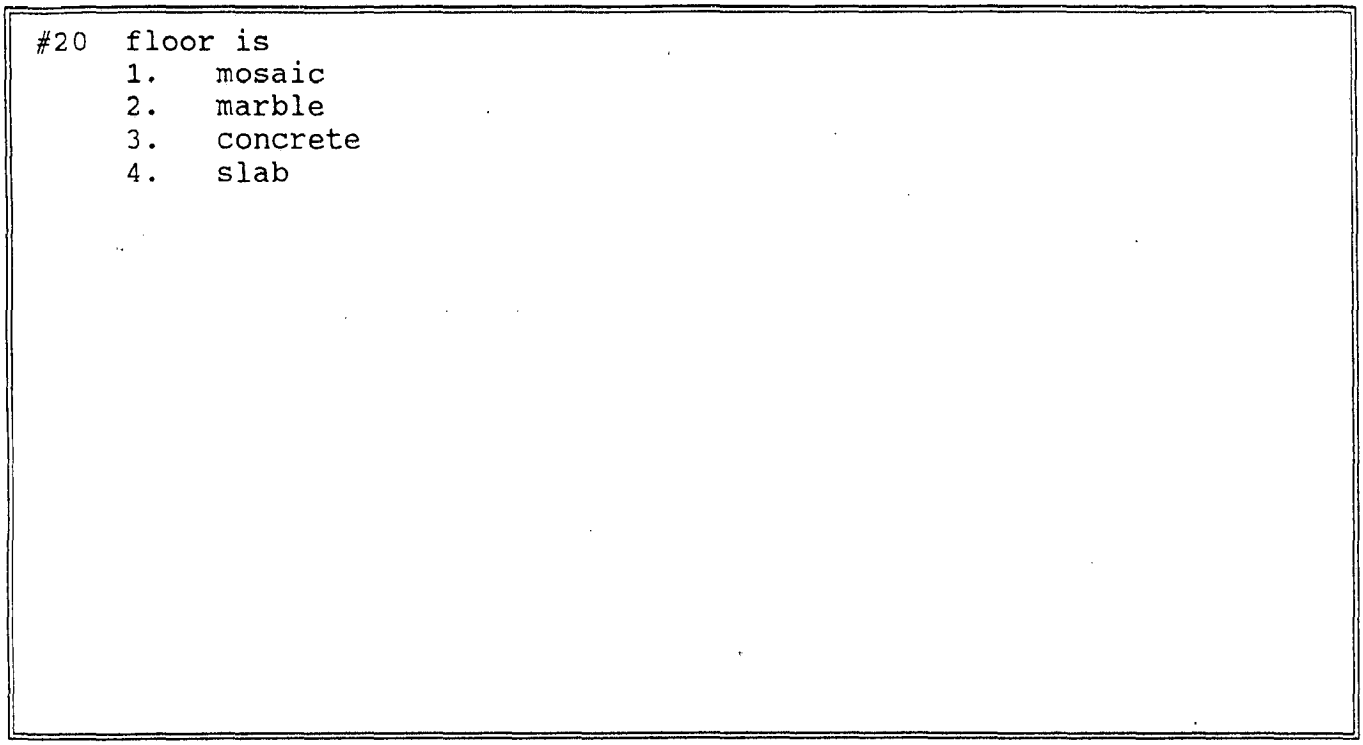

#20 floor is
1. mosaic
2. marble
3. concrete
4. slab
Previous(← or ↑), Next(→ or ↓), Qualifier #(N), Qualifier name(M), Exit(X)

fig 3.2

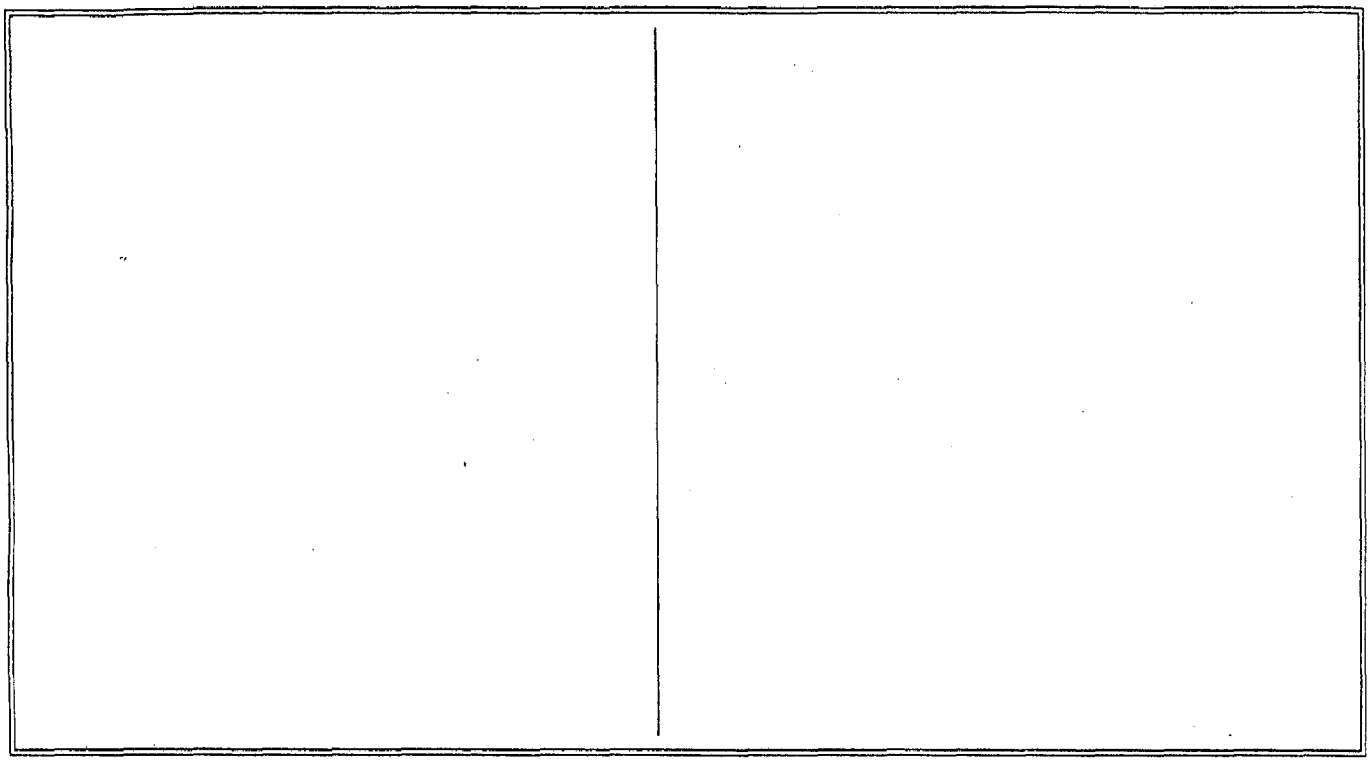
relationship or object name is a qualifier is missing an error message is given. Appropriate error messages are also given if two attributes are same or the object and attribute name are the same.

This facility is used not only to introduce new objects in the semantic net, but also to add more information to the objects already existing in the knowledge base. Eventually one is expanding the already existing semantic net by this facility.

Viewed from the perspective of the inferencing modules one is adding facts and thus expanding within the domain, areas on which decisions can be provided. If one wants proper decisions, it is necessary to store facts using this facility properly, as any discrepancy will lead to the inferencing modules providing false or no decisions.

(ii) View The Objects With Their Attributes :

This facility displays all the objects and the attributes associated with them w.r.t. a certain relation (fig 3.2). As seen from the fig 3.2, one can browse around the qualifiers entered by him by using cursor keys (for chronological order of display) or use facilities to recall a qualifier by name or by number (each qualifier is assigned is assigned a unique number in ascending order when they are entered). Searching is strong in the sense that even search string like " HoUSe haS ." is able to recall the qualifier "House has". The facility is used to view all the qualifiers entered with the help of the previous facility. This facility can be used to gather facts from the client of the architect about the various objects or components of building that he want, and will not be present when the session with the ESS is underway.



Any text(T), Attribute(A), Related qualifier(R), Parent(P), Exit(X)

fig 3.3

(iii) View All Or Only The Objects At The Highest Level :

A facility is provided to view all the objects which do no occur as attribute(s) of any other object(s). These super objects (as they are called here) are those objects which have only outgoing arcs & no incoming arcs. An example of such an object is the object "House" in fig 2.3. This facility displays all or super objects in alphabetical order for users convenience. The facility for tracking super objects in each sub-domains is useful in future traversal of the complete KB for finding any information from the semantic net. As each object is displayed, so are the information related to the various relationships emanating from the object. In fig 2.3 semantic net for object "House" these will be "House has", "House used_for" & "House gives". If an ordinary object is displayed, the name of the qualifiers which have that object as their attribute are also displayed. This facility can be used to trace out the semantic net manually and check the correctness of the KB.

(iv) Random Traversal Of Semantic Net :

This facility can be use to move to any object in the KB randomly. To start with user is presented with the options shown in fig 3.3. The user enter a string which may be an object name or the name of a qualifier. As an string is entered and associated information displayed, the available to the user (fig 3.4) allows him to go to the qualifiers attribute object, see any related qualifier or see any qualifier which are containing the specified object (called parent qualifiers), or he can simply move to any unrelated object in the KB by entering a new search string. Thus three degrees of freedom of movement can take place from any point, thus allowing a flexible, manual traversal of the semantic net to gather any information. This facility can be used to gather information if one is not able to get information from the remaining query answering facilities.

#20 floor is 1. mosaic 2. marble 3. concrete 4. slab	Present in following qualifier(s) : 1. (# 5) Room has Related qualifiers : 1. (#20) floor is

Any text(T), Attribute(A), Related qualifier(R), Parent(P), Exit(X)

fig 3.4

1. House has Room 2. Room has floor	DESTINATION : FLOOR SOURCE : HOUSE RELATION(S) : GENERAL SEARCH

)ath between the Source and Destination. ...Press any key

fig 3.5

In this facility, if the string entered is present both as an object as well as a qualifier, the user is provided a chance to take decision to continue traversing from any one of them. Thus a perfect tool is provided to gather any information from the semantic net depending upon one's query.

(v) Find A Path Between Two Objects :

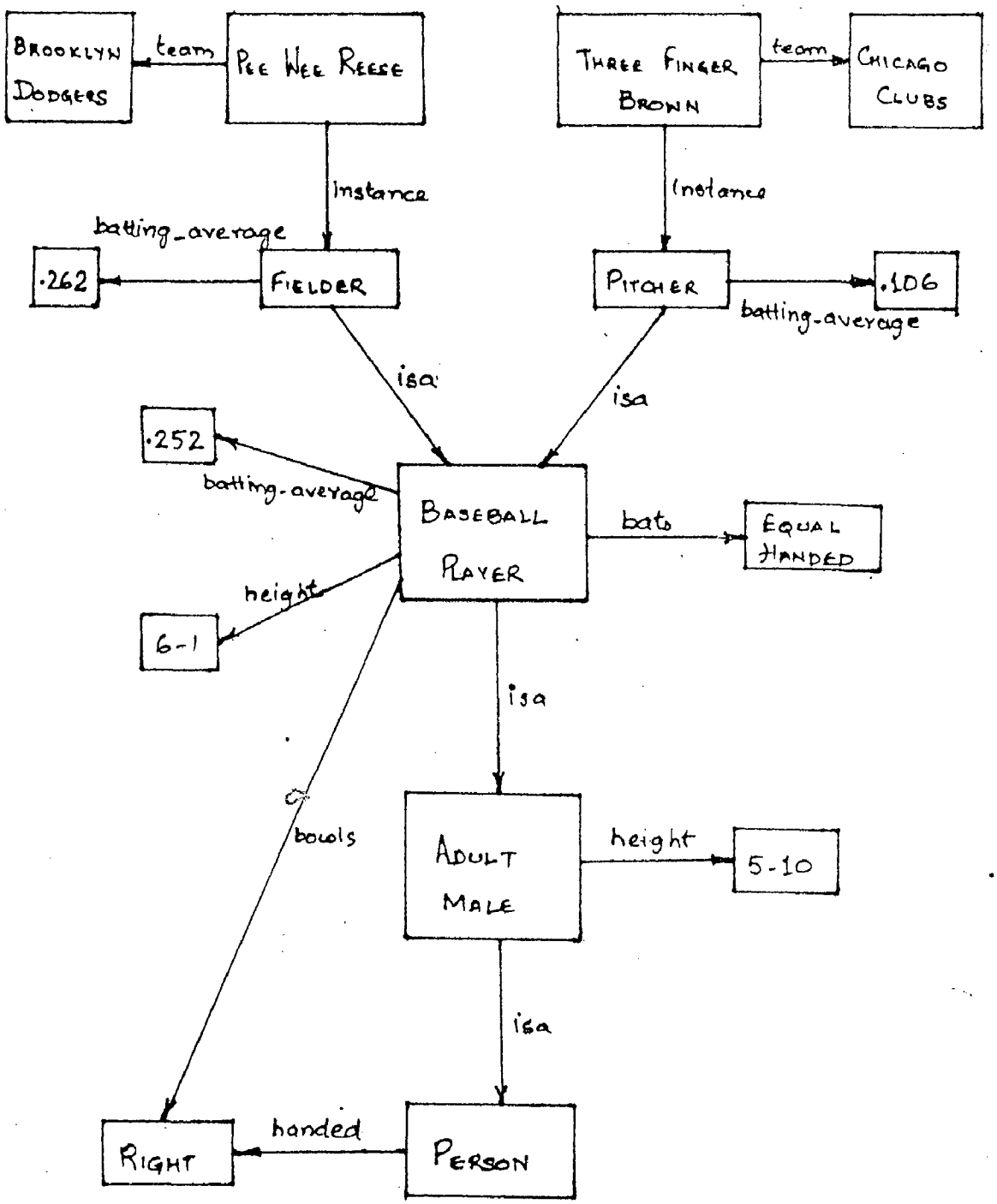
A facility is provided to allow one to find a path between two objects. One has option to specify a set of relations as a basis of search or can even specify a search on all the relation in the KB (fig 3.5). This is a specialized facility, using the connectivity of the objects in the semantic net forming a graph. This facility is used to test connectivity between two objects and if no link is found (but was expected), one can be sure to deduce that KB is incomplete. In the context of the semantic net alone, this facility can be used to confirm or give the reasoning that why certain fact is true, like if one has a semantic net for storing facts about the animals in general, then this facility can be used to prove on the basis of the net, why a dog is a mammal, for which the line of reasoning may be "Dog isa animal", "animal are mammal".

(vi) Finding A Value Of An Object w.r.t. A Given Relation :

This facility answers queries about values of an object w.r.t. a given relation. The simplest case is of display of a qualifier attributes. More complicated cases involve exploitation of the hierarchical feature of the objects in the semantic net. For the semantic net of fig 3.6, a sample query is shown in fig 3.7. This too is a specialized facility like the previous one. This facility can also be used to check the correctness of the semantic net in the sense :

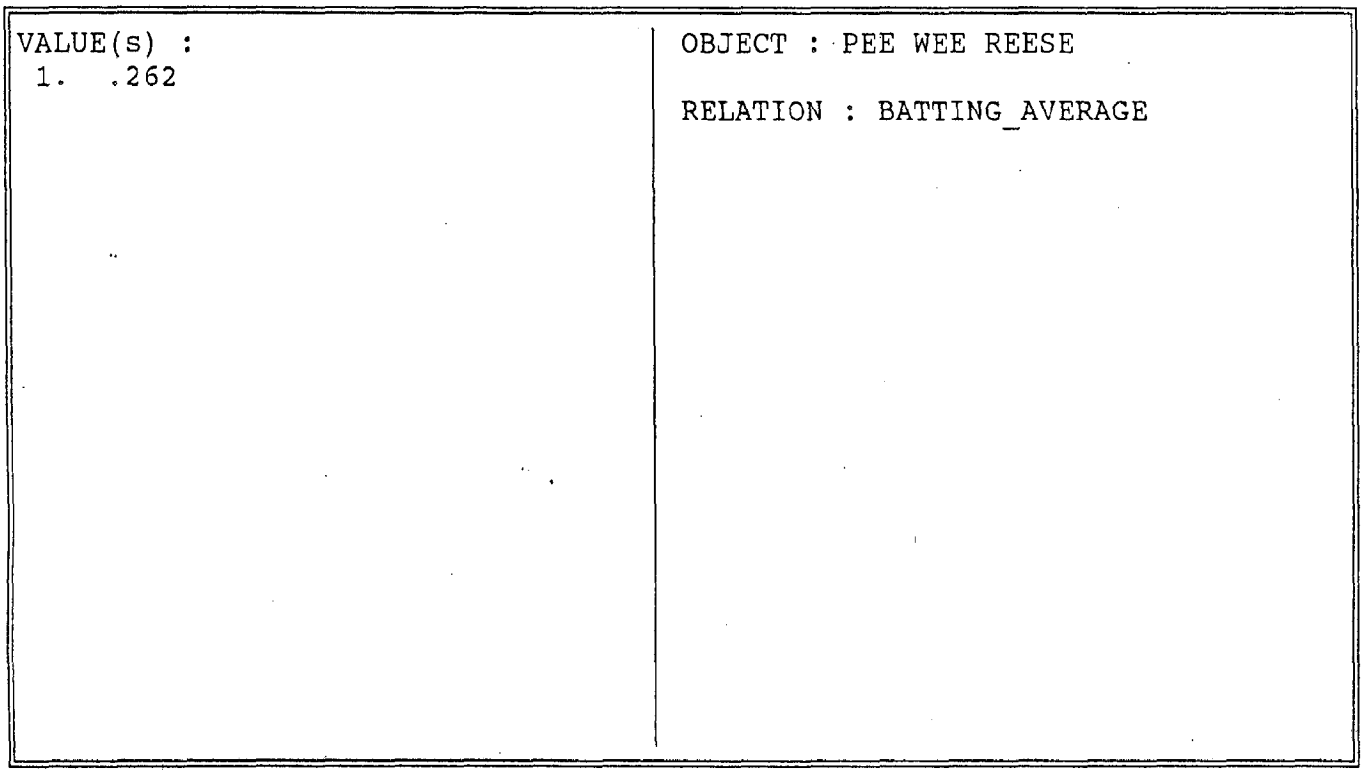
(a) If value(s) is (are) found, whether it is correct.

(b) If no value if found but one was expected.



A Semantic Network

fig 3.6



Value is..... Press any key to continue.....

fig 3.7

These tests ultimately help increase one's reliability in the decisions provided by the ESS's IE modules.

(vii) Find Objects Related To Other Objects w.r.t. To A Certain Relationship :

This facility helps one find objects which are attached to other objects w.r.t. a certain relationship. For a given relationship like "isa" there are objects which form qualifiers with it, "Corridor isa", "Bedroom isa", etc., while there are objects which occur as attributes of these qualifiers like "circulation", "room", etc.. This facility can display list of names of the either type of objects. This facility is very useful in checking the correctness of the semantic net w.r.t. a certain relationship, as these facts are used by the learning IE module. The decisions will be unreachable or may be incorrect if proper semantic net is not maintained. This facility may also be used to find a list of objects which are associated with a certain relation, a specialized function w.r.t. a semantic net.

(b) Non-learning Rule Based Inferencing Module :

This module has rules in the form of if-then statements forming the rule base. A sample production rule used by this module is shown in fig 3.8.

```
IF
   Weather is sunny
   [Temp] > 37
THEN
   Drink Water - 10/10
   Take Umbrella while going out - 10/10
```

fig 3.8

SAMPLE PRODUCTION RULE USED BY NON-LEARNING MODULE

The operation of a transformation rule or production rule in a design grammar is shown in fig h.

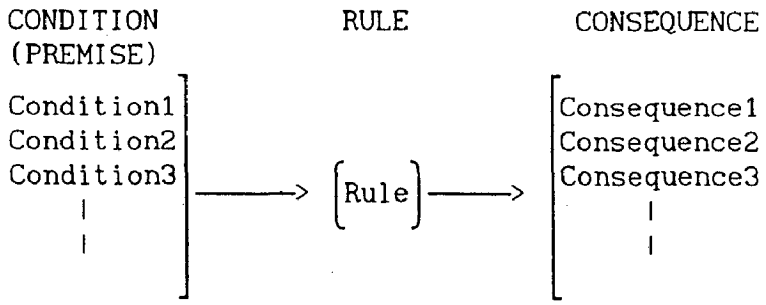


fig 3.9

OPERATION OF A PRODUCTION OR TRANSFORMATION IN A DESIGN GRAMMAR

With respect to the rule in fig 3.8, the conditions in fig 3.9 are "Weather is sunny" & "[Temp] > 37", while consequences form the decision taken by the ESS. Thus a rule simulates the thinking or decision mechanism of a human being, who decides on something (consequences) only when conditions (premises) hold true. The advantage of this kind of ESS is the ease of transfer of knowledge from the domain expert to the ESS KB.

Various facilities provided by this module are :

(i) Enter new rules
(ii) Save rules
(iii) View entered rules .
(iv) Fire rules
(v) View successful/failed rules
(vi) View results or decisions reached

(i) Enter New Rules :

The user is allowed to enter rules whether temporary or permanent to get decisions based on the facts valid for the session. The format of the rule is shown in fig 3.8. The user has to input certain conditions or premises forming the "IF" part of the rule

as well as certain consequences forming the "THEN" part of the rules. The "THEN" part as explained earlier constitute the decisions taken by the ESS.

The user while entering the premises has two choices viz. either use a qualifier or use a logical mathematical expression to decide upon the present situation. In fig 3.8 the first premise was formed by using a qualifier "Weather is", while the second one was formed by using a logical mathematical expression "[Temp] > 37". For the first premise the complete qualifier is shown in fig 3.10.

Weather is

1. Sunny
2. Rainy
3. Normal

fig 3.10

A sample qualifier used in rule of fig 3.8

User chooses one or more instances of the qualifier to create a premise, so that when the rules are fired, the instance which hold at that moment will decide which decision has to be taken.

The other method of constructing a premise as already indicated is by using mathematical expressions. The user can use any mathematical expression to test a condition. Besides using variables, various mathematical operations supported are addition (+), subtraction (-), multiplication (*), division (/), power (^), modulus (%), sine (sin()), cosine (cos()), tangent (tan()), sine inverse (asin()), cosine inverse (acos()), tangent inverse (atan()), flooring (floor()), ceiling (ceil()), square root (sqrt()), exponentation (exp()) & natural log (log()). Relational operators supported are greater than (>), less than (<), equal to (=) and not equal to (!=). Besides these the use of parenthesis ("(" & ")") is advocated to improve legibility and

understanding of the expressions at a future date. The variables entered can be initialized beforehand, given a text to explain their real name or meaning and also their values can be displayed at the end forming part of the decision list. Thus for an architect this combination can create all sort of conditions which are required to take any decision while designing a house.

In the consequence part of the rule, the user has four options from which he can choose. Unlike the premise construction, the user can also any external program or describe a goal statement with some probability. With the feature of calling an external program, one can thus use SHELL as a central module, initiating various activities from itself, and delegating complex tasks to programs designed to do so. This provides convenience to the user in a sense that, interface with other programs becomes transparent, unlike other case where the user intervention is necessary to pass various data either manually or explicitly. As files are used to pass data between two programs, any amount of data can be communicated between them in either direction. The data communication is in terms of real numbers, with the user specifying the names of the variables to be passed to an external program or those returned by the external program. Thus the user can not only handle complex tasks, he can use this facility to display intermediate results or designs, display error graphically, get any data which requires complex interaction or which requires iterative interaction with the user, initialize variables if they are larger in number or if their values depend upon the project handled, etc.. Thus implications of this facility are limited only by the imagination power of the user

The other method of consequence construction which involves using a text associated with some probability rating in the scale of 0 to 10 is shown in fig 3.8. Probability rating of 0 means that the method or conclusion is absolutely false, while a rating of 10 implies absolute correctness of a fact or method. The texts

associated form the decisions of the SHELL. These texts are displayed at the end of the firing session, along with the probability associated (0 to 1). These consequences thus imply certain tasks or conditions valid for a project with some probability.

(ii) Save Rules :

A user may sometime want to create some rules temporarily for a session or he may want to test the validity of the rules before inducting them into permanent KB. The user is thus allowed to create rules and save them at his discretion. Once the rules are saved they form the part of the permanent KB.

(iii) View Rules :

User may want to view all the rules he has created till date. The list of rules displayed also contain the temporary rules or rules which are not yet part of the permanent rule base. This facility may be used to check the rules entered for correctness. The user is allowed to browse around the list of rules using cursor keys or move randomly to a rule by its number (fig 3.11).

(iv) Fire Rules :

The rules are fired starting from the first rule to the last rule sequentially (forward chaining) by default, unless some condition requires some other rule(s) to be fired altering the normal sequence of rule firing (backward chaining). Thus order in which the rules are fired depend upon the the types of rules (premises and consequences), and their relative ordering. Thus the order in which the same rules are placed may provide different result or different course of interaction if, the rules are not serializable [23]. When firing rules, SHELL prompts the user to provide facts only when they can not be deduced from the rule base. This minimizes unnecessary interaction with the user.

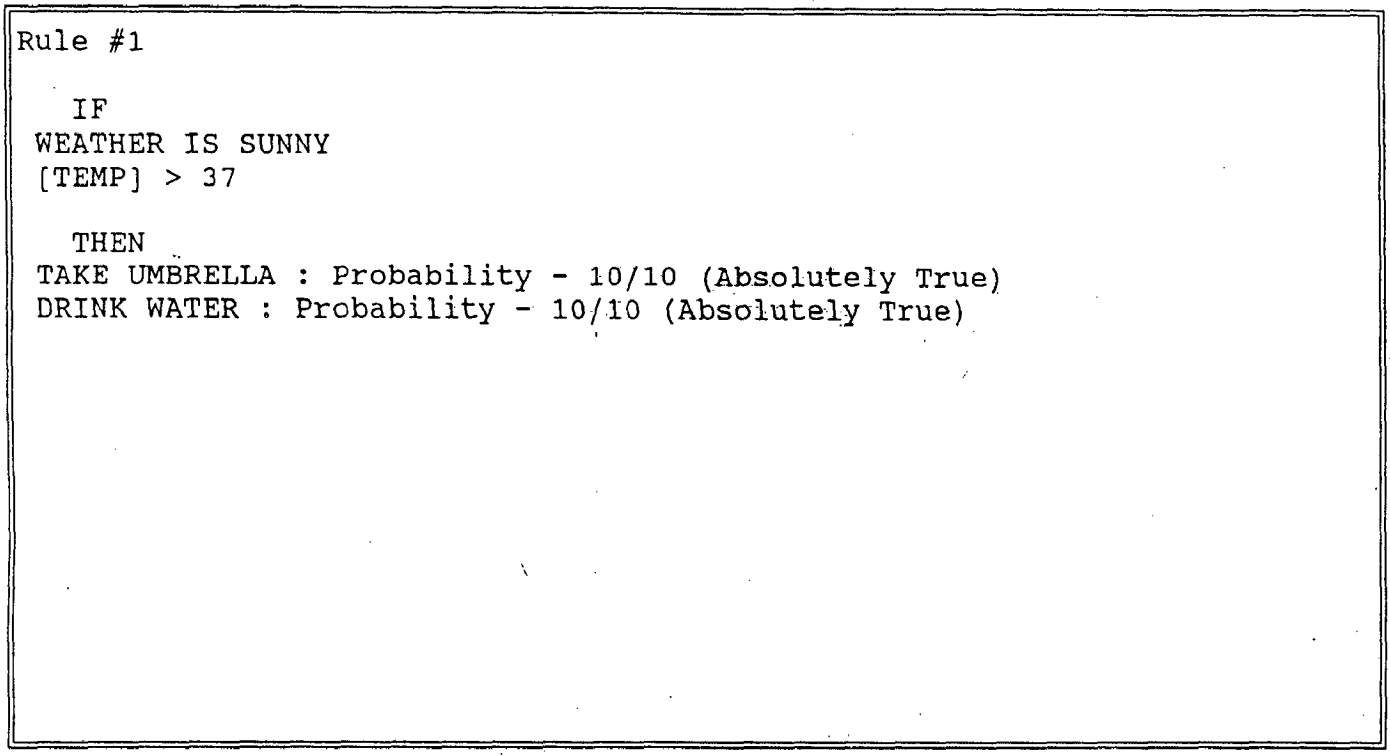

Rule #1
IF
WEATHER IS SUNNY
[TEMP] > 37
THEN
TAKE UMBRELLA : Probability - 10/10 (Absolutely True)
DRINK WATER : Probability - 10/10 (Absolutely True)
revious(← or ↑), Next(→ or ↓), Rule #(N), Exit(X)

fig 3.11

Whenever prompted to provide a fact the user is allowed to ask the ESS the reason by using the "WHY" facility and check the validity of IE for its reasoning methodology. This facility again help boost the confidence of the user in SHELL's decisions. At the end of the inferencing session, SHELL displays the various results it has prepared.

(v) View Successful / Failed Rules :

These facilities act as a crude form of implementation of "HOW" explanation facility. This can explain to the user how the results were reached, by informing him of the succesful and failed rules. These facilities can then be used to confirm the decisions of SHELL. These modules again help boost the confidence of the user in the SHELL's decisions. This facility may be suppressed once the validity of the IE has been checked and confirmed brick-to-brick.

(vi) View Results or Decisions :

The user is allowed to view the results of the previous session at any time, if he is not able to recall the decisions reached during the last session.

(c) Learning Rule Based Inferencing Module :

This module requires rules in the form of if-then statements. A sample production rule used by this module is shown in fig 3.12.

IF

[x] has tail

[x] does barking

THEN

[x] isa dog

fig 3.12

A sample production rule used by learning module

These rules are also transformation rule like fig 3.9. The condition in this case are "[x] has tail" & "[x] does barking", while consequence form the decision taken by the SHELL. This module differs from the non-learning module in the sense that, variable may be used to represent objects in the rules. This module takes the appropriate objects from the semantic net when in the firing session. This module thus relies heavily on the correctness of the semantic net to reach its decisions. This module unlike non-learning module automatically updates the KB both in terms of new qualifiers & in terms of new rules, when it learns to handle new situations. Learning is interactive. The user has to provide/confirm the ESS with appropriate reasons & rules when faced with a new or conflicting situation.

This module provides various facilities listed below :

(i) Enter new rules

(ii) Save rules

(iii) View rules in the rule base

(iv) Fire rules

(v) View successful/failed rules

(vi) View results or decisions

(i) Enter New Rules :

The user can either enter all or partial set of rules applicable to project. The rules entered if not perfect or complete lead to

learning situation at a later time. The format of the rule is shown in fig 3.13. The user has to specify the relationship explicitly, while he can use variables in place of objects to make rules as general as possible. The format of creating the premises and consequences is the same

(ii) Save Rules :

(iii) View Rules in the Rule Base :

These facilities serve the same purpose and have same format in the non-learning rule based module.

(iv) Fire Rules :

The rules are fired sequentially from the first to the last rule and this flow cannot be changed, thus only forward chaining is supported by this module. The rules are displayed as they are fired. The interaction with the user is to get the object name or its instance, whose name is represented by a variable. Method of minimizing this interaction is discussed in 7.2 (Suggestions for future works). The results are displayed as the rules become successful. The result is then used to update the semantic net if the user grants confirmation. If the user is not satisfied with the result, which happens in two cases i.e. if the results were not correct or if the rules were not perfect. In the second case a new learning session starts leading to the refining of the existing rules. Towards this, the user is provided with some explanations, which SHELL thinks is causing conflicts with the previous successful case. The user may then either choose one of the explanations offered to him or may provide a new reason from his own side. In both the cases, the system ends up learning new things & can now handle the present condition in the future correctly simulating the learning methodology of a human being.

(v) View Successful/Failed Rules :

The aim of this facility is the same as in the non-learning module & need no further explanation.

(vi) View results :

This facility displays all the decisions taken by the ESS which resulted in the updation of the semantic net. The use of this facility is the same as in the non-learning module's corresponding facility.

3.2 CHARACTERISTICS OF ESS :

The shell developed has several unique characteristics besides use of the HLL 'C' in its construction. Some of the important characteristics are listed below :

(i) Very user friendly environment, totally menu driven.

(ii) Special data entry routines to minimize data entry mistakes.

(iii) Commands invoked by pressing single keys.

(iv) Incorporation of explanation facility (usually absent in many ESS).

(v) Creation of a backup file where necessary.

(vi) Powerful searching engine (a strong foundation of SHELL).

(vii) Incorporation of facilities to invoke external program to create very powerful packages.

(viii) Very efficient in terms of memory usage.

(ix) Very fast execution.

(x) Extensive error messages provision, to make user understand and rectify his mistake easily and quickly.

(xi) Incorporation of hybrid control strategy, controlled by the user.

(xii) User confirmation where mistakes in data entry are maximum.

(xiii) Use of Dynamic Node Structure semantic net to

represent facts.

(xiv) Incorporation of facilities to solve complex mathematical expression, used extensively in complex application like design.

(xv) Very modular and thus easily understandable for future modifications.

(xvi) Handling of cyclic rules (fig 3.14), preventing the system to enter into a deadlock condition. There is a danger of system entering into deadlock because of the triggering of backward chaining when certain facts can be deduced from the rule base without the user intervention.

#1 if A then B

#2 if B then A

fig 3.14

Sample cyclic rules

As in the rules of fig 3.14, deadlock is broken by prompting the user to provide data to solve B and hence solve rule #2, which will then decide the solution of rule #1.

CHAPTER 4

KNOWLEDGE BASE FOR BUILDING DESIGN

Three examples along with the test runs are presented here, indicating the versatility of the shell to handle different situations.

(i) Foundation design

(ii) Cost analysis

(iii) Fault diagnosis and repair indication

All of these examples have been implemented using the non-learning inferencing module, because of the modules capability to handle the complex rules of these problems.

(i) Foundation Design :

The rules described following can help design the foundation. Inputs required are the various facts related to the soil conditions, and the output is the minimum depth of the foundation.

Qualifiers Used :

#1 Base type_is

1. Rock
2. Soil

#2 Rock type_is

1. Hard rock e.g. Granite, Trap, Diorite
2. Laminated rock e.g. Sandstone, Limestone
3. Broken bed rock, cemented material
4. Soft rock

#3 Soil type_is

1. Cohesive (Clay type)
2. Non-cohesive (Sandy)

#4 Cohesive (Clay type) is

1. Dry hard stiff clay
2. Medium clay (can be indented with thumb nail)
3. Moist clay / Sand mixture
4. Soft clay (can be indented with moderate thumb pressure)
5. Very soft clay (can be penetrated several centimeters with thumb)

#5 Non-cohesive (Sandy) is

1. Compact Gravel / Sand Gravel
2. Compact dry coarse sand
3. Compact dry medium sand
4. Loose Gravel
5. Fine sand/slit (Dry lumps)
6. Loose dry fine sand

Variables used :

S.No.	Name	Text	Initialized	Displayed	Initial-Value
1.	[PI]	Mathematical constant $\prod$	YES	NO	3.14159
2.	[BC]	Bearing constant of soil	NO	YES	----
3.	[PHI]	Angle of repose	NO	YES	----
4.	[MIN_DEPTH]	Minimum depth of the foundation based on soil conditions	NO	YES	----
5.	[DENSITY]	A constant based	NO	YES	----

on soil conditions

External Programs called :

*** *NONE* ***

Methods or Conclusions :

*** *NONE* ***

Rules :

Qualifiers Used :

#1 Base type_is

1. Rock
2. Soil

#2 Rock type_is

1. Hard rock e.g. Granite, Trap, Diorite
2. Laminated rock e.g. Sandstone, Limestone
3. Broken bed rock, cemented material
4. Soft rock

#3 Soil type_is

1. Cohesive (Clay type)
2. Non-cohesive (Sandy)

#4 Cohesive (Clay type) is

1. Dry hard stiff clay
2. Medium clay (can be indented with thumb nail)
3. Moist clay / Sand mixture
4. Soft clay (can be indented with moderate thumb pressure)
5. Very soft clay (can be penetrated several centimeters with

thumb)

#5 Non-cohesive (Sandy) is

1. Compact Gravel / Sand Gravel
2. Compact dry coarse sand
3. Compact dry medium sand
4. Loose Gravel
5. Fine sand/slit (Dry lumps)
6. Loose dry fine sand

Variables used :

S.No.	Name	Text	Initialized	Displayed	Initial-Value
1.	[PI]	Mathematical constant ∏	YES	NO	3.14159
2.	[BC]	Bearing constant of soil	NO	YES	----
3.	[PHI]	Angle of repose	NO	YES	----
4.	[MIN_DEPTH]	Minimum depth of the foundation based on soil conditions	NO	YES	----
5.	[DENSITY]	A constant based on soil conditions	NO	YES	----

External Programs called :

*** *NONE* ***

Methods or Conclusions :

*** *NONE* ***

Rules :

#1

IF

1 = 1

THEN

[PI] = 3.141593

#2

IF

Base type_is Rock

Rock type_is Hard rock e.g. Granite, Trap, Diorite

THEN

[BC] = 3300

[PHI] = [PI]/180*45

#3

IF

Base type_is rock

Rock type_is Laminated rock e.g. Sandstone, Limestone

THEN

[BC] = 1650

[PHI] = [PI]/180*35

#4

IF

Base type_is Rock

Rock type_is Broken bed rock, cemented material

THEN

[BC] = 900

[PHI] = [PI]/180*35

#5

IF

Base type_is Rock

Rock type_is Soft rock

THEN

[BC] = 450

[PHI] = [PI]/180*32

#6

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Cohesive (Clay type) is Dry hard stiff clay

THEN

[BC] = 450

[PHI] = [PI]/180*30

#7

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Cohesive (Clay type) is Medium clay (can be indented with thumb nail)

THEN

[BC] = 250

[PHI] = [PI]/180*45

#8

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Cohesive (Clay type) is Moist clay / Sand mixture

THEN

[BC] = 150

[PHI] = [PI]/180*15

#9

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Cohesive (Clay type) is Soft clay (can be indented with moderate thumb pressure)

THEN

[BC] = 100

[PHI] = [PI]/180*0

#10

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Cohesive (Clay type) is Very soft clay (can be penetrated several centimeters with thumb)

THEN

[BC] = 40

[PHI] = [PI]/180*40

#11

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Non-cohesive (Sandy) is Compact Gravel / Sand Gravel

THEN

[BC] = 450

[PHI] = [PI]/180*35

#12

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Non-cohesive (Sandy) is Compact dry coarse sand

THEN

[BC] = 450

[PHI] = [PI]/180*35

#13

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Non-cohesive (Sandy) is Compact dry medium sand

THEN

[BC] = 250

[PHI] = [PI]/180*30

#14

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Non-cohesive (Sandy) is Loose Gravel

THEN

[BC] = 250

[PHI] = [PI]/180*30

#15

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Non-cohesive (Sandy) is Fine sand/slit (Dry lumps)

THEN

[BC] = 150

[PHI] = [PI]/180*26

#16

IF

Base type_is Soil

Soil type_is Cohesive (Clay type)

Non-cohesive (Sandy) is Loose dry fine sand

THEN

[BC] = 100

[PHI] = [PI]/180*26

#17

IF

[BC] > 0

THEN

[MIN_DEPTH] = ([BC]/[DENSITY])*(((1-SIN([PHI]))/(1+SIN([PHI])))^2)

Test Runs :

#1

Facts :

(i) Base type_is Rock

(ii) Rock type_is Hard rock e.g. Granite, Trap, Diorite

(iii) [DENSITY] is given value 17

RESULTS

1. Bearing constant of the soil : 3300

2. Angle of repose : 0.785398

3. Minimum depth of the foundation based on soil condition : 5.71429

4. Constant based on soil condition : 17

#2

Facts :

(i) Base type_is Soil

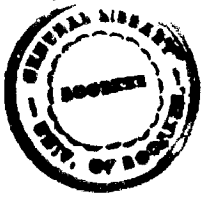
(ii) Soil type_is Non-cohesive (Sandy)

(iii) Non-cohesive (Sandy) is Loose Gravel

(iv) [DENSITY] is given value 17

RESULTS

1. Bearing constant of the soil : 250



2. Angle of repose : 0.523599

3. Minimum depth of the foundation based on soil condition : 1.63399

4. Constant based on soil condition : 17

Test Runs :

#1

Facts :

(i) Base type_is Rock

(ii) Rock type_is Hard rock e.g. Granite, Trap, Diorite

(iii) [DENSITY] is given value 17

RESULTS

1. Bearing constant of the soil :
2. Angle of repose :
3. Minimum depth of the foundation based on soil condition :
4. Constant based on soil condition :

#2

Facts :

(i) Base type_is Soil

(ii) Soil type_is Non-cohesive (Sandy)

(iii) Non-cohesive (Sandy) is Loose Gravel

(iv) [DENSITY] is given value 17

RESULTS

1. Bearing constant of the soil :
2. Angle of repose :
3. Minimum depth of the foundation based on soil condition :
4. Constant based on soil condition :

(ii) Cost Analysis :

This hypothetical example calculates the total cost of estimated in terms of land, infrastructure & dwelling, it also calculates the difference in terms of cost estimated using thumb rules & the cost actually incurred in infrastructure & displays appropriate results.

Qualifires Used :

#1 Site is levelled
1. Manually
2. With Bulldozers

#2 The kind of drains are
1. Open surfaced
2. Hume pipe drains

#3 Landscaping is to be
1. Minimal
2. Moderate
3. Elaborate

#4 Construction is to have
1. Raft Foundation
2. Pile Foundation

#5 Flooring is to be done with
1. Kotah stone
2. Mosaic

#6 Walls are to be
1. White washed
2. Distempered

Variables used :

S.No.	Name	Text	Initialized	Displayed	Initial-Value
1.	[Tot_cost]	Total cost of the project	No	YES	-----
2.	[L_cost]	Land cost	No	YES	-----
3.	[I_cost]	Infrastructure cost	No	YES	-----
4.	[D_cost]	Dwelling cost	No	YES	-----
5.	[DIFF]	Difference in actual and calculated cost the infrastructure	No	YES	-----
6.	[AI_cost]	Actual infrastructure cost	No	YES	-----

External Programs called :

Act_infr --> It supposingly calculates the actual infrastructure cost.

Methods or Conclusions :

1. Deduced infrastructure cost exceeds actual infrastructure cost
2. Actual infrastructure cost exceeds actual infrastructure cost
3. Deduced and actual infrastructure cost are the same

Rules :

#1

IF

Site is levelled Manually

The kind of drains are Open surfaced

Landscaping is to be Minimal

Construction is to have Raft Foundation

Flooring is to be done with Kotah Stone

Walls are to be White washed

THEN

[L_cost] = [T_cost] * 0.40

[I_cost] = [T_cost] * 0.25

[D_cost] = [T_cost] * 0.35

#2

IF

Site is levelled With Bulldozers

The kind of drains are Hume pipe drains

Landscaping is to be Moderate

Construction is to have Raft Foundation

Flooring is to be done with Kotah Stone

Walls are to be White washed

THEN

[L_cost] = [T_cost] * 0.40

[I_cost] = [T_cost] * 0.35

[D_cost] = [T_cost] * 0.25

#3

IF

Site is levelled With Bulldozers

The kind of drains are Hume pipe drains

Landscaping is to be Elaborate

Construction is to have Raft Foundation

Flooring is to be done with Kotah Stone

Walls are to be White washed

THEN

[L_cost] = [T_cost] * 0.36

[I_cost] = [T_cost] * 0.39

[D_cost] = [T_cost] * 0.25

#4

IF

Site is levelled Manually

The kind of drains are Open surfaced

Landscaping is to be Minimal

Construction is to have Pile Foundation

Flooring is to be done with Mosaic

Walls are to be Distempered

THEN

[L_cost] = [T_cost] * 0.49

[I_cost] = [T_cost] * 0.21

[D_cost] = [T_cost] * 0.30

#5

IF

Site is levelled Manually

The kind of drains are Open surfaced

Landscaping is to be Moderate

Construction is to have Pile Foundation

Flooring is to be done with Mosaic

Walls are to be Distempered

THEN

[L_cost] = [T_cost] * 0.42

[I_cost] = [T_cost] * 0.37

[D_cost] = [T_cost] * 0.21

#6

IF

Site is levelled Manually

The kind of drains are Hume pipe drains

Landscaping is to be Elaborate

Construction is to have Pile Foundation

```
    Flooring is to be done with Mosaic
    Walls are to be Distempered
   THEN
    [L_cost] = [T_cost] * 0.38
    [I_cost] = [T_cost] * 0.38
    [D_cost] = [T_cost] * 0.24
#7
   IF
    1 = 1
   THEN
    ([AI_cost]) = "Act_infr" ()

#8
   IF
    [AI_cost] > [I_cost]
   THEN
    Actual infrastructure cost exceeds actual infrastructure cost - 10/10
    [DIFF] = [AI_cost] - [I_cost]

#9
   IF
   THEN
    Deduced infrastructure cost exceeds actual infrastructure cost - 10/10
    [DIFF] = [I_cost] - [AI_cost]

#10
   IF
    [AI_cost] = [I_cost]
   THEN
    Deduced and actual infrastructure cost are the same - 10/10
```

Test Runs :

#1

Facts :

1. Site is levelled Manually
2. The kind of drains are open surfaced
3. Landscaping is to be minimal
4. Construction is to have Raft foundation
5. Flooring to be done with Kotah stone
6. Walls are to be white washed
7. Total Cost is given value 180
8. Actual Infrastructure cost is given value 55

Results :

1. Actual cost exceeds calculated infrastructure cost
   - probability 1 (i.e. Absolutely)
2. Total cost : 180
3. Land cost : 72
4. Infrastructure cost : 45
5. Dwelling cost : 63
6. Actual infrastructure cost : 55
7. Difference in actual and calculated infrastructure cost : 10

#2

Facts :

1. Site is levelled Bulldozers
2. The kind of drains are Hume pipe drains
3. Landscaping is to be Elaborate
4. Construction is to have Raft foundation
5. Flooring to be done with Kotah stone
6. Walls are to be White washed
7. Total Cost is given value 180
8. Actual Infrastructure cost is given value 55

Results :

1. Calculated infrastructure cost exceeds actual cost
   - probability 1 (i.e. Absolutely)

2. Total cost : 180
3. Land cost : 72
4. Infrastructure cost : 63
5. Dwelling cost : 45
6. Actual infrastructure cost : 55
7. Difference in actual and calculated infrastructure cost : 8

(iii) Fault diagnosis and repair indication :

This example illustates the use of SHELL in buildin fault diagnosis and to suggest a repair method. The fact base is illustrated by means of the semantic net of fig 4.1.

Variables used :

**** *NONE* ****

External Program called :

1. *Det_crak* --> Supposingly explains the cause of crack
2. *Exp_repr* --> Supposingly explains the repair method

Rules :

#1

IF

stage of building is during service
concrete member is beam
orientation of crack is transverse
crack pattern is bt3

THEN

cause of crack is excessive bending

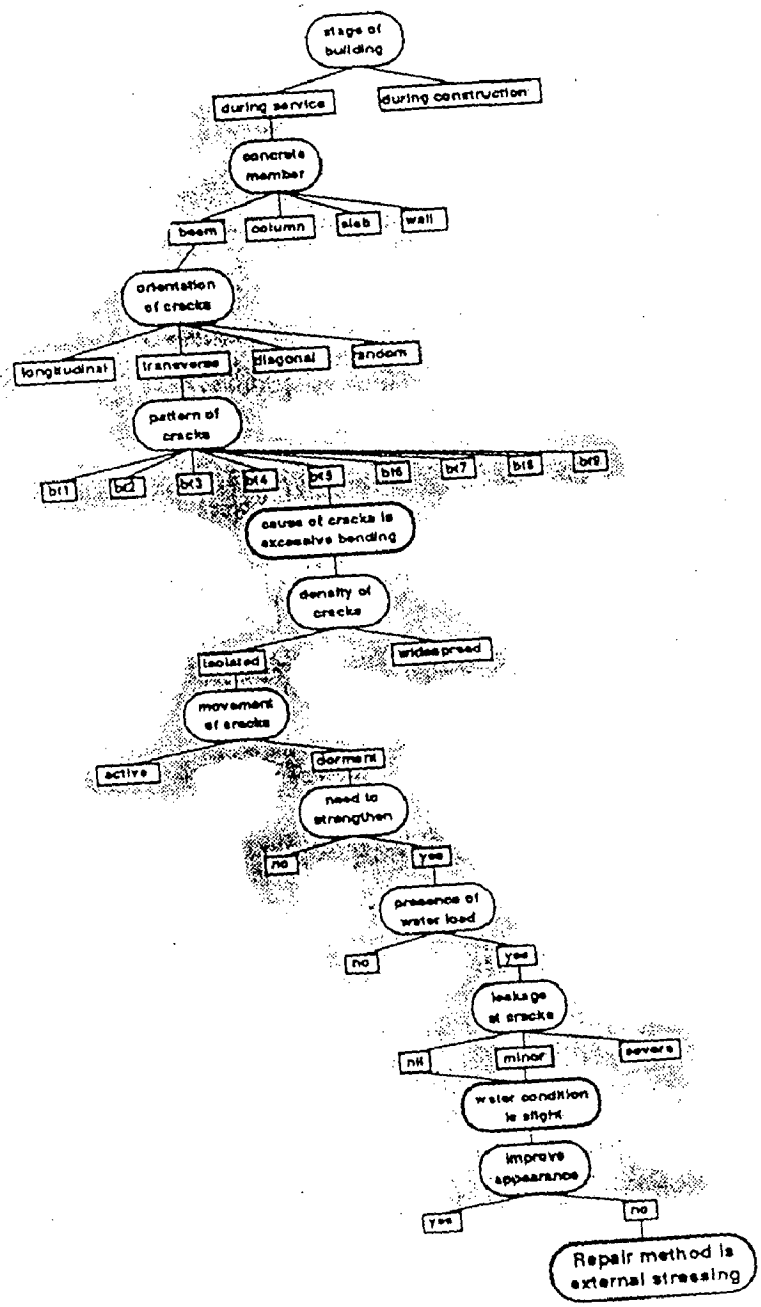


fig 4.1

SEMANTIC NET FOR EXAMPLE -(iii)

```
   IF

#2
   IF
     presence of water load is yes
     leakage at cracks is nil OR minor
   THEN
     water condition is slight

#3
   IF
     cause of cracks is excessive bending
     density of cracks is isolated
     movement of cracks is dormant
     need to strengthen is yes
     water condition is slight
     improve appearance is no
   THEN
     repair method is external stressing

 #4
   IF
     cause of cracks is excessive bending
   THEN
     () = "det_crak" ()

#5
   IF
     repair method is external stressing
     concrete member is beam
   THEN
     () = "Exp_repr" ()
```

Test Run :

A test run was taken with facts which support the fact that external stressing repair is required.

CHAPTER 5

KNOWLEDGE BASE FOR ARCHITECTURAL DESIGN

Architectural design is a job which requires creativity and extensive exchange of ideas between an Architect and the client. So based on the requirement, the learning rule based module was used to implement an example of completing a plan. The problem is to make the module learn how to open the door in the right direction. The changes made or decisions made are projected via the semantic net.

The initial plan consisting of two rooms (a bedroom and a livingroom) leading on to corridor (fig 5.1). The semantic net corresponding to the initial plan is given in fig 5.2. The architect initially provides the system with a crude rule given below.

```
#1
   IF
     [z] isa circulation
     [y] isa room
     [x] isa door
     [x] belong_to [z]
     [x] belong_to [y]
   THEN
     [x] open_towards [y]
```

The architect then places door D1 between corridor and bedroom. The system automatically opens the door towards bedroom (see fig 5.3) and the solution is accepted. The semantic net is updated to fig 5.4. Fig 5.5 shows the configuration for the example concerning the opening of a door towards a room. Fig 6 & 7 show the solution of the situation of opening of the door between the

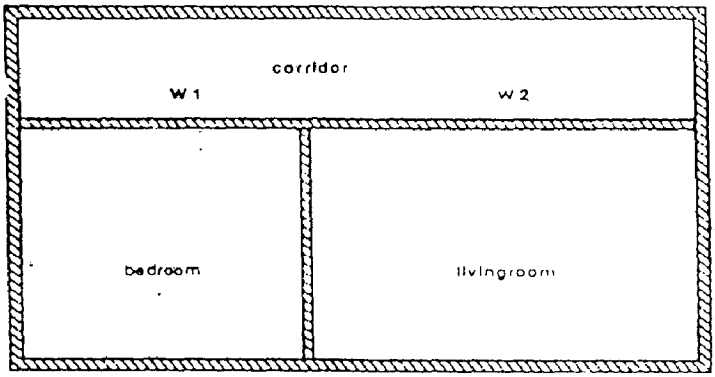


fig 5.1 : Initial Plan

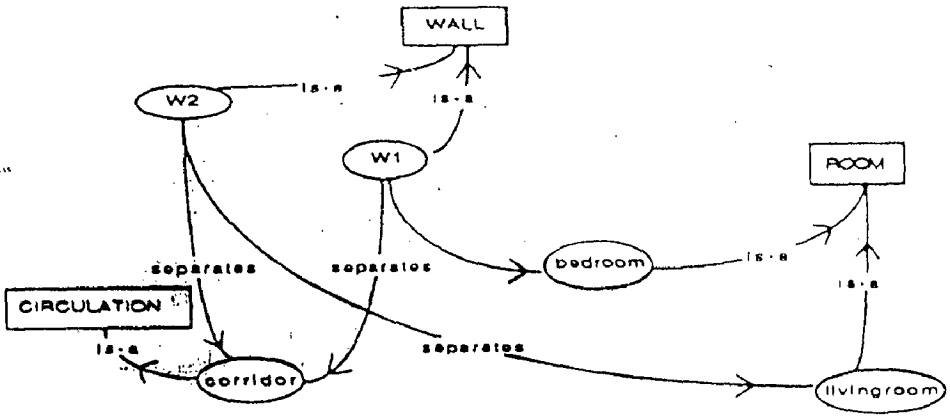


fig5.2 : Semantic network corresponding to the initial plan.

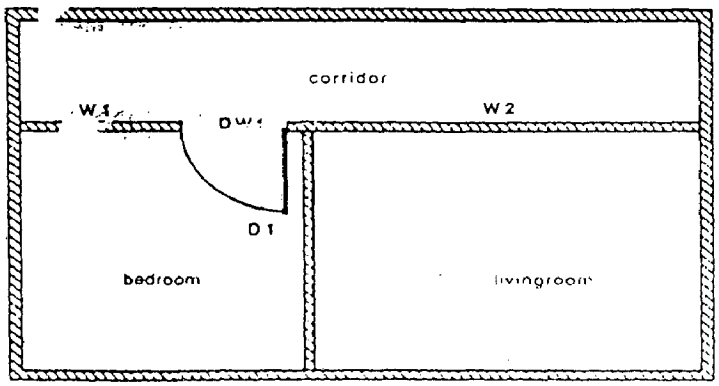


fig 5.3 : Installation of a door between the bedroom and the corridor.

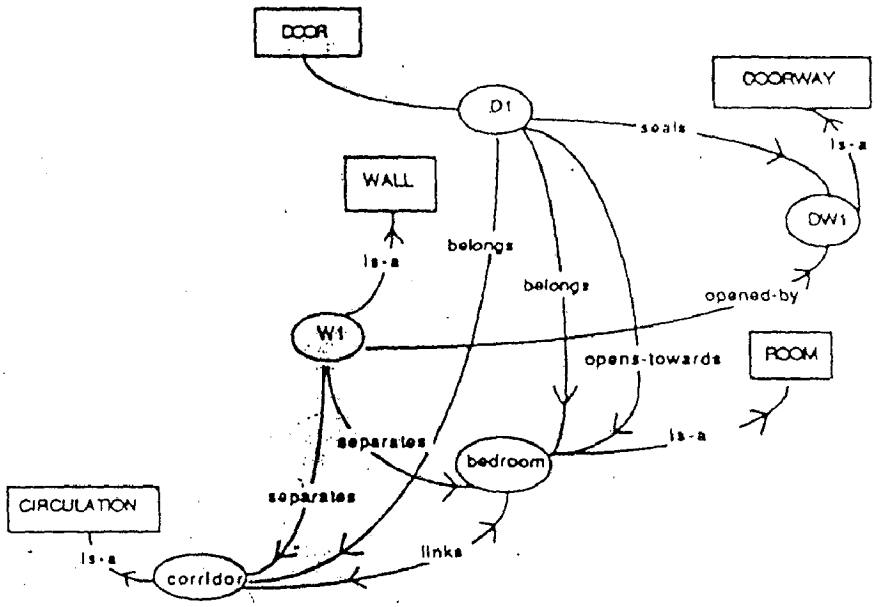


fig 5.4 : The object network after the installation of the door between the bedroom and the corridor.

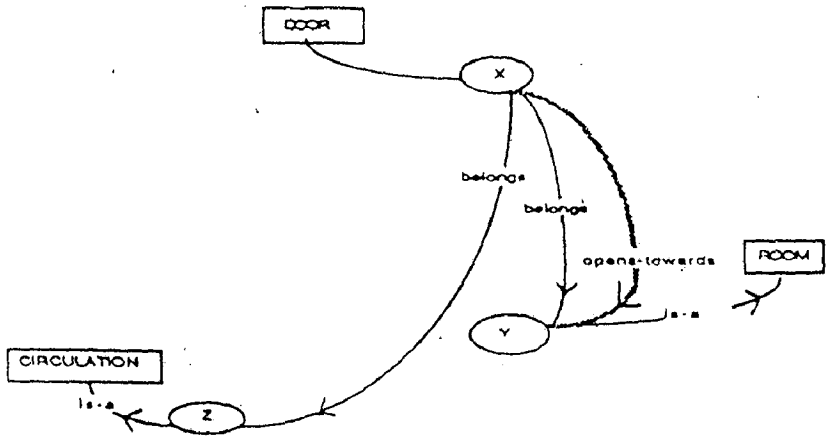


fig 5.5 : configuration for the example concerning the opening of a door towards a room

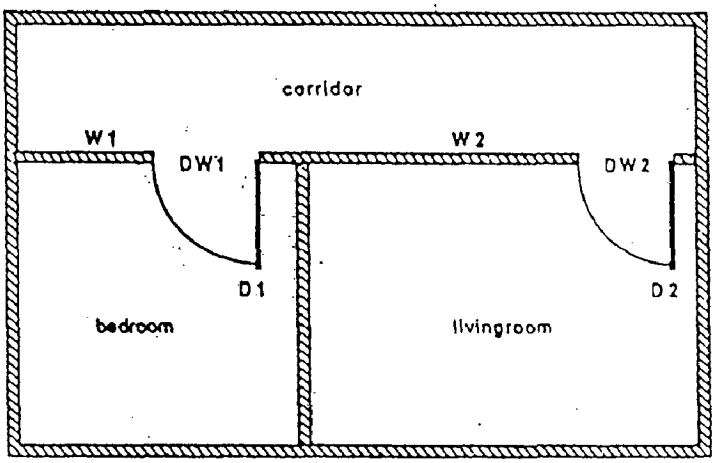


fig 5.6 : The system determines the direction in which the door opens between the corridor and the livingroom

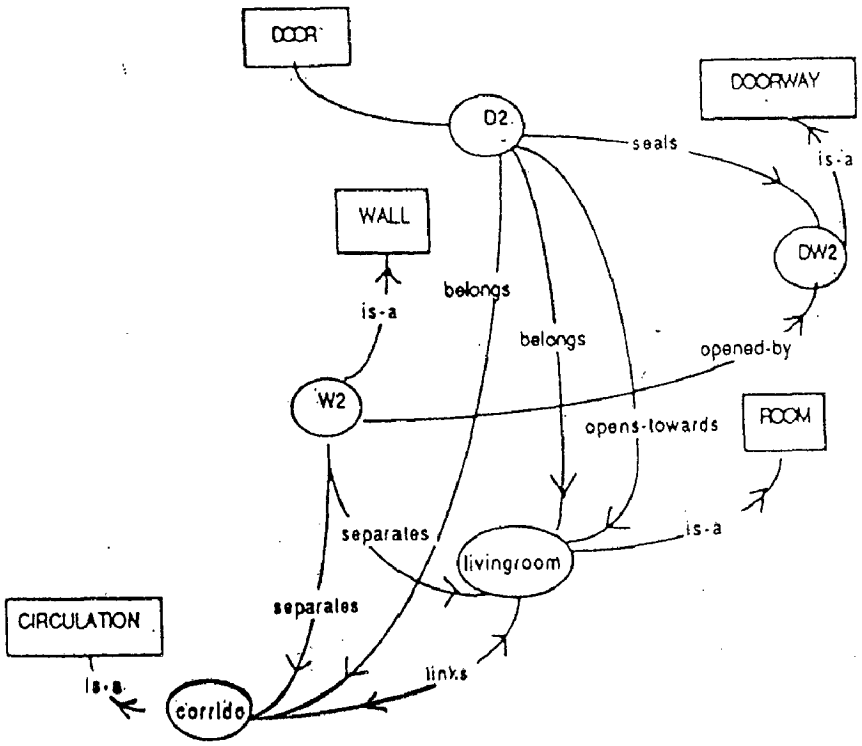


fig 5.7 : The object network after the installation of the door between the livingroom and the corridor.

corridor and the livingroom and the corresponding semantic network. Fig 5.8 shows the situation where a door is to be installed between toilet and corridor. The solution provided by the system is shown in fig 5.9, which is refuted by the architect who sees the solution as opening the door towards the corridor. The system is therefore faced with a counter example of the application of the rule #1 and considers that its knowledge on the direction of opening of door is still incomplete. It tries to specialize its knowledge. It looks for differences & the similarities existing in the configuration of the positive example (fig 5.5) and that of the counter example (fig 5.10). The system proposes following explanations to the designer

1. Toilet contains washbasin
2. D3 seal DW3

. . . . . .

. . . . . .

Explanation 1 is accepted as the possible solution, so rule #1 is modified and duplicated & the new solution for opening a door in case a room contains washbasin is provided viz '[x] open_towards [z]'. Fig 5.11 shows the configuration of an example concerning the direction in which a door opens between a circulation and a room containing bathroom installations. The modified rules are given below.

#1

IF

[z] isa circulation

[y] isa room

[x] isa door

[x] belong_to [z]

[x] belong_to [y]

[y] contains NOT washbasin

THEN

[x] open_towards [y]

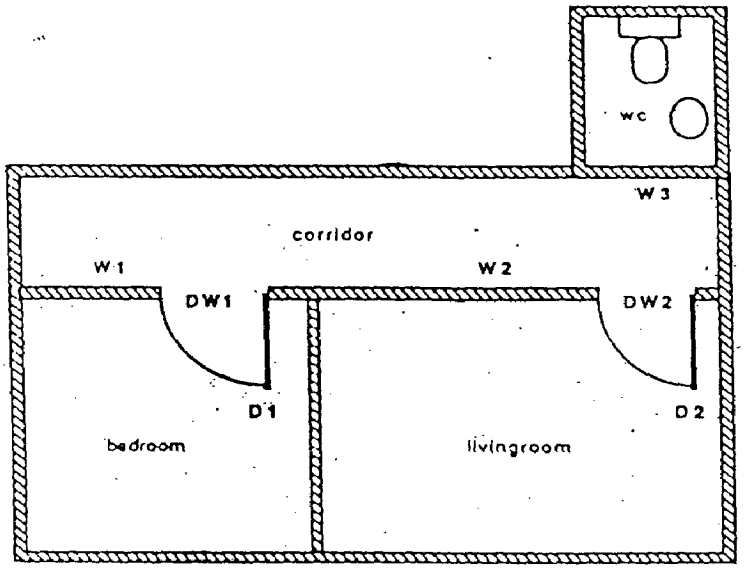


fig 5.8 : Installation of a toilet by the designer

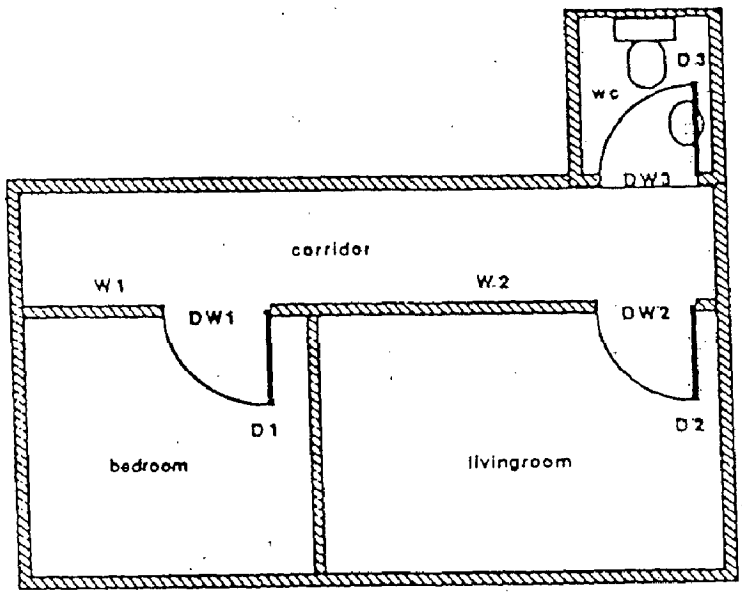


fig 5.9 The system determines the direction in which the door opens between the corridor and the toilet.

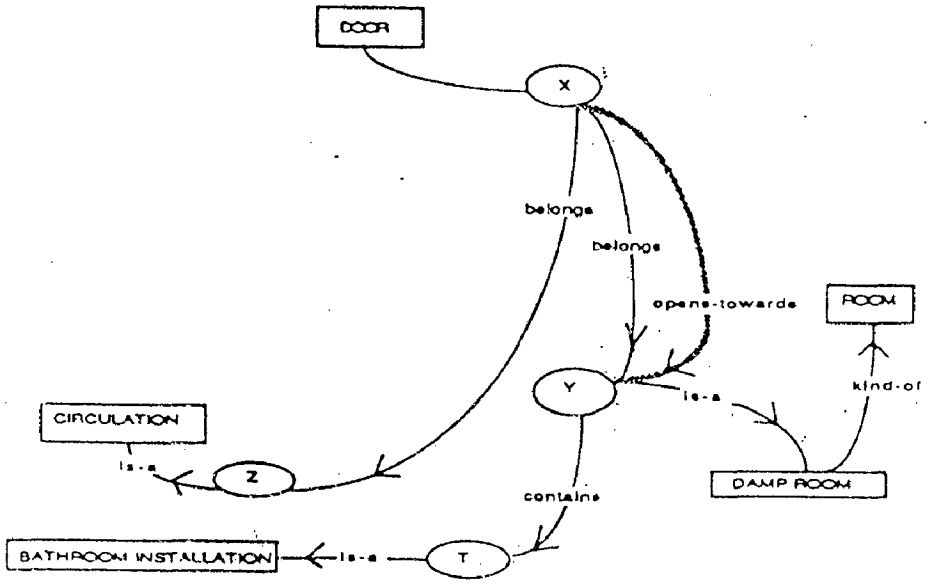


fig 5.10 : Configuration of a counter-example concerning the direction in which the door opens between a circulation and a room containing bathroom installations.

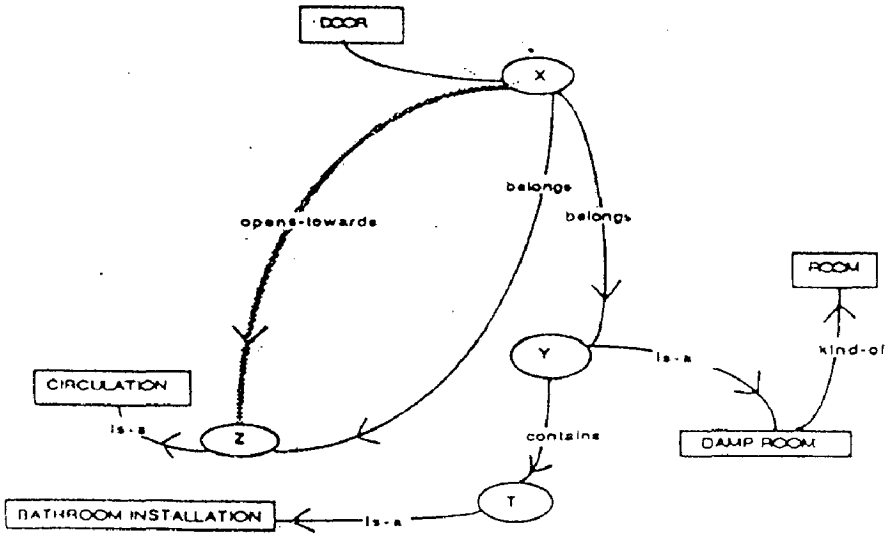


fig 5.11 Configuration of an example concerning the direction in which a door opens between a circulation and a room containing bathroom installations.

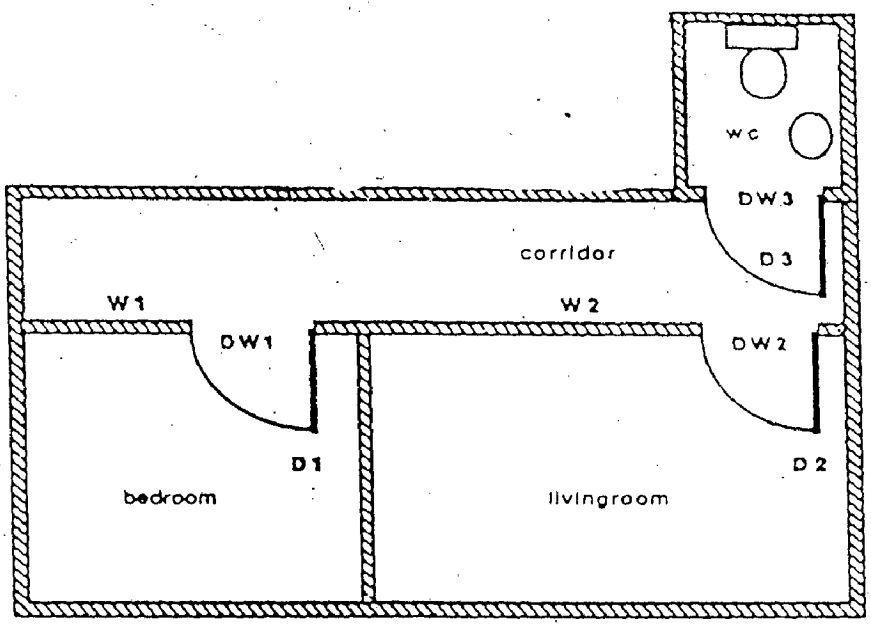


fig 5.12 : State of the plan after the direction in which the door opens has been modified by the designer.

#2

IF

[z] isa circulation

[y] isa room

[x] isa door

[x] belong_to [z]

[x] belong_to [y]

[y] contains washbasin

THEN

[x] open_towards [y]

The rules are fired again and the new solution is presented to the user. The modified plan after the user accepts the solution is shown in fig 5.12. The system thus learns to handle new situation with the help of the architect who teaches it just like a human being giving clues for handling new situation.

CHAPTER 6

IMPLEMENTATION

In this chapter, the various implementation details of the ESS SHELL developed are provided. The details are provide w.r.t. the modular breakup in chapter 3.

(a) Module for Creation of Semantic Net With Dynamic Node Structure

(b) Non-learning rule based inferencing module

(c) Learning rule based inferencing module

(a) Module for Creation of Semantic Net With Dynamic Frame Structure :

The need was felt for storing the data in the semantic net because of the hierarchical nature of the data involved in the field of design. If we consider the fig 5.1, the facts associated with it can be easily and usefully conveyed with the help of the semantic net of fig 5.2. If we take the case of the architect who is designs say houses and towns, the facts can be depicted by the complex semantic net of fig 6.1. The need was felt for creating a semantic net with dynamic frame structure, so that any no. of relations can be associated to any object at time without any prior information about the size and no. of attributes. All this has to be done without compromising on disk or memory space. This has to be done without compromising on disk or memory space. This idea was successfully implemented with no compromise on the flexibility and facilities provided by a semantic net. What follows is the implementation details of each of the sub-modules to create and use the semantic net. The various functions associated with each modules are also described and named.

Data structures used to create and manipulate the semantic net are

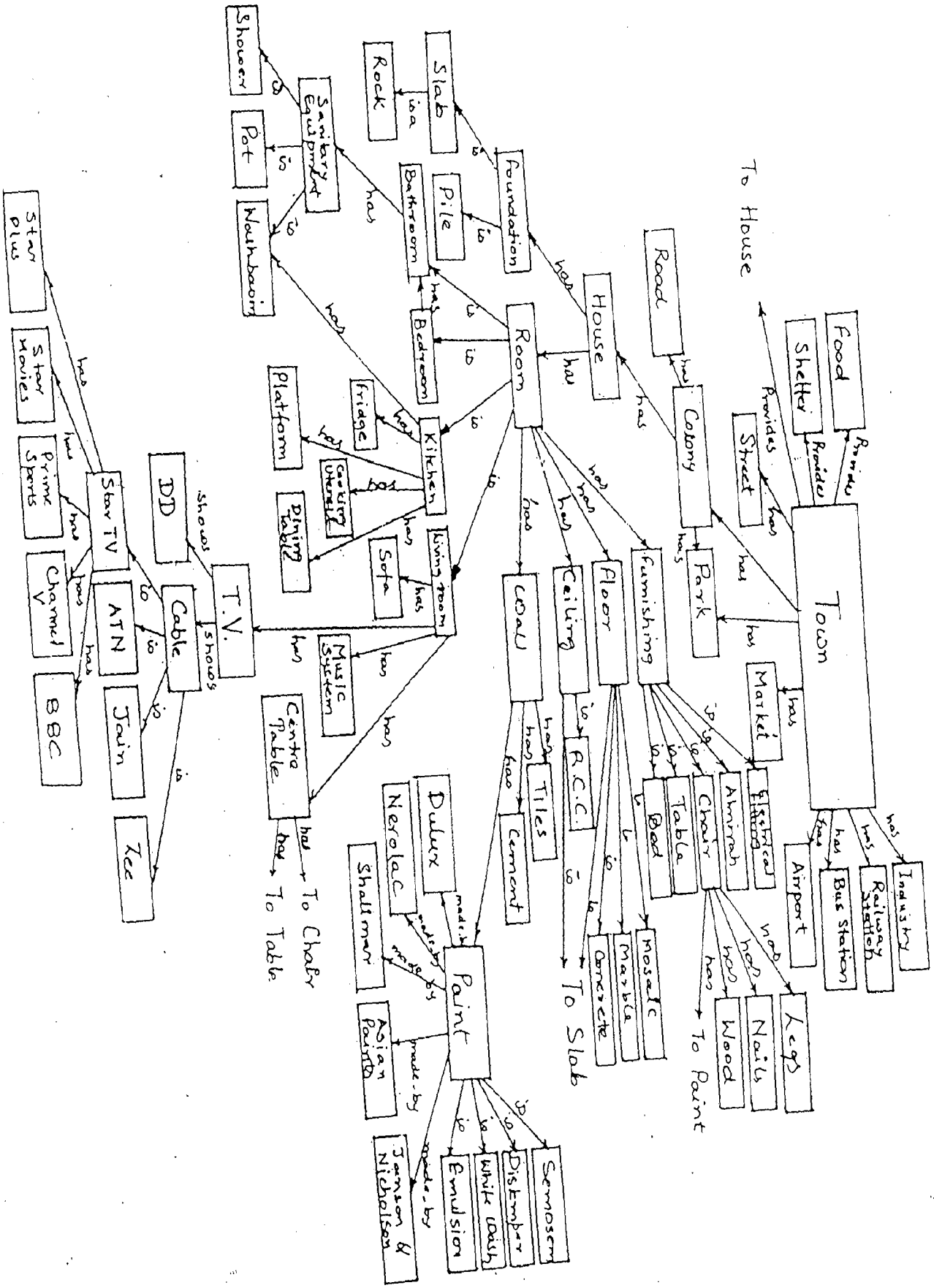


fig6.1 A SAMPLE SEMANTIC NET

```
typedef struct choice_list {
        int choice ; /* -ve choice for NOT choices */
        struct choice_list *next ;
       } SELECTION ;
```

6(a)

```
typedef struct frame {
         char *name ;
         int qualifier_no ;
         int no_childs ;
         char *childs[NO_CHILDS] ;
        } FRAME ;
```

6(b)

```
struct qual_name {
    char *name ;
    int no_links ;
    struct name_list *head ;
    long int line_gpr ;
    int no_parents ;
    int traversed ;
    struct parent_list *root ;
    struct qual_name *left, *right ;
   } ;
```

6(c).

```
struct name_tree {
    char *name ;
    int qualifier_no ;
    SELECTION *head ; /* Used when rules are fired */
    long int line_gen ;
    struct name_tree *left_name, *right_name ;
    struct name_tree *qual_left, *qual_right ;
```

```
} ;
```

6(d)

```
struct parent_list {
      char *name ;
      /* char *child_relation ;*/
      struct parent_list *next ;
     } ;
```

6(e)

```
struct name_list {
      char *name ;
      struct name_list *next ;
     } ;
```

6(f)

```
typedef struct {
  int total_texts ;
  struct name_list *tos ;
        } STACK ;
```

6(g)

```
typedef struct relations {
     char *attribute ;
     int no_left ;
     struct name_list *left_head ;
     int no_right ;
     struct name_list *right_head ;
     struct relations *left, *right ;
    } RELATIONS ;
```

6(h)

The structure 6(b) is used to store the information about a qualifier such as its name, its number (see chapter 3), the no. of attributes & name of each attribute. The information about each qualifier is stored in a file & not kept in memory. The

information where the qualifier is stored in a file and other firing session information (explained later) is stored in the structure 6(d). This thus creates an index file concept, resulting in faster data retrieval. The structure 6(c) stores the information about each object, like its name, no. of outgoing links (i.e. no of qualifiers with that object), list of names of each qualifier (using structures 6(f)), no. of incoming links (decides no. of parents) & a list of each such parent qualifier using structures 6(e). Provision has been given in structure 6(e) to incorporate the name of relationship from child to it's parent. Say if we have qualifier "House has Room", then we can also say "Room is_in House". This type of inverse relationship can also be incorporated to provide more sophisticated applications or for answering more typical queries involving the semantic net. The semantic net is simulated with the help of structures 6(b), 6(c) & 6(d) of which structures 6(c) and 6(d) constitute the binary trees. A sample simulation is shown in fig 6.2 for the semantic net of fig 2.3.

The functions involved in creating/updating/retrieving previous semantic net are :

read_from_file() retrieves the previously created semantic net from the files. This also updates the various data structures for further use/updation. This also call modules gdx_read(), link_read_gdx() to read the dissected information about a qualifier. The counterparts are w_frm_pr_ind(), w_gpr_gdx(), gpr_w(), gdx_w() & gdx_link_w().

w_frame() writes a qualifier information into a file, while read_frm() read it.

The function input_frame() gets various information about qualifiers from the user. This function call get_data() to get a single qualifier information. The function also updates the semantic net data structures by calling functions process_frame(),

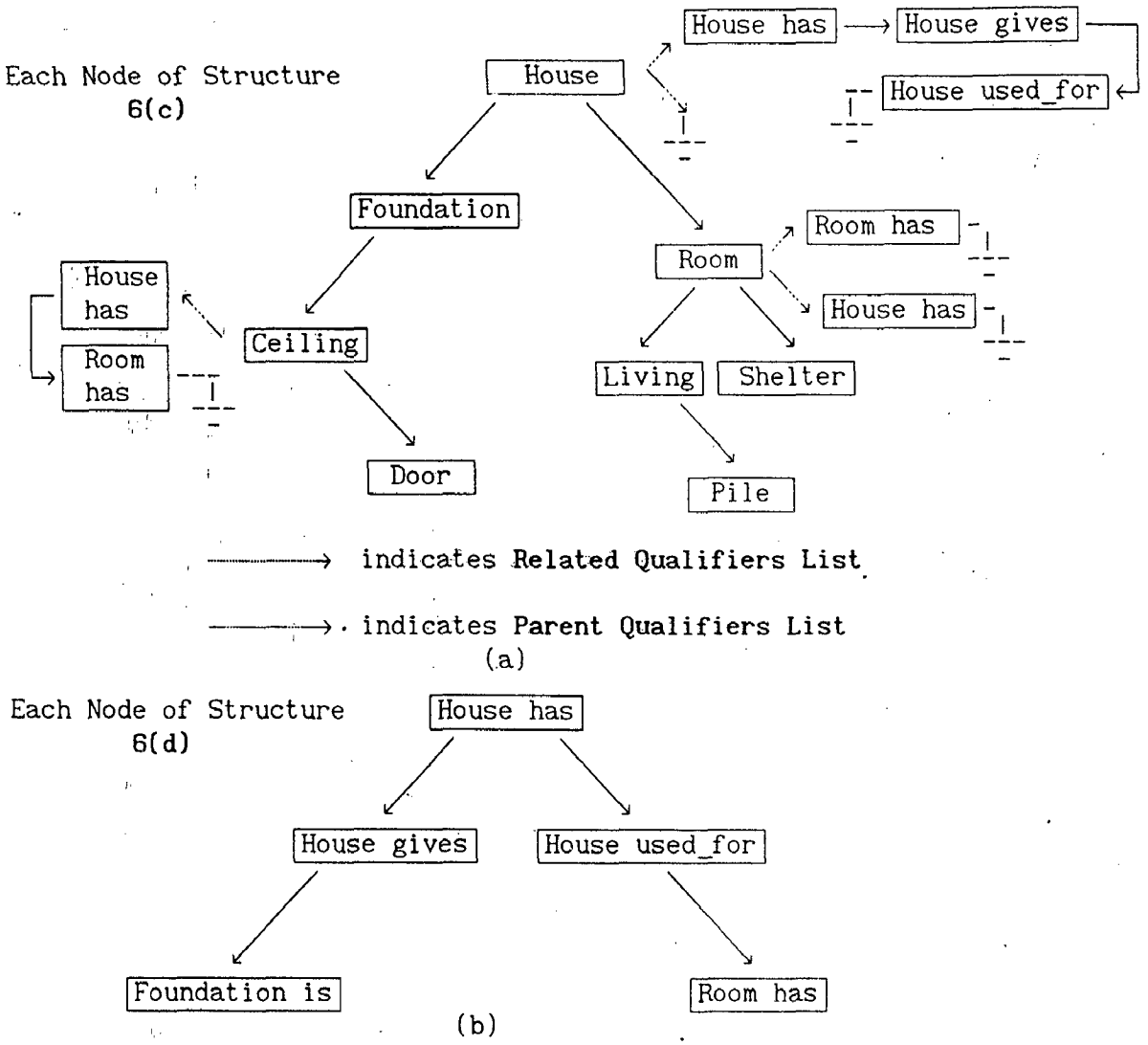


fig 6.2

INTERACTION OF TWO BINARY TREES TO CREATE A SEMANTIC NET

which in turn calls functions add_qualifier(), add_frame(), add_no_qual(), add_name_to_qualifier(), fill_relation_tree() & qual_parent_add().

These functions update the various data structures like binary trees & linked lists to update/create the semantic net.

Various functions were created to retrieve a particular information from a given data structure, for e.g. get_object() which returns a pointer to a node of structure 6(b) giving various details about an object, whose name is passed to it; get_nm_qualifier() & get_num_qualifier() which retrieve

information about a qualifier depending upon the qualifier name & number respectively; get_no_qualifier_in_tree() & get_frm_ptr() retrieve structure 6(c) depending upon the qualifier name and number respectively; etc.

Functions like disp_no_qualifier(), disp_name_qualifier(), disp_frame(), disp_attr_info() & disp_info_qualifier() called independently and also by the modules like show_qualifiers(), random_movement(), show_each_attributes() & show_super_attributes() which perform functions of displaying each qualifiers, allowing random traversal of semantic net, displaying information about each or only super objects respectively.

The functions find_in_object() & value_find() are used to answer various queries explained in chapter 3. They both use breadth first approach to perform their job. Care has been taken to prevent the search from entering into cycles in the semantic net. This approach also eliminates the consideration of the same object again when searching, thus ending the search process earlier. These modules use function get_object() to verify the presence of objects entered, so that user may be intimated about the precise error. These functions require extensive stack usage, so stack handling routines were created. These routines are used to create, initialize, pop or push an stack element & free the stack completely. The functions implementing them are get_stack(), clear_stack(), pop(), push() & free_STACK(), besides the function empty_stack(), which returns the state of the stack (empty or filled). As the structure 6(g) indicates, the stack element are string, a choice which makes the routine capable of storing data of any type from integers to double (after converting them into strings).

Various functions which exploit the use of relation to link two objects are left_list_rel() & right_list_rel() which display objects which relate to or are related to other objects by a given relation. The function disp_rel_list() displays objects which are

related to other objects by a given relation provided the name of the relation & extra information about the type of objects (i.e. whether objects which have outgoing arcs with that relation or those objects which have incoming arcs with that relation in the semantic net). The functions to read & write relation information are read_relations(), read_one_rel() and write_relations(), w_rel() & rel_w(). The functions fill_relation_tree(), add_relation(), add_left_relation() and add_right_relation() are used to add new relations. The structure used heavily in all these operations is given 6(h).

Other than the above specified specialized functions, various general purpose support functions like get_char_array() (to get an array of characters), dup_str() (to duplicate the string passed), draw() and ver_line() (to create a window and draw a vertical line respectively), etc. were created to provide modularity, fast processing, efficient memory usage & above all reduced code length.

(b) Non learning rule based inferencing module :

This module as already explained in chapter 3 uses production rules in the form of if - then statements. The formation of rules was explained there, here the internals i.e. functions and data structures involved in performing the job are explained.

```
typedef struct {
 char *qual_name ;
 int no_choices ;
 struct choice_list *head ;
      } QUALIFIER ;
```

6(i)

```
typedef struct method {
        char *method ;
        int probability ;
```

```
        float result_probability ;
        struct method *next ;
        struct method *same_next ;
       } CHOICE ;
```

6(j)

```
typedef struct {
                int var_expr ;
  char *expression ;
  char *left_part ;
  char *right_part ;
  char relation [3] ; /* to accomodate "<>" */
       } EXPRESSION ;
```

6(k)

```
typedef struct choice_list {
       int choice ; /* -ve choice for NOT choices */
       struct choice_list *next ;
      } SELECTION ;
```

6(l)

```
typedef struct progs {
        char *name ;
        int share_count ;
        int pass_no ;
        struct name_list *p_head ;
        int return_no ;
        struct name_list *r_head ;
        struct progs *next ;
       } EXTPROG ;
```

6(m)

```
union qual_math {
   QUALIFIER *ql ;
   EXPRESSION *expr ;
  } ;
```

6(n)

```
union choice_extprog_qual_math {
    CHOICE *ch ;
    EXTPROG *extprog ;
    QUALIFIER *ql ;
    EXPRESSION *expr ;
          } ;
```

6(o)

```
typedef struct ifs {
      int fired ;
      int act_rule_no ;
      int which ;
      union qual_math qm ;
      struct ifs *next ;
      struct ifs *left, *right ;
      struct ifs *same_next ;
     } IF ;
```

6(p)

```
typedef struct thens {
      int fired ;
                  int act_rule_no ;
      int which ;
      union choice_extprog_qual_math ceqm ;
      struct thens *next ;
      struct thens *left, *right ;
      struct thens *same_next ;
     } THEN ;
```

6(q)

```
typedef struct rule {
       int fire_rule_no ;
       int act_rule_no ;
```

```
        int no_ifs ;
        int tot_ifs_fired ;
        IF *rule_if ;
        int no_thens ;
        int tot_thens_fired ;
        THEN *rule_then ;
        int fired ;
        struct rule *next, *prev ;
       } RULE ;
```

6(r)

```
typedef struct var_desc {
    char *name ;
    char *txt ;
    double value ;
    int initialized ;
    int displayed ;
    int val_accepted ;
    int disp_in_res ;
    struct var_desc *left, *right ;
   } VARIABLE ;
```

6(s)

```
typedef struct res_stack {
     int act_rule_no ;
     struct res_stack *next ;
    } ELEMENT_RULE_STACK ;
```

6(t)

```
typedef struct {
  ELEMENT_RULE_STACK *tos, *curr_why ;
      } RULE_STACK ;
```

6(u)

The structure 6(r) stores information about a rule like rule no., actual (fixed) and fire_rule_no (to provide provision for future changes in the ordering of rules), no. of 'IF' condition and their list (a linked list of structure 6(p)), no of 'THEN' condition and their list (a linked list of structure 6(q)) and the state of a rule i.e. whether fired or not (used at the time when inferencing session is going on). The rules are connected to each other by a doubly linked list. The structure of 'IF' and 'THEN' conditions have sub-structures which indicate the type of if and then condition viz. qualifier instance, mathematical expressions, external programs, or a text associated with some probability in the range of 0 to 10. The structures 6(i) to 6(o) provide means to represent each of the above conditions.

```
#1
   IF
     Building is tall
   THEN
     Use deep foundation - 7/10

#2
   IF
     [no_storey] > 3
   THEN
     Building is tall
Qualifier used
#1 Building is
     1. tall
     2. normal
Variable [no_storey] indicates no. of floors in the building.
```

fig 6.3

A SAMPLE RULE INITIATING BACKWARD CHAINING

The default control strategy to fire the rules is forward chaining, but the rules can be framed in such a way that they produce a goal and thus triggering a chain of rules which may lead to the goal. An example of such a rule is given in fig 6.3. As seen in fig 6.3, the default method of firing rules will first try to solve rule #1, but this rule requires a fact about the type of building to be built i.e. if "building is tall", now the IE will search for a rule that will tell it about the state of building and so it sets a goal of finding the state of building. It finds rule #2, which has in its consequence the state of building, thus this rule is fired to find the state of the building. Thus we have implemented an example of backward chaining. The IE's control strategy is thus a hybrid of forward and backward chaining.

Explanation module for answering "WHY" is implemented by using the rule stack and reciting those rules along with their various premises which are presently waiting to be solved. The "HOW" facility is implemented in a crude form by reciting the rules which were successful and ones which were not(along with their first premise which failed) . The rule stack was created using structures 6(t) and 6(u). The field "tot_ifs_fired" of each rule structure is used to determine the state of rule; fired completely, fired successfully, was unsuccessful, or is currently under fire, by assigning special values.

What follows is the description and names of the important functions used to implement this module.

new_get_rules() is used to input new rules from the user. This function in turn calls two functions get_ifs() and get_thens() to form a complete rule. Function if_single_get() and then_single_get() get a single premise and consequence

respectively. The rules acquired are added to the list of rules by function add_rule_list(). Within the function if_single_get() and then_single_get() code is written to accept the premise or consequence of any type viz. qualifier, mathematical expression, external program or a text associated with a probability.

Functions which keep track of type and list of each type of premises and consequences are if_new_rule_add(), each_if_add(), if_q_add_tree() and if_m_add_tree() with corresponding functions for the consequences of each type.

Functions to store the rule base are rules_write(), one_rule_write(), ifs_write(), one_if_write(), QUL_write(), EPR_write(), CHE_write() and MATH_write(). Corresponding functions exist to retrieve the rule base with main function being read_rules(). This function sets the various data structures and global variables to the state in which the last session ended.

Function disp_rule(), disp_ifs(), disp_thens(), disp_choice(), disp_all_methods() and ext_all_disp() used independently and in other functions display various informations about the rules entered or part of rules.

Various house keeping functions like discard_rule(), if_discard(), etc. were created to use memory efficiently.

The function fire_rules() form the IE, calling function single_rule_fire() which fires the rules provided to it. This function calls the function true_if() to test the status of the premise. A rule is fired until either a premise turns out to be false, its premise requires a fact obtainable from other rule (indicated by functions can_fire() and can_var_fire()) or all of its premises are true. In the last case the function fire_thens() is called by single_rule_fire(). The results prepared by this function are added to the result list by function fire_CHE_then() and update_var_in_res(). Function true_qualifier() checks the premise of a qualifier type for truth or false. This uses the field 'head' of structure 6(d) to deduce its results.

The 'WHY' facility if provided by the function why_answer() which displays the rules presently in the rule stack. This uses various rule stack manipulation function viz. rule_empty_stack(), init_rule_stack(), prepare_for_why(), why_rule_retrieve_stack(), rule_pop_stack() and rule_push_stack().

The functions which implement the 'HOW' facility are successful_rules_view() and fail_rule_detail(),which use the field 'tot_if_fired' of each rule structure to provide the answers. No extra information is generated to explain the user its line of reasoning for reaching a goal. The results are printed by the module res_print(), called both independently and automatically after the fire_rules() function.

Various functions were created to parse and solve complex mathematical expressions using complex mathematical operators, functions. The variables can also be used to facilitate formation of complex rules. The parser of the expression was implemented using recursive decent parsing principle [20, 21]. The grammar [22] for this application was specially designed and implemented. The function which perform parsing also used to solve an expression, thus saving code. The backward chaining can also be triggered if the value of a variable can be deduced from some rule's consequence.

The external program link is performed by spawning a new process and parameter passing is with the help of files in terms of list of real numbers. Parameters are parsed by file with extension 'PAS' and the parameters are received from a file by a file with the programs name with extension 'RET'. The values of the variables to be passed and if they are not available at that moment, than rules which initialize those variables are fired, triggering backward chaining.

(c) Learning rule based Inferencing module :

This module like previous rules uses production rules in the form

of if-then statements to provide inferences. It differs from the learning based module in terms of the type of premises and consequences used by it to form a rule (see fig 3.12). It uses simple structures to represent its premises, consequences and rules.

```
typedef struct if_and_then {
        int left_part ; /* VARIABLE or OBJECT */
        char *left_object ;
        int right_part ;
        char *right_object ;
        char *attribute ;
        struct if_and_then *next, *prev ;
       } IF_THEN ;
```

6(v)

```
typedef struct rule {
        int no_ifs ;
        int tot_ifs_fired ;
        IF_THEN *rule_if ;
        int no_thens ;
        int tot_thens_fired ;
        IF_THEN *rule_then ;
        struct rule *next, *prev ;
       } RULE ;
```

6(w)

```
typedef struct vars_in_rules {
          char *name ;
          char *object ;
          struct vars_in_rules *next ;
         } RULE_VARS ;
```

6(x)

The structure 6(v) is used to represent both premises and consequences because of their similar nature. The rule structure is very much similar to the one in the non-learning module's rule structure. The structure for storing premises or conditions contain information about the type of left part (variable or object name), its name, the type of right part (again variable or object name), its name, the relation relating them. The premises and consequences of a rule are linked by doubly linked lists and so are rules. Structure 6(x) is provided for storing information about the variables at run time. The various such structures are present in a alphabetically sorted list. Each such structure contains variable name and the object name it is assigned for that firing session.

The function names and description is presented below which were used to implement this module.

Rules are acquired by function get_rules() which calls function get_if_thens() to get various premises and consequences of each rule.

To view rules function view_rules() is called which displays all the rules in the rule base (both permanent and the ones which are created when system learns some thing). The individual rules are displayed by the function disp_rule() or no_disp_rule(). Rules are saved explicitly by the user for the reason explained in chapter 3 . The function rules_write() is involved for this purpose. The rules are jread automatically when this module is invoked. The function read_rules() reads the rules, initializes and sets up various data structures and variables. This module is called in the function initialize() which initializes all the data structures to the state in which last session ended, opens all the necessary files and rads rules and the semantic net information by invoking module read_from_file().

The decision process starts by invocation of the function fire_rules() which fires rules sequentially from first to last

rule. Backward chaining is not incorporated, so all facts have to presented , when the rules are fired, specifically when a certain premise is containing a variable is fired, it should be known what object is assigned to it. This fact if not previously told, has to be provided. The system displays a list of probable objects vying for that variable. The rule which is currently under fire is displayed to help user input appropriate information. The user upon having this information updates the variable information using add_var(). The function solve_premise() is used to test a premise, which uses the semantic net to test that whether two objects in a premise are linked to each other by the relation in the premise. The rule is stopped firing, when either one of its premise turns out to false or all of the premise prove to be true, triggering consequences and ESS providing decisions. The results are displayed(function result_display())as the rules become successful and confirmation is taken from the user. If the result is confirmed by the user, the semantic net is updated (by function update_semantic_net()) by adding new qualifiers as given by the consequences of the rule. The history is saved by saving the current status of the each variable, which can be recalled if any conflict takes place in decision provided by the system & user's choice. If the user is not satisfied by the ESS decision, he may discard the ESS decision, leading to a new learning session. The ESS first tries to find the reason why the conflict has occurred & for this it compares the current semantic net with the net which existed when last time the rules were fired successfully. It then displays all the differences encountered as a suggestion to create a new rule. The user can then either select an explanation offered to him or can himself suggest a new reason. The learning session ends there, rules are duplicated & modified in conformance with the new situation. The rules are fired again to arrive at the decision according to the modified rules & the decisions made again are put before user for confirmation & the cycle repeats for

all the decisions during the session.

The functions find_differences() & var_differences_find() are used to find the difference between the successful case semantic net and the present case. The stack is used to save the differences. The explanations are retrieved and displayed from the stack by the function disp_explanations(). The explanation accepted by the user is used to create premises, a job done by the function get_complementary_premises(). Rules are duplicated by the function dup_RULE() & new premises added to both. New consequences are provided by the user for the new situation. If he does not provide one, the previous consequences are complemented for e.g. if previous consequence was "[x] has floor", then the complemented consequence will be "[x] has NOT floor". The complementation of a consequence is done by calling the function all_reverse_consequences().

The 'HOW' facility like previous inferencing module is implemented by displaying successful and unsuccessful premises & by displaying the list of variables, along with the objects they are assigned. This function is provided by successful_rule_view() and fail_rule_detail() & res_print().

Global variables use was kept to a minimum, but was not avoidable due to overhead in passing parameters where recursion is a basic method of reaching a solution. The global variables were used mainly to represent certain flags used in many functions, pointers to data structures used in many function, represent the state of the system in terms of no. of rule, no. of qualifiers entered, etc..

The shell has been implemented on Intel's X86 platform using Turbo C version 2.0. The source code was written in several files. The project facility of Turbo C was used to compile and link all the files. The large memory model was chosen because of large code size (some 15000 lines) and large run time RAM

requirements. Various files are created (some 7 files) to store the information. Steps to minimize the no. of files is discussed in 7.2 (Suggestion for future work). Backup file "RBK" is created everytime rule base is updated, so that accidental updation of rule base does not lead one with incorrect rules permanently.

CHAPTER 7

CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

7.1 CONCLUSION

The work carried is used as central module for a Department of Electronics (DOE) government of INDIA project titled "Integrated Knowledge Based System for CAD applications". Results of the various rules and problems posed to it in chapter 4 & 5 clearly show that it can be used satisfactorily for Architectural and Building Design problems. The inferencing is fast justifying the choice of HLL 'C'. Choice for using the rule based model along with a semantic net for fact storage and use of both learning and non-learning inferencing modules communicating via semantic net proved right as a wide variety of problems were successfully solved. The developed can also be used for creating ES for domains other than Architecture and Building construction as shown by the rule base of fig 7.1, which is the rule base for an ES for medical domain. SHELL thus is a general tool for developing ESs for any domain, with facilities and features incorporated which ease the creation of ESs for Architectural and Building Design. As SHELL is written in HLL language, it can be easily ported to any platform which has support for the language 'C'.

```
#1
   IF
     Person is sick
     Doctor is unavailable
   THEN
     Give him medicine - 10/10
     Call him again in the morning - 10/10
```

#2

IF

Face is green

THEN

Person is sick

#3

IF

Temperature above 100 degree

THEN

Person is sick

#4

IF

Hour is late

THEN

Doctor is unavailable

Qualifiers used :

#1 Person is

1. sick
2. well

#2 Face is

1. green
2. **red**

#3 Temperature is

1. above 100 degrees
2. below 100 degrees

#4 Hour is

1. late
2. early

#5 Doctor is

1. available

2. unavailable

fig 7.1

A SAMPLE RULE BASE FOR A HOSPITAL

7.2 SUGGESTIONS FOR FUTURE WORK

The shell at its present stage although is able to satisfy the purpose of its creation, still leaves some space for future enhancements. Also in a work of this magnitude scope for system enhancement in terms of versatility and convenience is limitless. Some of the major directions in which work can be carried out in future are given below.

(i) Learning module can be integrated with a CAD tool, so that user interaction is limited to making changes, giving confirmation & teaching the system method to handle new situations. User should no longer indicate to the system explicitly which instance of the objects he is presently interested in. Other approach may be to limit the no. of choices by looking ahead and trying all the combinations which fit that variable in all the rules, for e.g. in asking for choices for variable "[x]", one should list only the various instances of door like "D1", "D2", etc..

(ii) The Learning module in its present state, simulates learning by adding new premises. This is good, but provision should be there to suppress previously entered not so important premises.

(iii) The two inferencing modules at present share the semantic net, future works could try using the two modules in tandem to build an even more powerful system.

(iv) The shell can be implemented on a LAN or a distributed environment, so that many people can use it concurrently and perform their jobs concurrently. For e.g. one person may be creating the foundation, one may be analyzing the foundation built, one may be designing interior at the same time, speeding up the design process.

(v) The explanation facility 'HOW' could be refined more & made more versatile, to make the user understand the line of reasoning of SHELL.

(vi) Knowledge acquisition module using inductive learning can be added to ease creation of KB and allow smooth transfer of knowledge from the domain expert to the system.

(vii) Facilities to provide more information about the KB, like indicating rules which contain a certain qualifier or variable or a method etc. could be incorporated helping the user in debugging and refining process of the rules.

(viii) Facility to rerun the system, by changing certain facts at the end should be provided so that, user may test and analyze the result for change in certain facts. For e.g. the change in cost if a 3 storey building is created instead of a 2 storey building with remaining facts being same.

(ix) Facilities to edit present rules i.e. add or delete premises to a rule, delete or move a rule, etc. should be incorporated.

(x) A tutoring module should be included to teach a novice how to use the shell efficiently, create rules and perform debugging. This is a very module as writing rules is equivalent to writing a program in any language.

(xi) Online help should be provided to make the system more convenient to use.

(xii) The application can be ported as an windows application, having menu, mouse support, etc.

(xiii) The source code written uses many files. Some of the files like the files with extension "GDX", "GPR", "REL" can be done away with, as information contained in them can be regenerated at some run time over head, by using the contents of file with extension "GEN".

(xiv) Efforts can be made to get facts from the user in a

graphical way. Each premise can trigger display of various choices & user can select one or more by using a mouse or some other pointing device.

With the source code available one can make efforts to implement the above suggestions. Modifications may also be made to the source code to customize SHELL according to ones taste. One can make efforts to remove redundancies in the code to make it more compact and faster. The modular nature of the code will make the job of the programmer much easier in achieving these changes.

REFERENCES

[1] Flemming U., "Knowledge Representation and Acquisition in the LOOS system", Building and Environment, Pergamon Press, Oxford, 1990, vol. 25, NO. 3, pp 209-219.

[2] Bedard C. and Gowri K., "Automating Building Design Process with KBES", Journal of computing in Civil Engineering, ASCE Publication, vol. 4, No. 2, April 1990.

[3] Pohl J. and Chapman A., "An Expert Design Generator", Architectural Science Review, vol. 31, 1988, pp 75-86.

[4] Fenves S.J. et.al., "An Integrated Software Environment For Building Design and Construction", Proc. of the 5th Conference on Computing in Civil Engineering, ASCE, pp 21-32.

[5] Morese D.V. and Hendrickson C., "Model for Communication in Automated Interactive Engineering Design", Journal of Computing in Civil engineering, vol. 5, No. 1, Jan 1991, pp 4-24.

[6] Carrara G. and Novemberi G., "Knowledge Acquisition in the Process of Architectural Design", Building and Environment, Pergamon Press, Oxford, 1990, vol. 25, No. 3, pp 199-207.

[7] Rosenman M.A., "Applications of Expert Systems to Building Design and Evaluation", Building and Environment, Pergamon Press, Oxford, 1990, vol. 25, No. 3, pp 209-219.

[8] Evans P.M., "Rule-based applications for Checking Standards Compliance of Structural members", Building and Environment, Pergamon Press, Oxford, 1990, vol. 25, No. 3, pp 209-219.

[9] Garret J.H., "Application of Knowledge-based System techniques to standards representation and usage", Building and Environment, Pergamon Press, Oxford, 1990, vol. 25, No. 3, pp 209-219.

[10] Chang T.C. and Ibbs C.W., "PRIORITY RANKING - A Fuzzy Expert System for Priority Decision Making in Building Construction Resource Scheduling", Building and Environment, Pergamon Press, Oxford, 1990, vol. 25, No. 3, pp 209-219.

[11] Mohan S., "Expert Systems Applications in Construction Management and Engineering", Journal of Construction Engineering and Management, ASCE publication, vol. 116, No. 1, 1990, pp 87-99.
[12] Guena F. and Zreik K, "An Architect Assisted Architectural Design System", Artificial Intelligence in Design, Computational Mechanics Publications, pp 159-179.
[13] Tean-Jee Lee et. al., "Expert Systems for Buildings - Case Studies of Design, Diagnosis, Contracts Interpretation", Architectural Science Review, 1994, vol. 37, No. 1, March 1994, pp 21-34.
[14] Firebaugh M., ARTIFICIAL INTELLIGENCE - A KNOWLEDGE BASED APPROACH, Pws-Kent Publishing Co., Boston, 1988.
[15] Forsyth R., "The Expert System Phenomenon", Expert Systems - Principles and Case Studies, Chapman and Hall Computing, 2 ed., pp 3-21.
[16] Forsyth R., "Inductive Learning for Expert Systems", Expert Systems - Principles and Case Studies, Chapman and Hall Computing, 2 ed., pp 197-221.
[17] Waterman D.A., A GUIDE TO EXPERT SYSTEMS, Addison-Wesley Publishing Company, USA, 1986.
[18] Rich E., ARTIFICIAL INTELLIGENCE, McGraw-Hill Company, 1983.
[19] Schutzer D.A., ARTIFICIAL INTELLIGENCE - AN APPLICATION ORIENTED APPROACH", Van Nostrand Reinhold Company - NEW YORK, 1982, pp 23-36.
[20] Aho A.V. Ullman J.D., PRINCIPLES OF COMPILER DESIGN, Narosa Publishing House, 1985.
[21] Barret et.al., COMPILER CONSTRUCTION THEORY AND PRACTICE, Galgotia Publications Pvt. Ltd, 2 ed..
[22] Denning P.J., Dennis J.B. and Qualitz J.E., MACHINES, LANGUAGES AND COMPUTATION, Prentice Hall Inc. - NEW JERSEY, 1978.
[23] Korth H.F. and Silbershatz A., DATABASE SYSTEM CONCEPTS, McGraw Hill Inc, 2 ed.
[24] Kernighan B.W. and Ritchie D.M., THE C PROGRAMMING LANGUAGE,

PHI Pvt. Ltd., 2 ed..

[25] Kruse R.L., DATA STRUCTURES AND PROGRAM DESIGN, PHI Pvt. Ltd., 2 ed..

[26] Schild H., TURBO C/C++ - THE COMPLETE REFERENCE, Osborne/McGraw Hill, 2 ed.

[27] Gottfried B.S., PROGRAMMING WITH C, Schaum Outline Series, Tata McGraw Hill Publication Company Limited, First edition.

[28] Schild H., C - THE COMPLETE REFERENCE, Osborne/Mcgraw Hill, First edition.

```
/******* KEYS.h *******/


#ifndef KEYS_INCLUDED
#define KEYS_INCLUDED
  /* Various Definitions used */
#define ESC '\x1b'
#define ENTER '\x0a'
#define LEFT_ARR 0x4b
#define RIGHT_ARR 0x4d
#define UP_ARR 0x48
#define DOWN_ARR 0x50
#define YES 1
#define NO 0
#define EXTENDED 0
#define BACKSPACE 0x08
#define TRUE 1
#define FALSE 0
#define BEGIN 1
#define END 0
#define NO_CHILDS 20
#define MAX_LENGTH 500
#define CHLD 1
#define PARENT 2
#define PRNT_CHILD 3
#define PART_PRNT 4
#define PART_CHILD 5
#define MESG_LENGTH 150
#define MATH_ERROR_STARTING 1000
enum {VARIABLE, OBJECT} ;
enum {IF, THEN} ;
enum {LEFT, RIGHT} ;
enum {QUL, MATH, EPR, CHE} ;
enum {PARSING, SOLVING, RES_PREP} ;
enum {VAR, EXPR} ;
enum {SIN=1, COS, TAN, ASIN, ACOS, ATAN, ABS, FLOOR, CEIL, SQRT, EXP,
LOG}  ;
enum {DELIMITER, NUMBER, VAR_FUNC} ;
#endif
```

```
/****** SHELL.H ******/


#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <alloc.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <math.h>
#include <errno.h>
#include <setjmp.h>
#include <io.h>
#include "keys.h"
#include "rules.h"
#include "semantic.h"
```

```
/*****/SEMANTIC.H ******/


typedef struct choice_list {
                            int choice ; /* -ve choice for NOT choices
 */
                            struct choice_list *next ;
                           } SELECTION ;
typedef struct frame {                  /* for .GEN file */
                      char *name ;
                      int qualifier_no ;
                      int no_childs ;
                      char *childs[NO_CHILDS] ;
                     } FRAME ;
struct qual_name {
                  char *name ;
                  int no_links ;
                  struct name_list *head ;
                  long int line_gpr ;
                  int no_parents ;
                  int traversed ;
                  struct parent_list *root ;
                  struct qual_name *left, *right ;
                 } ;
struct name_tree {
                  char *name ;
                  int qualifier_no ;
                  SELECTION *head ;
                  long int line_gen ;
                  struct name_tree *left_name, *right_name ;
                  struct name_tree *qual_left, *qual_right ;
                 } ;
struct parent_list {
                    char *name ;
                    /* char *child_relation ;*/
                    struct parent_list *next ;
                   } ;
struct name_list {
                  char *name ;
                  struct name_list *next ;
                 } ;
typedef struct {
                int total_texts ;
                struct name_list *tos ;
               } STACK ;
typedef struct relations {
                          char *attribute ;
                          int no_left ;
                          struct name_list *left_head ;
                          int no_right ;
                          struct name_list *right_head ;
                          struct relations *left, *right ;
                         } RELATIONS ;
```

```
/****** IN_SCR.C ******/


#include "shell.h"
/* In Other Files */
struct name_list *get_link_node(void) ;
void list_dispose(struct name_list *) ;
/* In This File */
char *get_ch_txt(int, int, int, int) ;
char *get_val_var_txt(int, int, int, int) ;
char *get_prob_txt(int, int, int, int) ;
char *input_choices(int, int, int, int) ;
void draw(int,int,int,int ) ;
void status_window(char *) ;
void ver_line(int,int,int) ;
char *get_char_array(int) ;
void clear_windows(void) ;
char *trim(char *) ;
char *dup_str(char *) ;
STACK *get_stack(void) ;
void push(char *, STACK *) ;
char *pop(STACK *) ;
int empty_stack(STACK *) ;
void clear_stack(STACK *) ;
void free_STACK(STACK *) ;
char *get_ch_txt(int x1, int y1, int x2, int y2)
{
 struct text_info ti ;
 char *txt = NULL, *ret_txt = NULL ;
 int response = TRUE, index = 0 ;
 int key, cur_x = 1, cur_y = 1 ;
 int maxind = (x2 * y2) - 1 ;
 gettextinfo(&ti) ;
 --x1, --y1 ;
 x1 += ti.winleft ;
 y1 += ti.wintop ;
 txt = get_char_array(maxind) ;
 textattr(BLACK | (CYAN << 4)) ;
 window(x1,y1,x2+x1-1,y2+y1-1) ;
 clrscr() ;
 txt[0] = '\0' ;
 while(TRUE)
      {
       key = getch() ;
       if(isprint(key))
         {
          response = TRUE ;
          if(index == maxind)
            response = FALSE ;
          if(response == FALSE)
            continue ;
          else
              {
               txt[index++] = key ;
               txt[index] = '\0' ;
               gotoxy(cur_x,cur_y) ;
               cprintf("%c",key) ;
               cur_x = wherex() ;
               cur_y = wherey() ;
```

```
                }
            }
        else
            {
             switch(key)
                    {
                     case ENTER : if(index != 0)
                                    goto finish ;
                     case ESC   : free(txt) ;
                                  txt = NULL ;
                                  goto finish ;
                     case EXTENDED  : key = getch() ;
                                      if(key != LEFT_ARR)
                                        break ;
                     case BACKSPACE : if(index != 0)
                                      {
                                       --index ;
                                       txt[index] = '\0' ;
                                       cur_x = (index % x2) + 1 ;
                                       cur_y = (index / x2) + 1 ;
                                       gotoxy(cur_x, cur_y) ;
                                       cprintf(" ") ;
                                       gotoxy(cur_x, cur_y) ;
                                      }
                                     break ;
                    } /* switch(key) */
            } /* else */
       } /* while(TRUE) */
 finish :
 window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
 textattr(ti.attribute) ;
 gotoxy(ti.curx, ti.cury) ;
 if(txt == NULL)
   return NULL ;
 else
     {
      ret_txt = dup_str(txt) ;
      free(txt) ;
      txt = NULL ;
      return ret_txt ;
     }
}
char *get_val_var_txt(int x1, int y1, int x2, int y2)
{
 struct text_info ti ;
 char *txt = NULL, *ret_txt = NULL ;
 int decimal_pt = NO ;
 int response = TRUE, index = 0 ;
 int key, cur_x = 1, cur_y = 1 ;
 int maxind = (x2 * y2) -1 ;
 gettextinfo(&ti) ;
 --x1, --y1 ;
 x1 += ti.winleft ;
 y1 += ti.wintop ;
 txt = get_char_array(maxind) ;
 textattr(BLACK | (BROWN << 4)) ;
 window(x1,y1,x2+x1-1,y2+y1-1) ;
 clrscr() ;
 txt[0] = '\0' ;
```

```
while(TRUE)
     {
      key = getch() ;
      if(key == '.' || key == '+' || key == '-' || isdigit(key))
        {
         response = TRUE ;
         if(index == maxind)
           response = FALSE ;
         else if(index > 0 && (key == '+' || key == '-'))
                 response = FALSE ;
              else if(key == '.' && decimal_pt == YES)
                      response = FALSE ;
         if(response == FALSE)
           continue ;
         else
             {
              if(key == '.')
                decimal_pt = YES ;
              txt[index++] = key ;
              txt[index] = '\0' ;
              gotoxy(cur_x, cur_y) ;
              cprintf("%c",key) ;
              cur_x = wherex() ;
              cur_y = wherey() ;
             }
        }
      else
         {
           switch(key)
                {
                 case ENTER : if(index != 0)
                                goto finish ;
                 case ESC   : free(txt) ;
                              txt = NULL ;
                              goto finish ;
                 case EXTENDED  : key = getch() ;
                                  if(key != LEFT_ARR)
                                    break ;
                 case BACKSPACE : if(index != 0)
                                    {
                                     --index ;
                                     if(txt[index] == '.')
                                       decimal_pt = NO ;
                                     txt[index] = '\0' ;
                                     cur_x = (index % x2) + 1 ;
                                     cur_y = (index / x2) + 1 ;
                                     gotoxy(cur_x, cur_y) ;
                                     cprintf(" ") ;
                                     gotoxy(cur_x, cur_y) ;
                                    }
                                  break ;
                } /* switch(key) */
         } /* else */
     } /* while(TRUE) */
finish :
window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
textattr(ti.attribute) ;
gotoxy(ti.curx, ti.cury) ;
if(txt == NULL)
```

```
        return NULL ;
    else
        {
         ret_txt = dup_str(txt) ;
         free(txt) ;
         txt = NULL ;
         return ret_txt ;
        }
}
char *get_prop_txt(int x1, int y1, int x2, int y2)
{
 struct text_info ti ;
 char *txt = NULL, *ret_txt = NULL ;
 int response = TRUE, index = 0 ;
 int key, cur_x = 1, cur_y = 1 ;
 int maxind = (x2 * y2) -1 ;
 gettextinfo(&ti) ;
 --x1, --y1 ;
 x1 += ti.winleft ;
 y1 += ti.wintop ;
 txt = get_char_array(maxind) ;
 textattr(WHITE | (LIGHTGRAY << 4)) ;
 window(x1,y1,x2+x1-1,y2+y1-1) ;
 clrscr() ;
 txt[0] = '\0' ;
 while(TRUE)
       {
        key = getch() ;
        if(isdigit(key))
          {
           response = TRUE ;
           if(index == maxind)
             response = FALSE ;
           if(response == FALSE)
             continue
           else
               {
                txt[index++] = key ;
                txt[index] = '\0' ;
                gotoxy(cur_x, cur_y) ;
                cprintf("%c",key) ;
                cur_x = wherex() ;
                cur_y = wherey() ;
               }
          }
        else
            {
             switch(key)
                   {
                    case ENTER : if(index != 0)
                                   goto finish ;
                    case ESC   : free(txt) ;
                                 txt = NULL ;
                                 goto finish ;
                    case EXTENDED  : key = getch() ;
                                     if(key != LEFT_ARR)
                                       break ;
                    case BACKSPACE : if(index != 0)
                                       {
```

```
                                        --index ;
                                        txt[index] = '\0' ;
                                        cur_x = (index % x2) + 1 ;
                                        cur_y = (index / x2) + 1 ;
                                        gotoxy(cur_x, cur_y) ;
                                        cprintf(" ") ;
                                        gotoxy(cur_x, cur_y) ;
                                      }
                                    break ;
                    } /* switch(key) */
              } /* else */
        } /* while(TRUE) */
 finish :
 window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
 textattr(ti.attribute) ;
 gotoxy(ti.curx, ti.cury) ;
 if(txt == NULL)
   return NULL ;
 else
     {
      ret_txt = dup_str(txt) ;
      free(txt) ;
      txt = NULL ;
      return ret_txt ;
     }
}
char *input_choices(int x1, int y1, int x2, int y2)
{
 struct text_info ti ;
 char *txt = NULL, *ret_txt = NULL ;
 int prev = 0 ;
 int response = TRUE, index = 0 ;
 int key, cur_x = 1, cur_y = 1 ;
 int maxind = (x2 * y2) -1 ;
 gettextinfo(&ti) ;
 --x1, --y1 ;
 x1 += ti.winleft ;
 y1 += ti.wintop ;
 txt = get_char_array(maxind) ;
 textcolor(WHITE) ;
 textbackground(BROWN) ;
 window(x1,y1,x2+x1-1,y2+y1-1) ;
 clrscr() ;
 txt[0] = '\0' ;
 while(TRUE)
      {
       key = toupper(getch()) ;
       if(key == 'N' || key == ' ' || key == ',' || isdigit(key))
         {
          response = TRUE ;
          if(index == maxind)
            response = FALSE ;
          else if(index > 0 && key == 'N')
                 response = FALSE ;
               else if(index == 0 && (key == ' ' || key == ','))
                      response = FALSE ;
                    else if((key == ' ' || key == ',') && (txt[index-1
] == ' ' || txt[index-1] == ',' || txt[index-1] == '+'))
                           response = FALSE ;
```

```
        if(response == FALSE)
          continue ;
        else
            {
             if(key == 'N')
               {
                txt[0] = 'N' ;
                txt[1] = 'O' ;
                txt[2] = 'T' ;
                txt[3] = '+' ;
                prev = 0 ;
                index = 4 ;
               }
             else
                 {
                  txt[index] = key ;
                  prev = index++ ;
                 }
             txt[index] = '\0' ;
             gotoxy(cur_x, cur_y) ;
             cprintf(txt+prev) ;
             cur_x = wherex() ;
             cur_y = wherey() ;
            }
      }
    else
        {
         switch(key)
               {
                case ENTER : if(index != 0)
                               goto finish ;
                case ESC   : free(txt) ;
                             txt = NULL ;
                             goto finish ;
                case EXTENDED  : key = getch() ;
                                 if(key != LEFT_ARR)
                                   break ;
                case BACKSPACE : if(index != 0)
                                   {
                                    --index ;
                                    if(txt[index] == '+')
                                      {
                                       txt[0] = txt[1] = txt[2] = t
xt[3] = ' ' ;

                                       txt[4] = '\0' ;
                                       index = prev = 0 ;
                                      }
                                    else
                                        {
                                         prev = index ;
                                         txt[prev] = ' ' ;
                                        }
                                    cur_x = (index % x2) + 1 ;
                                    cur_y = (index / x2) + 1 ;
                                    gotoxy(cur_x, cur_y) ;
                                    cprintf(txt+prev) ;
                                    txt[index] = '\0' ;
                                    gotoxy(cur_x, cur_y) ;
                                   }
```

```
                                        break ;
                    } /* switch(key) */
                } /* else */
        } /* while(TRUE) */
 finish :
 window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
 textattr(ti.attribute) ;
 gotoxy(ti.curx, ti.cury) ;
 if(txt == NULL)
   return NULL ;
 else
      {
       ret_txt = dup_str(txt) ;
       free(txt) ;
       txt = NULL ;
       return ret_txt ;
      }
}
void ver_line(int x,int top,int bottom)
{
  int i ;
  window(1,1,80,25) ;
  for (i=top; i<bottom ; i++)
      {
        gotoxy(x,i) ;
        putchar(179) ;
      }
}
void draw(int left,int top,int right,int bottom)
       {
         int j ;
          window(1,1,80,25) ;
          for (j=left+1; j<=right; j++)
              {
                  gotoxy(j,top) ;
                  putchar(205) ;
                  gotoxy(j,bottom) ;
                  putchar(205) ;
              }
          for (j=top+1; j<=bottom; j++)
              {
                gotoxy(left,j) ;
                putchar(186) ;
                gotoxy(right,j) ;
                putchar(186) ;
              }
          gotoxy(left,top) ;          putchar(201) ;
          gotoxy(right,top) ;         putchar(187) ;
          gotoxy(left,bottom) ;        putchar(200) ;
          gotoxy(right,bottom) ;   putchar(188) ;
         window(left+1,top+1,right-1,bottom-1) ;
          clrscr() ;
} /* end of draw */
void status_window(char *msg)
{
 struct text_info ti ;
 gettextinfo(&ti) ;
 window(1,25,80,25) ;
 textbackground(WHITE) ;
```

```
 textcolor(BLACK) ;
 clrscr() ;
 cprintf("%s",msg) ;
 window(ti.winleft,ti.wintop,ti.winright,ti.winbottom) ;
 gotoxy(ti.curx,ti.cury) ;
 textattr(ti.attribute ) ;
}
char *get_char_array(int n)
{
 char *p ;
 p = (char *) malloc(sizeof(char)*(n+1)) ;
 if(p == NULL)
   {
    printf("\nOUT OF MEMORY , ABORTING !    (get_char_array)") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 *p = '\0' ;
 return p ;
}
void clear_windows(void)
{
 struct text_info ti ;
 gettextinfo(&ti) ;
 window(41,3,77,23) ;
 clrscr() ;
 window(4,3,38,23) ;
 clrscr() ;
 window(ti.winleft,ti.wintop,ti.winright,ti.winbottom) ;
 return ;
}
char *trim(char *str)
{
 int i, k = 0 ;
 char txt[MAX_LENGTH] ;
 char prev ;
 for(i=strlen(str)-1 ; i >= 0 ; --i)
    {
     if(isspace(str[i]) || iscntrl(str[i]))
       str[i] = '\0' ;
     else
       break ;
    }
 for(i=0 ; i < strlen(str) && (isspace(str[i]) || iscntrl(str[i])) ; ++
+i)
    ;
 strcpy(txt,str+i) ;
 for(k=0, i=0, prev='A' ; i < strlen(txt) ; ++i)
    {
     if(isspace(txt[i]) || iscntrl(txt[i]))
       {
        if(txt[i] == '\t' || txt[i] == ' ')
          txt[i] = ' ' ;
        else
          continue ;
       }
     if((prev == txt[i]) && (prev == ' '))
       continue ;
```

```
        str[k++] = txt[i] ;
        prev = txt[i] ;
      }
  str[k] = '\0' ;
  return str ;
}
char *dup_str(char *input)
{
  char *p = get_char_array(strlen(input)) ;
  strcpy(p,input) ;
  return p ;
}
STACK *get_stack(void)
{
  STACK *p ;
  p = (STACK *) malloc(sizeof(STACK)) ;
  if(p == NULL)
    {
      printf("\OUT OF MEMORY,      ABORTING !    ...(get_stack)") ;
      fflush(stdin) ;
      getch() ;
      exit(1) ;
    }
  p->total_texts = 0 ;
  p->tos = NULL ;
  return p ;
}
int empty_stack(STACK *S)
{
  if(S->tos == NULL)
    return YES ;
  else
      return NO ;
}
void push(char *info, STACK *S)
{
  struct name_list *new_info ;
  new_info = get_link_node() ;
  new_info->name = dup_str(info) ;
  new_info->next = S->tos ;
  S->tos = new_info ;
  ++S->total_texts ;
  return ;
}
char *pop(STACK *S)
{
  struct name_list *reqd_info ;
  char *temp = NULL ;
  reqd_info = S->tos ;
  temp = reqd_info->name ;
  S->tos = S->tos->next ;
  free(reqd_info) ;
  reqd_info = NULL ;
  --S->total_texts ;
  return temp ;
}
void clear_stack(STACK *S)
{
  if(S->tos)
```

```
    {
     list_dispose(S->tos) ;
     S->tos = NULL ;
    }
 return ;
}
void free_STACK(STACK *S)
{
 clear_stack(S) ;
 free(S) ;
 S = NULL ;
return ;
}
```

```
/****** SHELL.C *******/


#include "shell.h"
extern int curr_rule_no ;
struct qual_name *qualifier_tree ;
struct name_tree *frame_tree, *no_qual_tree ;
long int curr_gen = 1, curr_gpr = 1 ;
int curr_qualifier_no = 1 ;
STACK *res_stack ;
int passed = NO ;
char file_name[40] = "GENERAL" ;
FILE *frm_fp, *frm_ndx_fp, *frm_pr_fp, *frm_rel ;
/* In Other Files */
void status_window(char *) ;
char *get_char_array(int) ;
void draw(int, int, int, int) ;
void ver_line(int, int, int) ;
char *get_ch_txt(int, int, int, int) ;
char *get_prob_txt(int, int, int, int) ;
void get_rules(void) ;
void view_rules(void) ;
void rules_write(void) ;
void read_rules(void) ;
void fire_rules(void) ;
void res_print(void) ;
void release_result_memory(void) ;
void successful_rules_view(void) ;
void fail_rule_detail(void) ;
char *trim(char *) ;
char *dup_str(char *) ;
void clear_windows(void) ;
STACK *get_stack(void) ;
void push(char *, STACK *) ;
char *pop(STACK *) ;
int empty_stack(STACK *) ;
void fill_relation_tree(FRAME *) ;
RELATIONS *get_RELATION_node(void) ;
RELATIONS *add_relation(RELATIONS *, RELATIONS *) ;
RELATIONS *RELATIONS_get(char *) ;
void add_left_relation(char *, RELATIONS *) ;
void add_right_relation(char *, RELATIONS *) ;
int read_relations(void) ;
int write_relations(void) ;
void show_relation_info(void) ;
void new_rule_save_option(void) ;
/* In This File */
void initialize(void) ;
void read_from_file(void) ;
int open_files(void) ;
void input_frame(void) ;
int exist_frame(char *, struct name_tree **, struct qual_name **, int)
 ;
int gdx_read(struct qual_name *) ;
int link_read_gdx(struct name_tree *) ;
int gpr_read(struct qual_name *) ;
int get_data(void) ;
struct name_tree * list_get_name_ptr(char *, struct name_list *) ;
int get_verb_ind(char *) ;
```

```
void goto_line(FILE *, long int) ;
char *name_get(char *) ;
void qual_parent_add(char *, struct qual_name *) ;
void add_name_to_qualifier(char *, struct qual_name *) ;
struct name_tree * add_no_qual(struct name_tree *, struct name_tree *)
 ;
struct name_tree *add_frame(struct name_tree *, struct name_tree *) ;
struct qual_name * add_qualifier(struct qual_name *, struct qual_name
*) ;
struct name_tree * get_frm_ptr(char *) ;
int w_frame(FRAME *) ;
FRAME *read_frm(FILE *) ;
int w_frm_pr_ind(void) ;
int w_gpr_gdx(struct qual_name *) ;
int gpr_w(struct qual_name *) ;
int gdx_w(struct qual_name *) ;
int gdx_link_w(struct name_tree *) ;
struct qual_name * get_qual_node(void) ;
FRAME *getnode(void) ;
struct name_tree *get_name_node(void) ;
struct parent_list *get_parent_node(void) ;
struct name_list *get_link_node(void) ;
char *nstrcpy(char *, char *, int) ;
FRAME *get_nm_qualifier(char *) ;
FRAME *get_num_qualifier(int) ;
struct name_tree *get_no_qualifier_in_tree(int) ;
void disp_no_qualifier(int) ;
void disp_name_quaifier(char *) ;
void disp_frame(FRAME *) ;
int show_qualifiers(int) ;
void random_movement(void) ;
int show_super_attributes(struct qual_name *) ;
int show_each_attributes(struct qual_name *) ;
struct name_tree *disp_info_qualifier(struct qual_name *, char *, int
*) ;
void disp_attr_info(struct qual_name *) ;
struct qual_name *qual_name_node(char *) ;
int find_in_object(void) ;
struct qual_name *get_object(char *, struct qual_name *) ;
void list_dispose(struct name_list *) ;
void init_objects(struct qual_name *) ;
void path_prnt(STACK *) ;
int find_out(char *, char *, struct name_list *) ;
int get_value(void) ;
int value_find(char *, char *, long int) ;
void process_frame(FRAME *) ;
void free_FRAME(FRAME *) ;
main(int argc, char *argv[])
{
 int choice = 'A';
 int fired = NO ;
 window(1,1,80,25) ;
 textcolor(WHITE) ;
 textbackground(BLACK) ;
 clrscr() ;
 if(argc > 1)
   {
    passed = YES ;
    strcpy(file_name,argv[1]) ;
```

```
    }
 initialize() ;
do /* while(!exit) */
  {
   choice = 'K' ;
   if(strchr("KR",choice))
     {
      textcolor(BLACK) ;
      textbackground(YELLOW) ;
      draw(3,2,78,24) ;
      ver_line(39,3,24) ;
      status_window("(K)nowledge base, (R)ule base, e(X)it") ;
     }
   choice = toupper(getch()) ;
   if(choice == 'K')
     {
      do
        {
         choice = 'A' ;
         if(strchr("IARESLPV",choice))
           {
            textcolor(BLACK) ;
            textbackground(YELLOW) ;
            draw(3,2,78,24) ;
            ver_line(39,3,24) ;
            status_window("(I)nput,(A)ll,(R)andom,(E)ach/(S)uper,re(L)
ation,(P)ath,(V)alue,(Q)uit") ;
           }
         choice = toupper(getch()) ;
         switch(choice)
               {
                case EXTENDED : getch() ;
                                break ;
                case 'Q'      : break ;
                case 'I'      : input_frame() ;
                                break ;
                case 'A'      : show_qualifiers(1) ;
                                break ;
                case 'R'      : random_movement() ;
                                break ;
                case 'E'      : show_each_attributes(qualifier_tree) ;
                                break ;
                case 'S'      : show_super_attributes(qualifier_tree)
;
                                break ;
                case 'L'      : show_relation_info() ;
                                break ;
                case 'P'      : find_in_object() ;
                                break ;
                case 'V'      : get_value() ;
                                break ;
                default       : putchar('\a') ;
                                break ;
               }
        } while(choice != 'Q') ;
     }
   else if(choice == 'R')
          {
           do
```

```
                {
                 choice = 'A' ;
                 if(strchr("IVRSFAU",choice))
                   {
                    textcolor(BLACK) ;
                    textbackground(YELLOW) ;
                    draw(3,2,78,24) ;
                    textcolor(YELLOW) ;
                    textbackground(BLACK) ;
                    clrscr() ;
                    status_window("(I)nput,(V)iew,fi(R)e,(S)uccesful/(F)a
iled,s(A)ve rules,res(U)lts,(Q)uit") ;
                   }
                 choice = toupper(getch()) ;
                 switch(choice)
                       {
                        case EXTENDED : getch() ;
                                        break ;
                        case 'Q'      : break ;
                        case 'I'      : do
                                          {
                                           ver_line(39,3,24) ;
                                           get_rules() ;
                                           status_window("Add more Rules
? (Y/N)") ;

                                           do
                                             {
                                              fflush(stdin) ;
                                              choice = toupper(getch()) ;
                                             } while(!(choice == 'Y' || c
hoice == 'N')) ;
                                           choice = choice == 'Y' ? YES :
 NO ;
                                          } while(choice) ;
                                        window(4,3,77,23) ;
                                        clrscr() ;
                                        break ;
                        case 'A'      : rules_write() ;
                                        break ;
                        case 'R'    : if(curr_rule_no > 1)
                                          {
                                           window(4,3,77,23) ;
                                           clrscr() ;
                                           release_result_memory() ;
                                           fire_rules() ;
                                           window(4,3,77,23) ;
                                           res_print() ;
                                           fired = YES ;
                                          }
                                        break ;
                        case 'U'      : if(fired)
                                          {
                                           clrscr() ;
                                           window(4,3,77,23) ;
                                           res_print() ;
                                          }
                                        break ;
                        case 'S'      : if(fired)
                                          {
```

```
                                        clrscr() ;
                                        window(4,3,77,23) ;
                                        successful_rules_view() ;
                                       }
                                     break ;
                        case 'F'     : if(fired)
                                        {
                                        clrscr() ;
                                        window(4,3,77,23) ;
                                        fail_rule_detail() ;
                                       }
                                     break ;
                        case 'V'     : view_rules() ;
                                     break ;
                        default      : putchar('\a') ;
                                     break ;
                      }
                } while(choice != 'Q') ;
           }
   } while(choice != 'X') ;
 release_result_memory() ;
 new_rule_save_option() ;
 fcloseall() ;
 window(1,1,80,25) ;
 textcolor(WHITE) ;
 textbackground(BLACK) ;
 clrscr() ;
 return 0 ;
}
void initialize(void)
{
 int success ;
 clrscr() ;
 frame_tree = no_qual_tree = NULL ;
 qualifier_tree = NULL ;
 curr_gen = curr_ppr = 1 ;
 curr_qualifier_no = 1 ;
 success = open_files() ;
 if(success)
   {
    read_from_file() ;
    read_rules() ;
   }
 return ;
}
int open_files(void)
{
 char *txt, f_name[MAX_LENGTH] ;
 int choice, i ;
 int new = NO ;
 int x1,y1 ;
 struct text_info ti ;
 gettextinfo(&ti) ;
 if(passed)
    strcpy(f_name,file_name) ;
 else
    *f_name = '\0' ;
 do
   {
```

```
      new = YES ;
    }
  fclose(frm_fp) ;
  if(!new)
    {
     sprintf(f_name,"%s.GDX",file_name) ;
     frm_ndx_fp = fopen(f_name,"r") ;
     if(frm_ndx_fp == NULL)
       {
        status_window("Cannot find all the necessary data files.... ABO
RTING !") ;
        fcloseall() ;
        exit(1) ;
       }
     sprintf(f_name,"%s.GPR",file_name) ;
     frm_pr_fp = fopen(f_name,"r") ;
     if(frm_pr_fp == NULL)
       {
        status_window("Cannot find all the necessary data files.... ABO
RTING !") ;
        fcloseall() ;
        exit(1) ;
       }
     sprintf(f_name,"%s.REL",file_name) ;
     frm_rel = fopen(f_name,"r") ;
     if(frm_rel == NULL)
       {
        status_window("Cannot find all the necessary data files.... ABO
RTING !") ;
        fcloseall() ;
        exit(1) ;
       }
    }
  else
     {
      FILE *fp ;
      do
        {
         sprintf(f_name,"%s.RUL",file_name) ;
         fp = fopen(f_name,"r") ;
         if(fp == NULL)
           {
            sprintf(f_name,"Cannot open \"%s.RUL\".   New(N), Retry(R)
",file_name) ;
            status_window(f_name) ;
            do
              {
               fflush(stdin) ;
               choice = toupper(getch()) ;
              } while(!(choice == 'R' || choice == 'N')) ;
            if(choice == 'N')
              break ;
           }
        } while(fp == NULL) ;
      if(fp == NULL)
        {
         sprintf(f_name,"%s.RUL",file_name) ;
         fp = fopen(f_name,"w") ;
         if(fp == NULL)
```

```
            {
             status_window("Cannot create file.      ..ABORTING       ...
Press any key") ;
             fflush(stdin) ;
             getch() ;
             fcloseall() ;
             exit(1) ;
            }
          fprintf(fp,"%d\n",1) ;
          fclose(fp) ;
         }
       else
           read_rules() ;
      }
 sprintf(f_name,"%s.GEN",file_name) ;
 frm_fp = fopen(f_name,"at") ;
 if(frm_fp == NULL)
   {
    status_window("Error while creating(opening) a new(old) file....AB
ORTING !") ;
    fcloseall() ;
    exit(1) ;
   }
 return new ? NO : YES ;
}
void input_frame(void)
{
 int sure, more = YES, write = NO ;
 struct text_info ti ;
 gettextinfo(&ti) ;
 clrscr() ;
 window(3,2,78,24) ;
 textcolor(CYAN) ;
 textbackground(BLACK) ;
 clrscr() ;
 draw(3,2,78,24) ;
 ver_line(39,3,24) ;
 do
   {
    sure = get_data() ;
    write += sure ;
    status_window("Enter more qualifiers ? (Y/N)") ;
    do
      {
       more = toupper(getch()) ;
      } while(!(more == 'Y' || more == 'N')) ;
    more = more == 'Y' ? YES : NO ;
   } while(more) ;
 if(write)
   {
    w_frm_pr_ind() ;
    write_relations() ;
   }
 window(ti.winleft,ti.wintop,ti.winright,ti.winbottom) ;
 textattr(ti.attribute) ;
 gotoxy(ti.curx,ti.cury) ;
 return ;
}
void read_from_file(void)
```

```
{
 int finish = NO, done = NO ;
 int x ;
 long int ln ;
 struct qual_name *q ;
 fscanf(frm_ndx_fp,"%ld %d\n",&ln,&x) ;
 if(feof(frm_ndx_fp))
   {
    clrscr() ;
    printf("Abnormal end of \"%s.GDX\"..... ABORTING..\n",file_name) ;
    exit(1) ;
   }
 else
     {
      curr_gen = ln ;
      curr_qualifier_no = x ;
     }
 if(curr_gen > 1)
   {
    while(!finish)
         {
          q = get_qual_node() ;
          finish = gdx_read(q) ;
          if(!done)
            {
             if(finish)
               done = YES ;
             gpr_read(q) ;
             qualifier_tree = add_qualifier(q,qualifier_tree) ;
            }
         }
   }
 read_relations() ;
 return ;
}
int gdx_read(struct qual_name *q)
{
 long int ln ;
 int x ;
 struct name_tree *p = NULL ;
 char txt[MAX_LENGTH] ;
 fscanf(frm_ndx_fp,"%[^\n]\n",txt) ;
 q->name = dup_str(txt) ;
 fscanf(frm_ndx_fp,"%ld\n",&ln) ;
 q->line_gpr = ln ;
 fscanf(frm_ndx_fp,"%d\n",&x) ;
 for(; x >0 ; --x)
    {
     p = get_name_node() ;
     link_read_gdx(p) ;
     add_name_to_qualifier(p->name,q) ;
     frame_tree = add_frame(p,frame_tree) ;
     no_qual_tree = add_no_qual(p,no_qual_tree) ;
    }
 if(feof(frm_ndx_fp))
   return YES ;
 else
   return NO ;
}
```

```
int link_read_gdx(struct name_tree *p)
{
 char txt[MAX_LENGTH] ;
 int finish = NO, x ;
 long int ln ;
 fscanf(frm_ndx_fp,"%[^\n]\n",txt) ;
 p->name = dup_str(txt) ;
 fscanf(frm_ndx_fp,"%d\n",&x) ;
 p->qualifier_no = x ;
 fscanf(frm_ndx_fp,"%ld\n",&ln) ;
 p->line_gen = ln ;
 return finish ;
}
int gpr_read(struct qual_name *q)
{
 int x ;
 char txt[MAX_LENGTH] ;
 fscanf(frm_pr_fp,"%*[^\n]\n") ;          /* As "q" already contains
 the
                                             qualifier name */
 fscanf(frm_pr_fp,"%d\n",&x) ;
 for(; x > 0 ; --x)
    {
     fscanf(frm_pr_fp,"%[^\n]\n",txt) ;
     qual_parent_add(txt,q) ;
    }
 if(feof(frm_pr_fp))
   return YES ;
 else
   return NO ;
}
int exist_frame(char *txt, struct  name_tree **p, struct qual_name **q
, int search_for)
{
 char *name_only ;
 int test ;
 struct qual_name *curr ;
 *p = NULL ;
 *q = NULL ;
 if(search_for == CHLD)
   {
    for(curr=qualifier_tree ; curr ;)
       {
        if((test = stricmp(txt,curr->name)) > 0)
          curr = curr->right ;
        else if(test < 0)
               curr = curr->left ;
             else
                 {
                  *q = curr ;
                  return YES ;
                 }
       }
    return NO ;
   }
 else      /* search_for == PARENT */
    {
     name_only = name_get(txt) ;
     for(curr=qualifier_tree ; curr ;)
```

```
        {
         if((test = stricmp(name_only,curr->name)) > 0)
           curr = curr->right ;
         else if(test < 0)
                curr = curr->left ;
              else
                  {
                   *q = curr ;
                   if(curr->no_links != 0)
                     {
                      *p = list_get_name_ptr(txt,curr->head) ;
                      if(*p != NULL)
                        {
                         if(curr->no_parents > 0)
                           return PRNT_CHILD ;
                         else
                             return PARENT ;
                        }
                      else
                          {
                           *p = get_frm_ptr(curr->head->name) ;
                           if(curr->no_parents > 0)
                             return PART_CHILD ;
                           else
                               return PART_PRNT ;
                          }
                     }
                   else
                       return CHLD ;
                  }
        } /* for */
      return NO ;
     } /* else of if(search_for == CHLD) */
}
int get_data(void)     /* returns YES if additions are done & otherwise
 */
{                      /* returns NO
 */
 int no_of_attributes = 0 ;
 int exists = PARENT, sure = NO ;
 FRAME frm, *f ;
 char *temp_txt ;
 char txt[MAX_LENGTH], gen[MAX_LENGTH] ;
 struct name_tree *p, *parent_ptr ;
 struct qual_name *q ;
 struct parent_list *curr ;
 int x1, y1, y2, i, k ;
 int error = NO ;
 struct text_info t1, t2 ;
 frm.name = NULL ;
 frm.qualifier_no = 0 ;
 frm.no_childs = 0 ;
 for(i=0 ; i < NO_CHILDS ; ++i)
    frm.childs[i] = NULL ;
 gettextinfo(&t1) ;
 window(4,3,33,23) ;
 clear_windows() ;
 gotoxy(2,1) ;
 status_window("Enter a text ending in a verb") ;
```

```
 fflush(stdin) ;
 cprintf("#%d ",curr_qualifier_no) ;
 temp_txt = get_ch_txt(wherex(),wherey(),30,3) ;
 if(temp_txt == NULL)
   {
    clrscr() ;
    goto finish ;
   }
 trim(temp_txt) ;
 if(strlen(temp_txt) == 0)
   goto finish ;
 strcpy(txt,temp_txt) ;
 free(temp_txt) ;
 if(get_verb_ind(txt) == 0)
   {
    fflush(stdin) ;
    status_window("Error: Qualifier name or Verb is missing.. Press an
y key to continue..") ;
    getch() ;
    status_window("") ;
    goto finish ;
   }
 exists = exist_frame(txt, &p, &q, PARENT) ;
 if(exists)
   {
    gettextinfo(&t2) ;
    window(41,3,77,23) ;
    clrscr() ;
    switch(exists)
          {
           case CHLD       :
           case PART_CHILD : textcolor(BLACK) ;
                             textbackground(WHITE) ;
                             cprintf("Present in the following qualifi
ers :\n\r") ;
                             textcolor(WHITE) ;
                             textbackground(BLACK) ;
                             for(i=0, curr = q->root ; i < q->no_paren
ts ; curr = curr->next)
                                {
                                 parent_ptr = get_frm_ptr(curr->name)
;
                                 sprintf(gen,"  %2d.  (#%2d) %s\n\r",+
+i, parent_ptr->qualifier_no, parent_ptr->name) ;
                                 cprintf(gen) ;
                                }
                             break ;
           case PARENT     :
           case PRNT_CHILD :
           case PART_PRNT  : goto_line(frm_fp,p->line_gen) ;
                             f = read_frm(frm_fp) ;
                             if(f == NULL)
                               {
                                sprintf(gen,"Error in file \"%s.GDX\".
... ABORTING..\n\n") ;
                                clrscr() ;
                                printf(txt) ;
                                exit(1) ;
                               }
```

```
                                disp_frame(f) ;
                                break ;
                        } /* switch(exists) */
        window(t2.winleft,t2.wintop,t2.winright,t2.winbottom) ;
        textattr(t2.attribute) ;
        gotoxy(t2.curx,t2.cury) ;
       } /* if(exists) */
 if(exists == PARENT || exists == PRNT_CHILD)
   {
    status_window("This qualifier already exists... Press any key to c
ontinue..") ;
    fflush(stdin) ;
    getch() ;
    goto finish ;
   }
 else
     {
      clrscr() ;
      gotoxy(2,1) ;
      textcolor(BLACK) ;
      textbackground(WHITE) ;
      cprintf("  r%d  %s\n\r",curr_qualifier_no, txt) ;
      textattr(t1.attribute) ;
      frm.name = dup_str(txt) ;
      status_window("Enter attributes    ..<ENTER> to finish") ;
      do
        {
         x1 = wherex() ;
         y1 = wherey() ;
         error = NO ;
         cprintf("     %2d.  ",no_of_attributes+1) ;
         fflush(stdin) ;
         temp_txt = get_ch_txt(wherex(),wherey(),25,3) ;
         if(temp_txt == NULL)
           break ;
         trim(temp_txt) ;
         if(strlen(temp_txt) == 0)
           break ;
         strcpy(txt,temp_txt) ;
         free(temp_txt) ;
         y2 = wherey() ;
         if(stricmp(txt,name_get(frm.name)) == 0)
           {
            status_window("Error: Qualifier & attribute names can't be
 same..Press any key to continue") ;
            fflush(stdin) ;
            getch() ;
            error = YES ;
           }
         else
             {
              for(k=0 ; k < no_of_attributes ; ++k)
                 {
                  if(stricmp(txt,frm.childs[k]) == 0)
                    {
                     status_window("Error: Two attributes can't have s
ame names..Press any key to continue") ;
                     fflush(stdin) ;
                     getch() ;
```

```
                    error = YES ;
                   }
                 }
              }
         for(k=y1 ; k <= y2 || k <= y1+2 ; ++k)
            {
             gotoxy(1,k) ;
             clreol() ;
            }
         if(error == NO)
           {
            gotoxy(x1,y1) ;
            cprintf("    %2d.   %s\n\r",++no_of_attributes,txt) ;
            frm.childs[frm.no_childs] = dup_str(txt) ;
            ++frm.no_childs ;
           }
         else
             {
              status_window("Enter attributes    ..<ENTER> to finish")
;
              gotoxy(x1,y1) ;
             }
       } while(no_of_attributes < NO_CHILDS) ;
     if(no_of_attributes > 0)
       {
        status_window("Are You Sure ? (Y/N)") ;
        fflush(stdin) ;
        do
          {
           sure = toupper(getch()) ;
          } while(!(sure == 'Y' || sure == 'N')) ;
        sure = sure == 'Y' ? YES : NO ;
        if(!sure)
          {
           free(frm.name) ;
           for(i=0 ; i < frm.no_childs ; ++i)
              free(frm.childs[i]) ;
           goto finish ;
          }
       }
     else
         {
          free(frm.name) ;
          for(i=0 ; i < frm.no_childs ; ++i)
             free(frm.childs[i]) ;
          goto finish ;
         }
       process_frame(&frm) ;
    } /* else of if(exists == PARENT || exists == PRNT_CHILD) */
free(frm.name) ;
for(i=0 ; i < frm.no_childs ; ++i)
   free(frm.childs[i]) ;
finish :
clear_windows() ;
window(t1.winleft,t1.wintop,t1.winright,t1.winbottom) ;
textattr(t1.attribute) ;
gotoxy(t1.curx,t1.cury) ;
status_window("") ;
if(no_of_attributes && sure)
```

```
    return YES ;
  else
    return NO ;
}
struct name_tree * list_get_name_ptr(char *txt, struct name_list *root
)
{
 struct name_list *curr ;
 for(curr = root ; curr ; curr = curr->next)
    {
     if(stricmp(curr->name,txt) == 0)
       return get_frm_ptr(txt) ;
    }
 return NULL ;
}
int get_verb_ind(char *txt)
{
 int i ;
 for(i = strlen(txt)-1 ; i > 0 && txt[i] != ' ' ; --i)
    ;
 if(i == 0)
   return NO ;
 else
     return ++i ;  /* ie 1 plus position of last ' '*/
}
void goto_line(FILE *fp, long int line_no)
{
 long int i ;
 rewind(fp) ;
 for(i = 1 ; i < line_no ; ++i)
    fscanf(fp,"%*[^\n]\n") ;
 return ;
}
char *name_get(char *txt)
{
 int n = get_verb_ind(txt)-1 ;
 char *p = get_char_array(n) ;
 nstrcpy(p,txt,n) ;
 return p ;
}
void qual_parent_add(char *p_name, struct qual_name *p)
{
 struct parent_list *q ;
 q = get_parent_node() ;
 q->name = dup_str(p_name) ;
 q->next = p->root ;
 p->root = q ;
 ++p->no_parents ;
 return ;
}
void add_name_to_qualifier(char *p_name, struct qual_name *p)
{
 struct name_list *q ;
 q = get_link_node() ;
 q->name = dup_str(p_name) ;
 q->next = p->head ;
 p->head = q ;
 ++p->no_links ;
 return ;
```

```
}
struct name_tree * add_no_qual(struct name_tree *p, struct name_tree *
root)
{
 if(root == NULL)
   root = p ;
 else
     {
      if(p->qualifier_no > root->qualifier_no)
        root->qual_right = add_no_qual(p,root->qual_right) ;
      else
          root->qual_left = add_no_qual(p,root->qual_left) ;
     }
 return root ;
}
struct name_tree *add_frame(struct name_tree *p, struct name_tree *roo
t)
{
 if(root == NULL)
   root = p ;
 else
     {
      if(stricmp(p->name,root->name) > 0)
        root->right_name = add_frame(p,root->right_name) ;
      else
          root->left_name = add_frame(p,root->left_name) ;
     }
 return root ;
}
struct qual_name * add_qualifier(struct qual_name *p, struct qual_name
 *root)
{
 if(root == NULL)
   root = p ;
 else
     {
      if(stricmp(p->name,root->name) > 0)
        root->right = add_qualifier(p,root->right) ;
      else
          root->left = add_qualifier(p,root->left) ;
     }
 return root ;
}
struct name_tree * get_frm_ptr(char *name)
{
 int test ;
 struct name_tree *curr ;
 for(curr = frame_tree ; curr ;)
    {
     if((test = stricmp(name,curr->name)) > 0)
       curr = curr->right_name ;
     else if(test < 0)
            curr = curr->left_name ;
          else
              break ;
    }
 return curr ;
}
int w_frame(FRAME *frm)
```

```
{
 int success = YES ;
 int i ;
 fseek(frm_fp,0,SEEK_END) ;
 fprintf(frm_fp,"%s\n",frm->name) ;
 fprintf(frm_fp,"%d\n",frm->qualifier_no) ;
 fprintf(frm_fp,"%d\n",frm->no_childs) ;
 for(i=0 ; i < frm->no_childs ; ++i)
     fprintf(frm_fp,"%s\n",frm->childs[i]) ;
 curr_gen += (frm->no_childs+3) ;
 fflush(frm_fp) ;
 return success ;
}
FRAME * read_frm(FILE *fp)
{
 int x ;
 FRAME *frm ;
 char txt[MAX_LENGTH] ;
 int success = YES ;
 frm = getnode() ;
 fscanf(fp,"%[^\n]\n",txt) ;
 frm->name = dup_str(txt) ;
 fscanf(frm_fp,"%d\n",&x) ;
 frm->qualifier_no = x ;
 fscanf(fp,"%d\n",&x) ;
 frm->no_childs = x ;
 for(x=0 ; x < frm->no_childs ; ++x)
    {
     fscanf(fp,"%[^\n]\n",txt) ;
     frm->childs[x] = dup_str(txt) ;
    }
 if(!success)
   {
    free(frm->name) ;
    for(x=0 ; x < frm->no_childs ; ++x)
       free(frm->childs[x]) ;
    free(frm) ;
    frm = NULL ;
   }
 return frm ;
}
int w_frm_pr_ind(void)
{
 char temp[MAX_LENGTH] ;
 int success = YES ;
 int choice = 'N' ;
 fclose(frm_ndx_fp) ;
 fclose(frm_pr_fp) ;
 sprintf(temp,"%s.BOX",file_name) ;
 do
   {
    choice = 'N' ;
    frm_ndx_fp = fopen(temp,"w") ;
    if(frm_ndx_fp == NULL)
      {
       status_window("Error: Unable to create index files.. Retry ? (Y
/N)") ;
       fflush(stdin) ;
       do
```

```
          {
           choice = toupper(getch()) ;
          } while(!(choice == 'Y' || choice == 'N')) ;
        status_window("") ;
        if(choice == 'N')
          return NO ;
        else
          continue ;
       }
     sprintf(temp,"%s.GPR",file_name) ;
     frm_pr_fp = fopen(temp,"w") ;
     if(frm_pr_fp == NULL)
       {
        fclose(frm_ndx_fp) ;
        status_window("Error: Unable to create files.. Retry ? (Y/N)")
;
        fflush(stdin) ;
        do
          {
           choice = toupper(getch()) ;
          } while(!(choice == 'Y' || choice == 'N')) ;
        status_window("") ;
        if(choice == 'N')
          return NO ;
       }
    } while(choice == 'Y') ;
 fprintf(frm_ndx_fp,"%ld %d\n",curr_gen,curr_qualifier_no) ;
 curr_gpr = 1 ;
 w_gpr_gdx(qualifier_tree) ;
 return success ;
}
int w_gpr_gdx(struct qual_name *p)
{
 int success = YES ;
 if(p!= NULL)
   {
    gpr_w(p) ;
    gdx_w(p) ;
    w_gpr_gdx(p->left) ;
    w_gpr_gdx(p->right) ;
   }
 return success ;
}
int gpr_w(struct qual_name *p)
{
 struct parent_list *curr ;
 int success = YES, i ;
 p->line_gpr = curr_gpr ;
 fprintf(frm_pr_fp,"%s\n",p->name) ;
 fprintf(frm_pr_fp,"%d\n",p->no_parents) ;
 for(i=0, curr = p->root ; i < p->no_parents && curr ; ++i, curr = cur
r->next)
    {
     fprintf(frm_pr_fp,"%s\n",curr->name) ;
     /* Later :
        fprintf(frm_pr_fp,"%s\n",curr->child_relation) ; */
    }
 curr_gpr += (p->no_parents + 2 ) ;
 return success ;
```

```
}
int gdx_w(struct qual_name *p)
{
 int success = YES, i ;
 struct name_list *curr ;
 fprintf(frm_ndx_fp,"%s\n",p->name) ;
 fprintf(frm_ndx_fp,"%ld\n",p->line_gpr) ;
 fprintf(frm_ndx_fp,"%d\n",p->no_links) ;
 for(i=0, curr = p->head ; i < p->no_links && curr ; ++i, curr = curr-
>next)
    gdx_link_w(get_frm_ptr(curr->name)) ;
 return success ;
}
int gdx_link_w(struct name_tree *p)
{
 int success = YES ;
 fprintf(frm_ndx_fp,"%s\n",p->name) ;
 fprintf(frm_ndx_fp,"%d\n",p->qualifier_no) ;
 fprintf(frm_ndx_fp,"%ld\n",p->line_gen) ;
 return success ;
}
struct qual_name * get_qual_node(void)
{
 struct qual_name *p ;
 p = (struct qual_name *) malloc(sizeof(struct qual_name)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING        ....(get_qual_node)\n
") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->name = NULL ;
 p->no_links = p->no_parents = 0 ;
 p->line_gpr = 0 ;
 p->head = NULL ;
 p->root = NULL ;
 p->left = p->right = NULL ;
 p->traversed = NO ;
 return p ;
}
FRAME * getnode(void)
{
 int i ;
 FRAME *p ;
 p = (FRAME *) malloc(sizeof(FRAME)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING        ....(getnode)\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->name = NULL ;
 p->qualifier_no = 0 ;
 p->no_childs = 0 ;
```

```
 for(i=0 ; i < NO_CHILDS ; ++i)
    p->childs[i] = NULL ;
 return p ;
}
struct name_tree * get_name_node(void)
{
 struct name_tree *p ;
 p = (struct name_tree *) malloc(sizeof(struct name_tree)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING        ....(get_name_node)\n
") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->name = NULL ;
 p->line_gen = 0 ;
 p->head = NULL ;
 p->qualifier_no = 0 ;
 p->left_name = p->right_name = NULL ;
 p->qual_left = p->qual_right = NULL ;
 return p ;
}
struct parent_list *get_parent_node(void)
{
 struct parent_list *p ;
 p = (struct parent_list *) malloc(sizeof(struct parent_list)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING        ....(get_parent_node)
\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->name = NULL ;
 /* p->child_relation = NULL ; */
 p->next = NULL ;
 return p ;
}
struct name_list *get_link_node(void)
{
 struct name_list *p ;
 p = (struct name_list *) malloc(sizeof(struct name_list)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING        ....(get_link_node)\n
") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->name = NULL ;
 p->next = NULL ;
 return p ;
```

```
}
char *nstrcpy(char *dest, char *src, int n)
{
 strncpy(dest,src,n) ;
 dest[n] = '\0' ;
 return dest ;
}
FRAME * get_nm_qualifier(char * name)
{
 struct name_tree *p ;
 p = get_frm_otr(name) ;
 if(p)
   {
    goto_line(frm_fp,p->line_gen) ;
    return read_frm(frm_fp) ;
   }
 else
     return NULL ;
}
FRAME * get_num_qualifier(int qual_no)
{
 struct name_tree *p ;
 p = get_no_qualifier_in_tree(qual_no) ;
 if(p)
   {
    goto_line(frm_fp, p->line_gen) ;
    return read_frm(frm_fp) ;
   }
 else
     return NULL ;
}
struct name_tree * get_no_qualifier_in_tree(int qual_no)
{
 struct name_tree *curr ;
 for(curr = no_qual_tree ; curr ;)
    {
     if(qual_no > curr->qualifier_no)
       curr = curr->qual_right ;
     else if(qual_no < curr->qualifier_no)
            curr = curr->qual_left ;
          else
              break ;
    }
 return curr ;
}
void disp_no_qualifier(int qual_no)
{
 disp_frame(get_num_qualifier(qual_no)) ;
 return ;
}
void disp_name_qualfier(char *name)
{
 disp_frame(get_nm_qualifier(name)) ;
 return ;
}
void disp_frame(FRAME *frm)
{
 int i ;
 textcolor(BLACK) ;
```

```
 textbackground(WHITE) ;
 cprintf(" #%d  %s\n\r",frm->qualifier_no, frm->name) ;
 textcolor(WHITE) ;
 textbackground(BLACK) ;
 for(i=0 ; i < frm->no_childs ; ++i)
    cprintf("     %2d.   %s\n\r",i+1, frm->childs[i]) ;
 return ;
}
int show_qualifiers(int disp_no)
{
 struct text_info ti ;
 int key, x ;
 FRAME *f ;
 char msg1[MESG_LENGTH], txt[MAX_LENGTH] ;
 char *temp_txt ;
 if(curr_qualifier_no == 1)
   {
    clear_windows() ;
    status_window("No qualifier's Entered Yet thus No movement.  ..Pre
ss Any Key\a") ;
    fflush(stdin) ;
    getch() ;
    status_window("") ;
    return 0 ;
   }
 if(disp_no < 1 || disp_no > (curr_qualifier_no -1))
   {
    sprintf(msg1,"(#%d):No such qualifier no.(name) in the database.
..Press any key") ;
    status_window(msg1) ;
    fflush(stdin) ;
    getch() ;
    return 0 ;
   }
 gettextinfo(&ti) ;
 textcolor(WHITE) ;
 textbackground(BLACK) ;
 clear_windows() ;
 window(4,3,77,23) ;
 clrscr() ;
 disp_no_qualifier(disp_no) ;
 sprintf(msg1,"Previous(%c or %c), Next(%c or %c), Qualifier #(N), Qua
lifier name(M), Exit(X)",27,24,26,25) ;
 do
   {
    status_window(msg1) ;
    fflush(stdin) ;
    key = toupper(getch()) ;
    switch(key)
          {
           case 'N' : window(1,1,80,25) ;
                      status_window("") ;
                      gotoxy(1,25) ;
                      cprintf("Enter qualifier #") ;
                      temp_txt = get_prob_txt(wherex(),wherey(),80-whe
rex()-2,1) ;
                      if(temp_txt == NULL)
                        break ;
                      if(strlen(temp_txt) != 0)
```

```
                        strcpy(txt,temp_txt) ;
                      free(temp_txt) ;
                      x = atoi(txt) ;
                      window(4,3,77,23) ;
                      if(x < 1 || x >= curr_qualifier_no)
                        {
                         char msg2[MESG_LENGTH] ;
                         sprintf(msg2,"(#%d):No such qualifier no. in
the database.    ...Press any key",x) ;
                         status_window(msg2) ;
                         fflush(stdin) ;
                         getch() ;
                        }
                      else
                          {
                           disp_no = x ;
                           clrscr() ;
                           disp_no_qualifier(disp_no) ;
                          }
                      break ;
           case 'N' : window(1,1,80,25) ;
                      status_window("") ;
                      gotoxy(1,25) ;
                      cprintf("Enter qualifier name : ") ;
                      temp_txt = get_ch_txt(wherex(),wherey(),80-where
x()-2,1) ;
                      if(temp_txt == NULL)
                        break ;
                      trim(temp_txt) ;
                      if(strlen(temp_txt) != 0)
                        strcpy(txt,temp_txt) ;
                      free(temp_txt) ;
                      window(4,3,77,23) ;
                      f = get_nm_qualifier(txt) ;
                      if(f == NULL)
                        {
                         char msg2[MESG_LENGTH] ;
                         sprintf(msg2,"(%s):No such qualifier name in
the database.    ...Press any key",txt) ;
                         status_window(msg2) ;
                         fflush(stdin) ;
                         getch() ;
                        }
                      else
                          {
                           disp_no = f->qualifier_no ;
                           clrscr() ;
                           disp_frame(f) ;
                          }
                      break ;
           case 'X'   :
           case ENTER :
           case ESC   : window(1,1,80,25) ;
                        ver_line(39,3,24) ;
                        window(ti.winleft,ti.wintop,ti.winright,ti.win
bottom) ;
                        textattr(ti.attribute) ;
                        gotoxy(ti.curx,ti.cury) ;
                        return disp_no ;
```

```
            case EXTENDED : switch(key = getch())
                                {
                                 case UP_ARR    :
                                 case LEFT_ARR  : if(disp_no == 1)
                                                    {
                                                     disp_no = curr_q
ualifier_no - 1 ;
                                                     putchar('\a') ;
                                                    }
                                                  else
                                                     --disp_no ;
                                                  clrscr() ;
                                                  disp_no_qualifier(d
isp_no) ;
                                                  break ;
                                 case DOWN_ARR  :
                                 case RIGHT_ARR : if(disp_no == curr_
qualifier_no - 1)
                                                    {
                                                     disp_no = 1 ;
                                                     putchar('\a') ;
                                                    }
                                                  else
                                                     ++disp_no ;
                                                  clrscr() ;
                                                  disp_no_qualifier(d
isp_no) ;
                                                  break ;
                                }
                             break ;
           }
   } while(1) ;
}
void random_movement(void)
{
 char txt[MAX_LENGTH] ;
 char *temp_txt ;
 struct name_tree *p1, *p2, *p = NULL ;
 struct qual_name *q1, *q2, *q = NULL ;
 int as_attribute = NO, as_qualifier = NO, display_valid = NO, see_chl
d = NO ;
 int key, qual_no ;
 struct text_info ti ;
 if(curr_qualifier_no == 1)
   {
    clear_windows() ;
    status_window("No qualifier's Entered Yet thus No movement.  ..Pre
ss Any Key\a") ;
    fflush(stdin) ;
    getch() ;
    status_window("") ;
    return ;
   }
 gettextinfo(&ti) ;
 clear_windows() ;
 window(4,3,39,23) ;
 do
   {
    status_window("Any text(T), Attribute(A), Related qualifier(R), Pa
```

```
rent(P), Exit(X)") ;
     fflush(stdin) ;
     key = toupper(getch()) ;
     switch(key)
          {
           case 'A' : if(!display_valid)
                        break ;
                      else
                          {
                           switch(as_qualifier)
                                 {
                                  case PARENT     :
                                  case PART_PRNT  :
                                  case PRNT_CHILD : {
                                                     FRAME *frm ;
                                                     int chl_no ;
                                                     window(1,1,80,25)
  ;
                                                     status_window("")
  ;
                                                     gotoxy(1,25) ;
                                                     cprintf("Enter Ch
ild S.No. : ") ;
                                                     temp_txt = get_pr
ob_txt(wherex(),wherey(),80-wherex()-2,1) ;
                                                     if(temp_txt == NU
LL)
                                                       break ;
                                                     if(strlen(temp_tx
t) != 0)
                                                       strcpy(txt,temp
_txt) ;
                                                     free(temp_txt) ;
  ;
                                                     window(4,3,38,23)
) ;
                                                     chl_no = atoi(txt
                                                     frm = get_nm_qual
ifier(p->name) ;
                                                     if(chl_no < 1 ||
chl_no > frm->no_childs)
                                                       break ;
                                                     strcpy(txt,frm->c
hilds[chl_no-1]) ;
                                                     see_chld = YES ;
                                                    }
                                                   break ;
                                  default         : see_chld = NO ;
                                                    break ;
                                 }
                          }
                      if(!see_chld)
                        break ;
           case 'T' : if(!see_chld)
                        {
                         window(1,1,80,25) ;
                         status_window("") ;
                         gotoxy(1,25) ;
                         cprintf("Enter any text : ") ;
```

```
                        temp_txt = get_ch_txt(wherex(),wherey(),80-wh
erex()-2,1) ;
                        if(temp_txt == NULL)
                          break ;
                        trim(temp_txt) ;
                        if(strlen(temp_txt) != 0)
                          strcpy(txt,temp_txt) ;
                        free(temp_txt) ;
                        window(4,3,38,23) ;
                        trim(txt) ;
                       }
                      if(get_verb_ind(txt) != 0)
                        as_qualifier = exist_frame(txt,&p1,&q1,PARENT
) ;
                      else
                        as_qualifier = NO ;
                      as_attribute = exist_frame(txt,&p2,&q2,CHLD) ;
                      if(as_attribute || as_qualifier)
                        {
                         see_chld = NO ;
                         display_valid = YES ;
                         clear_windows() ;
                        }
                      if(as_attribute && !as_qualifier)
                        {
                         p = p2 ;
                         q = q2 ;
                         textbackground(BLACK) ;
                         textcolor(WHITE) ;
                         cprintf(q->name) ;
                         textattr(ti.attribute) ;
                         disp_attr_info(q) ;
                        }
                      else
                          {
                           if(as_qualifier && !as_attribute)
                             {
                              q = q1 ;
                              p = disp_info_qualifier(q,txt,&as_qualif
ier) ;
                             }
                           else
                               {
                                if(as_attribute && as_qualifier)
                                  {
                                   int first = YES, finish = NO ;
                                   status_window("Present as both a qu
alifier and a attribute. See Qualifier(Q), Attribute(A)") ;
                                   do
                                     {
                                      fflush(stdin) ;
                                      key = toupper(getch()) ;
                                      switch(key)
                                            {
                                             case 'A' : p = p2 ;
                                                        q = q2 ;
                                                        first = NO ;
                                                        clear_windows(
) ;
```

```
                                                        textbackground
(BLACK) ;
                                                        textcolor(WHIT
E) ;
                                                        cprintf(q->nam
e) ;
                                                        textattr(ti.at
tribute) ;
                                                        disp_attr_info
(q) ;
                                                        break ;
                                                case 'Q' : q = q1 ;
                                                        first = NO ;
                                                        clear_windows(
) ;
                                                        p = disp_info_
qualifier(q,txt,&as_qualifier) ;
                                                        break ;
                                                case ESC    :
                                                case ENTER  :
                                                case 'T'    : if(!first)
                                                          finish = Y
ES ;
                                                          break ;
                                                }
                                        status_window("Qualifier(Q), Att
ribute(A), Travel(T)") ;
                                        } while(!finish) ;
                                    }
                                else
                                    {
                                    if(!as_attribute && !as_qualifier
)
                                      {
                                       char mesg[100] ;
                                       clear_windows() ;
                                       sprintf(mesg,"(%s):No such att
ribute or qualifier.    ....Press any key",txt) ;
                                       status_window(mesg) ;
                                       fflush(stdin) ;
                                       getch() ;
                                       display_valid = NO ;
                                       p = NULL ;
                                       q = NULL ;
                                      }
                                    }
                                }
                            }
                        break ; /* case 'T' */
                case 'P' : if(!display_valid)
                             break ;
                           if(q->no_links == 0)
                             break ;
                           if(q->no_links == 1)
                             {
                              clear_windows() ;
                              as_qualifier = -1 ;
                              p = disp_info_qualifier(q,q->head->name,&as_q
ualifier) ;
```

```
                        }
                      else
                          {
                           int correct = NO, i ;
                           char q_no[20] ;
                           struct name_tree *nt ;
                           struct name_list *curr ;
                           window(1,1,80,25) ;
                           status_window("") ;
                           gotoxy(1,25) ;
                           cprintf("Enter qualifier #") ;
                           temp_txt = get_prob_txt(wherex(),wherey(),8
0-wherex()-2,1) ;
                           if(temp_txt == NULL)
                             break ;
                           if(strlen(temp_txt) != 0)
                             strcpy(q_no,temp_txt) ;
                           free(temp_txt) ;
                           qual_no = atoi(q_no) ;
                           if(qual_no < 1)
                              correct = NO ;
                           else
                               {
                                for(i = 0, curr = q->head ; i < q->no_
links && curr ; ++i, curr = curr->next)
                                   {
                                    if(p != NULL && stricmp(p->name, c
urr->name) == 0)
                                      continue ;
                                    else
                                        {
                                         nt = get_frm_ptr(curr->name)
;
                                         if(nt->qualifier_no == qual_n
o)
                                           {
                                            correct = YES ;
                                            p = nt ;
                                           }
                                        }
                                   }
                               }
                           status_window("") ;
                           if(correct)
                             {
                              clear_windows() ;
                              window(4,3,38,23) ;
                              as_qualifier = -1 ;
                              disp_info_qualifier(q,p->name,&as_qualif
ier) ;
                             }
                          }
                      break ; /* case 'R' */
              case 'P' : if(!display_valid)
                           break ;
                         if(q->no_parents == 0)
                           break ;
                         if(q->no_parents == 1)
                           {
```

```
                        clear_windows() ;
                        p = get_frm_ptr(q->root->name) ;
                        q = qual_name_node(name_get(q->root->name)) ;
                        as_qualifier = -1 ;
                        p = disp_info_qualifier(q,p->name,&as_qualifi
er) ;
                      }
                    else
                        {
                         int correct = NO, i ;
                         char q_no[20] ;
                         struct name_tree *nt ;
                         struct parent_list *curr ;
                         window(1,1,80,25) ;
                         status_window("") ;
                         gotoxy(1,25) ;
                         cprintf("Enter qualifier #") ;
                         temp_txt = get_prob_txt(wherex(),wherey(),8
0-wherex()-2,1) ;
                         if(temp_txt == NULL)
                           break ;
                         if(strlen(temp_txt) != 0)
                           strcpy(q_no,temp_txt) ;
                         free(temp_txt) ;
                         qual_no = atoi(q_no) ;
                         if(qual_no < 1)
                            correct = NO ;
                         else
                             {
                              for(i = 0, curr = q->root ; i < q->no_
parents && curr ; ++i, curr = curr->next)
                                  {
                                   nt = get_frm_ptr(curr->name) ;
                                   if(nt->qualifier_no == qual_no)
                                     {
                                      correct = YES ;
                                      p = nt ;
                                      break ;
                                     }
                                  }
                             }
                         status_window("") ;
                         if(correct)
                           {
                            clear_windows() ;
                            window(4,3,38,23) ;
                            q = qual_name_node(name_get(p->name)) ;
                            as_qualifier = -1 ;
                            disp_info_qualifier(q,p->name,&as_qualif
ier) ;
                           }
                        }
                    break ; /* case 'P' */
          case 'X'   :
          case ENTER :
          case ESC   : status_window("") ;
                       clear_windows() ;
                       window(ti.winleft,ti.wintop,ti.winright,ti.win
bottom) ;
```

```
                        textattr(ti.attribute) ;
                        gotoxy(ti.curx,ti.cury) ;
                        return ;
                }
    } while(1) ;
}
int show_super_attributes(struct qual_name *q)  /* q is "qualifier_tre
e" */
{                                               /* when it's called fi
rst */
 int finish = NO ;
 if(curr_qualifier_no == 1)
   {
    clear_windows() ;
    status_window("No qualifier's Entered Yet thus No Super Objects.
   Press Any Key\a") ;
    fflush(stdin) ;
    getch() ;
    status_window("") ;
    return YES ;
   }
 if(q != NULL)
   {
    finish = show_super_attributes(q->left) ;
    if(!finish)
      {
       if(q->no_parents == 0)
         {
          int key ;
          clear_windows() ;
          window(4,3,38,23) ;
          textcolor(BLACK) ;
          textbackground(WHITE) ;
          cprintf("Attribute name :\n\r") ;
          textbackground(BLACK) ;
          textcolor(WHITE) ;
          cprintf("   %s", q->name) ;
          disp_attr_info(q) ;
          textbackground(YELLOW) ;
          status_window("Press any key to continue.  .....<ESC> to Exi
t") ;
          fflush(stdin) ;
          key = getch() ;
          if(key == EXTENDED)
            getch() ;
          if(key == ESC)
            finish = YES ;
         }
      }
    if(!finish)
      finish = show_super_attributes(q->right) ;
   }
 return finish ;
}
int show_each_attributes(struct qual_name *q)  /* q is "qualifier_tree
" */
{                                              /* when it's called fi
rst */
 int finish = NO ;
```

```
 if(curr_qualifier_no == 1)
   {
    clear_windows() ;
    status_window("No qualifier's Entered Yet thus No Objects.      Pre
ss Any Key\a") ;
    fflush(stdin) ;
    getch() ;
    status_window("") ;
    return YES ;
   }
 if(q != NULL)
   {
    finish = show_each_attributes(q->left) ;
    if(!finish)
      {
       int key ;
       clear_windows() ;
       window(4,3,38,23) ;
       textcolor(BLACK) ;
       textbackground(WHITE) ;
       cprintf("Attribute name :\n\r") ;
       textbackground(BLACK) ;
       textcolor(WHITE) ;
       cprintf("   %s", q->name) ;
       disp_attr_info(q) ;
       textbackground(YELLOW) ;
       status_window("Press any key to continue.  .....<ESC> to Exit")
;
       fflush(stdin) ;
       key = getch() ;
       if(key == EXTENDED)
         getch() ;
       if(key == ESC)
         finish = YES ;
      }
    if(!finish)
      finish = show_each_attributes(q->right) ;
   }
 return finish ;
}
struct name_tree * disp_info_qualifier(struct qual_name *q, char *name
, int *as_a)
{
 struct text_info ti ;
 struct name_tree *p ;
 struct qual_name *q1 ;
 gettextinfo(&ti) ;
 *as_a = exist_frame(name, &p, &q1, PARENT) ;
 q = q1 ;
 switch(*as_a)
       {
        case PARENT     :
        case PRNT_CHILD :
        case CHLD       : p = get_frm_ptr(name) ;
                          disp_frame(get_nm_qualifier(name)) ;
                          disp_attr_info(q) ;
                          break ;
        case PART_CHILD :
        case PART_PRNT  : p = NULL ;
```

```
                        window(4,3,38,23) ;
                        clrscr() ;
                        textcolor(WHITE) ;
                        textbackground(BLACK) ;
                        cprintf("%s\n\r",name) ;
                        textattr(ti.attribute) ;
                        cprintf("   No such attribute or qualifier")
;
                        disp_attr_info(q) ;
                        break ;
        }
 status_window("") ;
 window(ti.winleft,ti.wintop,ti.winright,ti.winbottom) ;
 textattr(ti.attribute) ;
 gotoxy(ti.curx,ti.cury) ;
 return p ;
}
void disp_attr_info(struct qual_name *q)
{
 int i ;
 struct text_info ti ;
 struct name_tree *p ;
 gettextinfo(&ti) ;
 window(41,3,77,23) ;
 if(q->no_parents)
   {
    struct parent_list *curr ;
    textcolor(BLACK) ;
    textbackground(WHITE) ;
    cprintf("Present in following qualifier(s) :\n\r") ;
    textcolor(WHITE) ;
    textbackground(BLACK) ;
    for(curr = q->root, i=0 ; curr ; curr = curr->next)
       {
        p = get_frm_ptr(curr->name) ;
        cprintf("  %2d.  (#%2d) %s\n\r",++i, p->qualifier_no, p->name)
;
       }
   }
 if(q->no_links)
   {
    struct name_list *curr ;
    if(q->no_parents)
      cprintf("\n\r") ;
    textcolor(BLACK) ;
    textbackground(WHITE) ;
    cprintf("Related qualifiers :\n\r") ;
    textcolor(WHITE) ;
    textbackground(BLACK) ;
    for(curr=q->head, i=0 ; curr ; curr = curr->next)
       {
        p = get_frm_ptr(curr->name) ;
        cprintf("  %2d.  (#%2d) %s\n\r",++i, p->qualifier_no, p->name)
;
       }
   }
 window(ti.winleft,ti.wintop,ti.winright,ti.winbottom) ;
 textattr(ti.attribute) ;
 gotoxy(ti.curx,ti.cury) ;
```

```
 return ;
}
struct qual_name *qual_name_node(char *q_name)
{
 struct qual_name *curr ;
 int test ;
 for(curr=qualifier_tree ; curr ;)
    {
     if((test = stricmp(q_name,curr->name)) > 0)
        curr = curr->right ;
     else if(test < 0)
            curr = curr->left ;
          else
              break ;
    }
 return curr ;
}
int find_in_object(void)
{
 int correct = YES, result = NO ;
 char *txt = NULL ;
 struct text_info ti ;
 char *in_object = NULL, *what_object = NULL ;
 int x,y ;
 struct qual_name *p = NULL ;
 struct name_list *relations = NULL ;
 if(curr_qualifier_no == 1)
   {
    clear_windows() ;
    status_window("No qualifier's Entered Yet thus No Paths.     Press
 Any Key\a") ;
    fflush(stdin) ;
    getch() ;
    status_window("") ;
    return YES ;
   }
 gettextinfo(&ti) ;
 clear_windows() ;
 do
   {
    correct = YES ;
    status_window("Enter Destination Object") ;
    window(4,3,38,23) ;
    txt = get_ch_txt(1,1,35,20) ;
    if(txt == NULL)
      return NO ;
    trim(txt) ;
    if(strlen(txt) == 0)
      {
       free(txt) ;
       return NO ;
      }
    p = get_object(txt, qualifier_tree) ;
    if(p == NULL)
      {
       int choice ;
       status_window("No such object in the KB,  Retry ? (Y/N)") ;
       do
         {
```

```
            fflush(stdin) ;
            choice = toupper(getch()) ;
           } while(!(choice == 'N' || choice == 'Y')) ;
         free(txt) ;
         txt = NULL ;
         if(choice == 'N')
          goto finish ;
         else
             correct = NO ;
       }
     else
          {
           what_object = dup_str(txt) ;
           free(txt) ;
           txt = NULL ;
           window(41,3,77,23) ;
           textcolor(BLACK) ;
           cprintf("DESTINATION : ") ;
           textcolor(WHITE) ;
           cprintf("%s\n\r",what_object) ;
           x = wherex() ;
           y = wherey() ;
          }
   } while(!correct) ;
do
   {
    correct = YES ;
    status_window("Enter Source Object") ;
    window(4,3,38,23) ;
    txt = get_ch_txt(1,1,35,20) ;
    if(txt == NULL)
       {
        int choice ;
        correct = NO ;
        status_window("Retry ? (Y/N)") ;
        do
          {
           fflush(stdin) ;
           choice = toupper(getch()) ;
          } while(!(choice == 'N' || choice == 'Y')) ;
        if(choice == 'N')
          {
           free(what_object) ;
           goto finish ;
          }
       }
     else
          {
           trim(txt) ;
           if(strlen(txt) == 0)
             {
              int choice ;
              free(txt) ;
              txt = NULL ;
              correct = NO ;
              status_window("Retry ? (Y/N)") ;
              do
                {
                 fflush(stdin) ;
```

```
                choice = toupper(getch()) ;
               } while(!(choice == 'N' || choice == 'Y')) ;
             if(choice == 'N')
               {
                free(what_object) ;
                goto finish ;
               }
            }
          else
              {
               if(stricmp(txt, what_object) == 0)
                 {
                  status_window("Source & Destination Objects are same.
  ..Press any key") ;
                  fflush(stdin) ;
                  getch() ;
                  correct = NO ;
                  status_window("") ;
                  continue ;
                 }
               p = get_object(txt, qualifier_tree) ;
               if(p == NULL)
                 {
                  int choice ;
                  status_window("No such object in the KB,  Retry ? (Y/
N)") ;
                  do
                    {
                     fflush(stdin) ;
                     choice = toupper(getch()) ;
                    } while(!(choice == 'N' || choice == 'Y')) ;
                  free(txt) ;
                  txt = NULL ;
                  if(choice == 'N')
                    {
                     free(what_object) ;
                     goto finish ;
                    }
                  else
                      {
                       correct = NO ;
                       continue ;
                      }
                 }
               else
                   {
                    in_object = dup_str(txt) ;
                    free(txt) ;
                    txt = NULL ;
                    window(41,3,77,23) ;
                    textcolor(BLACK) ;
                    gotoxy(x,y) ;
                    cprintf("\n\rSOURCE : ") ;
                    textcolor(WHITE) ;
                    cprintf("%s\n\r",in_object) ;
                    x = wherex() ;
                    y = wherey() ;
                   }
              }
```

```
        }
    } while(!correct) ;
  status_window("Enter relation(s) for search (<ENTER> or <ESC> for gen
eral search)") ;
  window(4,3,39,23) ;
  txt = get_cn_txt(1,1,35,20) ;
  if(txt == NULL)
    relations = NULL ;
  else
      {
       trim(txt) ;
       if(strlen(txt) == 0)
         {
          relations = NULL ;
         }
       else
           {
            char *temp ;
            struct name_list *n = NULL, *tail = NULL ;
            temp = strtok(txt,", ") ;
            while(temp)
                 {
                  if(relations == NULL)
                    {
                     n = get_link_node() ;
                     n->name = dup_str(temp) ;
                     relations = tail =  n ;
                    }
                  else
                      {
                       struct name_list *curr ;
                       for(curr=relations ; curr ; curr = curr->next)
                          {
                           if(stricmp(temp,curr->name) == 0)
                             break ;
                          }
                       if(curr == NULL)
                         {
                          n = get_link_node() ;
                          n->name = dup_str(temp) ;
                          tail->next = n ;
                          tail = n ;
                         }
                      }
                  temp = strtok('\0',", ") ;
                 }
           }
       free(txt) ;
       txt = NULL ;
      }
  window(41,3,77,23) ;
  textcolor(BLACK) ;
  gotoxy(x,y) ;
  cprintf("\n\rRELATION(S) : ") ;
  if(relations == NULL)
    {
     textcolor(CYAN) ;
     cprintf("GENERAL SEARCH") ;
    }
```

```
 else
     {
      struct name_list *q = relations ;
      textcolor(WHITE) ;
      cprintf("%s",q->name) ;
      q = q->next ;
      for(; q ; q=q->next)
          cprintf(", %s",q->name) ;
     }
 window(4,3,38,23) ;
 clrscr() ;
 status_window("Please wait ..... Iam searching") ;
 res_stack = get_stack() ;
 init_objects(qualifier_tree) ;
 result = find_out(what_object, in_object, relations) ;
 if(result == NO)
   {
    status_window("No link between the Source and Destination.    ...Pr
ess any key") ;
    fflush(stdin) ;
    getch() ;
    status_window("") ;
   }
 else
     {
      status_window("Path between the Source and Destination.    ...Pre
ss any key") ;
      window(4,3,71,23) ;
      path_prnt(res_stack) ;
      fflush(stdin) ;
      getch() ;
      status_window("") ;
     }
 if(result)
   res_stack = NULL ;
 free(what_object) ;
 free(in_object) ;
 list_dispose(relations);
 relations = NULL ;
 finish :
        window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
        textattr(ti.attribute) ;
        gotoxy(ti.curx, ti.cury) ;
        return result ;
}
struct qual_name *get_object(char *name, struct qual_name *root)
{
 struct qual_name *curr = root ;
 int test ;
 for(; curr ;)
    {
     if((test = stricmp(name,curr->name)) > 0)
       curr = curr->right ;
     else if(test < 0)
            curr = curr->left ;
          else
              break ;
    }
  return curr ;
```

```
}
void list_dispose(struct name_list *p)
{
 if(p)
   {
    list_dispose(p->next) ;
    free(p) ;
    p = NULL ;
   }
 return ;
}
void init_objects(struct qual_name *p)
{
 if(p)
   {
    p->traversed = NO ;
    init_objects(p->left) ;
    init_objects(p->right) ;
   }
 return ;
}
void path_prnt(STACK *S)
{
 int i ;
 char *part_res ;
 for(i = 0 ; !empty_stack(S) ;)
    {
     part_res = pop(S) ;
     textcolor(BLACK) ;
     textbackground(WHITE) ;
     cprintf("%2d.  %s\n\r", ++i, part_res) ;
     free(part_res) ;
     part_res = NULL ;
    }
 return ;
}
int find_out(char *what, char *in, struct name_list *relations)
{
 int no_childs, found = NO ;
 char *qual_verb = NULL ;
 struct qual_name *q = NULL ;
 struct name_list *curr = NULL ;
 FRAME *p = NULL ;
 q = get_object(in,qualifier_tree) ;
 if(q->traversed)
   return NO ;
 else
    q->traversed = YES ;
 for(curr = q->head ; curr && !found ; curr = curr->next)
    {
     if(relations == NULL)
       qual_verb = dup_str(curr->name) ;
     else
         {
          struct name_list *r = NULL ;
          for(r = relations ; r ; r = r->next)
             {
              qual_verb = get_char_array(strlen(in)+strlen(r->name)+3)
 ;
```

```
                sprintf(qual_verb,"%s %s",in,r->name) ;
                if(stricmp(qual_verb,curr->name) != 0)
                  {
                   free(qual_verb) ;
                   qual_verb = NULL ;
                   continue ;
                  }
                else
                    break ;
               }
            if(qual_verb == NULL)
              continue ;
           }
      p = get_nm_qualifier(qual_verb) ;
      for(no_childs = 0 ; no_childs < p->no_childs && !found ; ++no_chi
lds)
          {
           if(stricmp(what,p->childs[no_childs]) == 0)
             {
              char *path ;
              found = YES ;
              path = get_char_array(strlen(qual_verb) + strlen(p->childs
[no_childs]) + 3) ;
              sprintf(path,"%s %s",qual_verb,p->childs[no_childs]) ;
              push(path,res_stack) ;
              free(path) ;
             }
          }
      if(!found)
        {
         for(no_childs = 0 ; no_childs < p->no_childs && !found ; ++no_
childs)
             {
              found = find_out(what,p->childs[no_childs],relations) ;
              if(found)
                {
                 char *path ;
                 path = get_char_array(strlen(qual_verb) + strlen(p->chi
lds[no_childs]) + 3) ;
                 sprintf(path,"%s %s",qual_verb,p->childs[no_childs]) ;
                 push(path,res_stack) ;
                 free(path) ;
                }
             }
        }
      free(qual_verb) ;
      qual_verb = NULL ;
     }
 return found ;
}
int get_value(void)
{
 int correct = YES, result = NO ;
 char *txt = NULL ;
 struct text_info ti ;
 char *in_object = NULL, *relation = NULL ;
 int x,y ;
 struct qual_name *p = NULL ;
 if(curr_qualifier_no == 1)
```

```
   {
    clear_windows() ;
    status_window("No qualifier's Entered Yet thus No Queries.      Pre
ss Any Key\a") ;
    fflush(stdin) ;
    getch() ;
    status_window("") ;
    return YES ;
   }
 gettextinfo(&ti) ;
 clear_windows() ;
 do
   {
    correct = YES ;
    status_window("Enter Object name") ;
    window(4,7,31,23) ;
    txt = get_ch_txt(1,1,35,20) ;
    if(txt == NULL)
      goto finish ;
    trim(txt) ;
    if(strlen(txt) == 0)
      {
       free(txt) ;
       goto finish ;
      }
    p = get_object(txt, qualifier_tree) ;
    if(p == NULL)
      {
       int choice ;
       status_window("No such object in the KB,  Retry ? (Y/N)") ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'N' || choice == 'Y')) ;
       free(txt) ;
       txt = NULL ;
       if(choice == 'N')
        goto finish ;
       else
           correct = NO ;
      }
    else
        {
         in_object = dup_str(txt) ;
         free(txt) ;
         txt = NULL ;
         window(41,3,77,23) ;
         textcolor(BLACK) ;
         cprintf("OBJECT : ") ;
         textcolor(WHITE) ;
         cprintf("%s\n\r",in_object) ;
         x = wherex() ;
         y = wherey() ;
        }
   } while(!correct) ;
 do
   {
    correct = YES ;
```

```
    status_window("Enter relation Name") ;
    window(4,3,30,23) ;
    txt = get_ch_txt(1,1,35,20) ;
    if(txt == NULL)
      {
       int choice ;
       correct = NO ;
       status_window("Retry ? (Y/N)") ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'N' || choice == 'Y')) ;
       if(choice == 'N')
         {
          free(in_object) ;
          goto finish ;
         }
      }
    else
        {
         trim(txt) ;
         if(strlen(txt) == 0)
           {
            int choice ;
            free(txt) ;
            txt = NULL ;
            correct = NO ;
            status_window("Retry ? (Y/N)") ;
            do
              {
               fflush(stdin) ;
               choice = toupper(getch()) ;
              } while(!(choice == 'N' || choice == 'Y')) ;
            if(choice == 'N')
              {
               free(in_object) ;
               goto finish ;
              }
           }
         else
             {
              char *temp = NULL ;
              temp = strtok(txt,", ") ;
              relation = dup_str(temp) ;
              free(txt) ;
              txt = NULL ;
              window(41,3,77,23) ;
              textcolor(BLACK) ;
              gotoxy(x,y) ;
              cprintf("\n\rRELATION : ") ;
              textcolor(WHITE) ;
              cprintf("%s\n\r",relation) ;
             }
        }
  } while(!correct) ;
window(4,3,51,23) ;
clrscr() ;
status_window("Please wait ..... Iam searching") ;
```

```
 init_objects(qualifier_tree) ;
 result = value_find(in_object, relation,1) ;
 if(result == NO)
   {
    status_window("Cannot find any Value.     ...Press any key\a") ;
    fflush(stdin) ;
    getch() ;
    status_window("") ;
   }
 clear_windows() ;
 free(in_object) ;
 free(relation) ;
 finish :
         window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
         textattr(ti.attribute) ;
         gotoxy(ti.curx, ti.cury) ;
         return result ;
}
int value_find(char *object, char *relation, long int tries)
{
 int no_childs, found = NO ;
 char *qual_verb = NULL ;
 struct qual_name *q = NULL ;
 struct name_list *curr = NULL ;
 FRAME *p = NULL ;
 if(tries == 1) /* i.e. First CALL */
   {
    ++tries ;
    q = get_object(object,qualifier_tree) ;
    if(q->traversed)
      return NO ;
    else
        q->traversed = YES ;
    qual_verb = get_char_array(strlen(object)+strlen(relation)+3) ;
    sprintf(qual_verb,"%s %s",object,relation) ;
    for(curr = q->head ; curr && ! found ; curr = curr->next)
       {
        if(stricmp(qual_verb,curr->name) == 0)
          {
           found = YES ;
           p = get_nm_qualifier(qual_verb) ;
           free(qual_verb) ;
           qual_verb = NULL ;
           window(4,3,39,23) ;
           textcolor(WHITE) ;
           textbackground(BLACK) ;
           cprintf("VALUE(s) :\n\r") ;
           textbackground(YELLOW) ;
           textcolor(CYAN) ;
           for(no_childs = 0 ; no_childs < p->no_childs ; ++no_childs)
              cprintf("%2d.  %s\n\r",no_childs+1,p->childs[no_childs])
 ;
           status_window("Value is.......   Press any key to continue.
......") ;
           fflush(stdin) ;
           getch() ;
           status_window("") ;
           return found ;
          }
```

```
        }
     free(qual_verb) ;
     qual_verb = NULL ;
     qual_verb = get_char_array(strlen(object)+strlen("instance")+3) ;
     sprintf(qual_verb,"%s instance",object) ;
     for(curr = q->head ; curr ; curr = curr->next)
        {
         if(stricmp(qual_verb,curr->name) == 0)
           {
            found = value_find(qual_verb,relation,tries) ;
            free(qual_verb) ;
            qual_verb = NULL ;
            return found ;
           }
        }
     free(qual_verb) ;
     qual_verb = NULL ;
     return NO ; /* No instance relation */
    }
 else /* Now object passed is "object"+("isa" or "instance") */
     {
      struct name_tree *temp = NULL ;
      qual_verb = name_get(object) ;
      q = get_object(qual_verb,qualifier_tree) ;
      if(q->traversed && tries++ != 2)
        {
         free(qual_verb) ;
         qual_verb = NULL ;
         return NO ;
        }
      else
          q->traversed = YES ;
      p = get_nm_qualifier(object) ;
      for(no_childs = 0 ; no_childs < p->no_childs && !found ; ++no_ch
ilds)
         {
          qual_verb = get_char_array(strlen(p->childs[no_childs])+strl
en(relation)+3) ;
          sprintf(qual_verb,"%s %s",p->childs[no_childs],relation) ;
          temp = get_frm_ptr(qual_verb) ;
          if(temp)
            {
             found = YES ;
             p = get_nm_qualifier(qual_verb) ;
             window(4,3,38,23) ;
             textcolor(WHITE) ;
             textbackground(BLACK) ;
             cprintf("VALUE(s) :\n\r") ;
             textbackground(YELLOW) ;
             textcolor(CYAN) ;
             for(no_childs = 0 ; no_childs < p->no_childs ; ++no_child
s)
                cprintf("%2d.  %s\n\r",no_childs+1,p->childs[no_childs
]) ;
             status_window("Value is.....   Press any key to continu
e.....") ;
             fflush(stdin) ;
             getch() ;
             status_window("") ;
```

```
            return found ;
           }
         else
             {
              free(qual_verb) ;
              qual_verb = NULL ;
             }
        }
      for(no_childs = 0 ; no_childs < p->no_childs && !found ; ++no_ch
ilds)
         {
          qual_verb = get_char_array(strlen(p->childs[no_childs])+strl
en("isa")+5) ;
          sprintf(qual_verb,"%s isa",p->childs[no_childs]) ;
          temp = get_frm_ptr(qual_verb) ;
          if(temp)
            found = value_find(qual_verb,relation,tries) ;
         }
       return found ;
      }
}
void process_frame(FRAME *p)
{
 struct qual_name *q, *s ;
 struct name_tree *n, *r ;
 int i, present ;
 char *name = NULL ;
 name = name_get(p->name) ;
 q = get_object(name, qualifier_tree) ;
 if(q == NULL)
   {
    q = get_qual_node() ;
    q->name = name ;
    qualifier_tree = add_qualifier(q, qualifier_tree) ;
   }
 n = get_name_node() ;
 n->name = dup_str(p->name) ;
 p->qualifier_no = n->qualifier_no = curr_qualifier_no++ ;
 frame_tree = add_frame(n,frame_tree) ;
 no_qual_tree = add_no_qual(n,no_qual_tree) ;
 add_name_to_qualifier(p->name,c) ;
 fill_relation_tree(p) ;
 for(i=0 ; i < p->no_childs ; ++i)
    {
     present = exist_frame(p->childs[i],&r,&s,CHLD) ;
     if(!present)
       {
        s = get_qual_node() ;
        s->name = dup_str(p->childs[i]) ;
        qualifier_tree = add_qualifier(s,qualifier_tree) ;
       }
     qual_parent_add(p->name,s) ;
    }
 n->line_gen = curr_gen ;
 w_frame(p) ;
 return ;
}
void free_FRAME(FRAME *p)
{
```

```
/**** RELATIONS.C ****/


#include "shell.h"
extern FILE *frm_rel ;
extern char file_name[] ;
RELATIONS *relation_tree ;
/* In Other Files */
char *name_get(char *) ;
int get_verb_ind(char *) ;
char *dup_str(char *) ;
struct name_list *get_link_node(void) ;
void status_window(char *) ;
char *get_ch_txt(int, int, int, int) ;
char *trim(char *) ;
/* In This File */
void fill_relation_tree(FRAME *) ;
RELATIONS *get_RELATION_node(void) ;
RELATIONS *add_relation(RELATIONS *, RELATIONS *) ;
RELATIONS *RELATIONS_get(char *) ;
void add_left_relation(char *, RELATIONS *) ;
void add_right_relation(char *, RELATIONS *) ;
int write_relations(void) ;
int w_rel(RELATIONS *) ;
int rel_w(RELATIONS *) ;
int read_relations(void) ;
RELATIONS *read_one_rel(void) ;
struct name_list *left_list_rel(char *) ;
struct name_list *right_list_rel(char *) ;
void display_rel_list(char *, int) ;
void disp_rel_list(struct name_list *head) ;
void show_relation_info(void) ;
int can_left_side(char *, char *) ;
int can_right_side(char *, char *) ;
void fill_relation_tree(FRAME *p)
{
 int i, ind = get_verb_ind(p->name) ;
 RELATIONS *r = RELATIONS_get(p->name+ind) ;
 char *name ;
 if(r == NULL)
   {
    r = get_RELATION_node() ;
    r->attribute = dup_str(p->name+ind) ;
    relation_tree = add_relation(r, relation_tree) ;
   }
 name = name_get(p->name) ;
 add_left_relation(name,r) ;
 for(i = 0 ; i < p->no_childs ; ++i)
    add_right_relation(p->childs[i],r) ;
 return ;
}
RELATIONS *get_RELATION_node(void)
{
 RELATIONS *p ;
 p = (RELATIONS *) malloc(sizeof(RELATIONS)) ;
 if(p == NULL)
   {
    printf("\OUT OF MEMORY,     ABORTING !    ...(get_RELATION_node)")
;
```

```
        fflush(stdin) ;
        getch() ;
        exit(1) ;
      }
   p->attribute = NULL ;
   p->no_left = p->no_right = 0 ;
   p->left_head = p->right_head = NULL ;
   p->left = p->right = NULL ;
   return p ;
}
RELATIONS *add_relation(RELATIONS *p, RELATIONS *root)
{
   if(root == NULL)
     root = p ;
   else
        {
         if(stricmp(p->attribute, root->attribute) > 0)
           root->right = add_relation(p,root->right) ;
         else
             root->left = add_relation(p,root->left) ;
        }
   return root ;
}
RELATIONS *RELATIONS_get(char *attribute)
{
   RELATIONS *curr = relation_tree ;
   int test ;
   for(; curr ;)
      {
       if((test = stricmp(attribute, curr->attribute)) > 0)
         curr = curr->right ;
       else if(test <0)
               curr = curr->left ;
            else
                break ;
      }
   return curr ;
}
void add_left_relation(char *name, RELATIONS *r)
{
   struct name_list *p ;
   if(r->left_head == NULL)
     {
      p = get_link_node() ;
      p->name = dup_str(name) ;
      r->left_head = p ;
      ++r->no_left ;
     }
   else
        {
         struct name_list *curr = r->left_head ;
         int test ;
         if((test = stricmp(name,curr->name)) == 0)
           return ;
         else if(test < 0)
                 {
                  p = get_link_node() ;
                  p->name = dup_str(name) ;
                  p->next = r->left_head ;
```

```
                r->left_head = p ;
                ++r->no_left ;
                return ;
               }
             else
                 {
                  struct name_list *prev ;
                  for(curr=prev=r->left_head ; curr ; prev = curr, curr
= curr->next)
                      {
                       if((test = stricmp(name,curr->name)) == 0)
                         return ;
                       else if(test < 0)
                              break ;
                      }
                  p = get_link_node() ;
                  p->name = dup_str(name) ;
                  prev->next = p ;
                  p->next = curr ;
                  ++r->no_left ;
                  return ;
                 }
      }
}
void add_right_relation(char *name, RELATIONS *r)
{
 struct name_list *p ;
 if(r->right_head == NULL)
   {
    p = get_link_node() ;
    p->name = dup_str(name) ;
    r->right_head = p ;
    ++r->no_right ;
   }
 else
     {
      struct name_list *curr = r->right_head ;
      int test ;
      if((test = stricmp(name,curr->name)) == 0)
        return ;
      else if(test < 0)
             {
              p = get_link_node() ;
              p->name = dup_str(name) ;
              p->next = r->right_head ;
              r->right_head = p ;
              ++r->no_right ;
              return ;
             }
           else
               {
                struct name_list *prev ;
                for(curr=prev=r->right_head ; curr ; prev = curr, curr
 = curr->next)
                    {
                     if((test = stricmp(name,curr->name)) == 0)
                       return ;
                     else if(test < 0)
                            break ;
```

```
                    }
                    p = get_link_node() ;
                    p->name = dup_str(name) ;
                    prev->next = p ;
                    p->next = curr ;
                    ++r->no_right ;
                    return ;
                   }
      }
}
int write_relations(void)
{
 char temp[MAX_LENGTH] ;
 int success = YES ;
 int choice = 'N' ;
 fclose(frm_rel) ;
 sprintf(temp,"%s.REL",file_name) ;
 do
   {
    choice = 'N' ;
    frm_rel = fopen(temp,"w") ;
    if(frm_rel == NULL)
      {
       status_window("Error: Unable to create relation file.. Retry ?
(Y/N)") ;
       fflush(stdin) ;
       do
         {
          choice = toupper(getch()) ;
         } while(!(choice == 'Y' || choice == 'N')) ;
       status_window("") ;
       if(choice == 'N')
         return NO ;
       else
         continue ;
      }
   } while(choice == 'Y') ;
 success = w_rel(relation_tree) ;
 return success ;
}
int w_rel(RELATIONS *p)
{
 int success = YES ;
 if(p!= NULL)
   {
    rel_w(p) ;
    w_rel(p->left) ;
    w_rel(p->right) ;
   }
 return success ;
}
int rel_w(RELATIONS *p)
{
 struct name_list *curr ;
 int success = YES, i ;
 fprintf(frm_rel,"%s\n",p->attribute) ;
 fprintf(frm_rel,"%d\n",p->no_left) ;
 for(i=0, curr = p->left_head ; i < p->no_left && curr ; ++i, curr = c
urr->next)
```

```
    {
     fprintf(frm_rel,"%s\n",curr->name) ;
    }
 fprintf(frm_rel,"%d\n",p->no_right) ;
 for(i=0, curr = p->right_head ; i < p->no_right && curr ; ++i, curr =
 curr->next)
    {
     fprintf(frm_rel,"%s\n",curr->name) ;
    }
 return success ;
}
int read_relations(void)
{
 int success = YES ;
 RELATIONS *r = NULL ;
 do
   {
    r = read_one_rel() ;
    if(r)
      relation_tree = add_relation(r,relation_tree) ;
   } while(r) ;
 return success ;
}
RELATIONS *read_one_rel(void)
{
 int i ;
 char txt[MESG_LENGTH] ;
 RELATIONS *p = NULL ;
 fscanf(frm_rel,"%[^\n]\n",txt) ;
 if(!feof(frm_rel))
   {
    p = get_RELATION_node() ;
    p->attribute = dup_str(txt) ;
    fscanf(frm_rel,"%d\n",&i) ;
    for(; i > 0 ; --i)
       {
        fscanf(frm_rel,"%[^\n]\n",txt) ;
        add_left_relation(txt,p) ;
       }
    fscanf(frm_rel,"%d\n",&i) ;
    for(; i > 0 ; --i)
       {
        fscanf(frm_rel,"%[^\n]\n",txt) ;
        add_right_relation(txt,p) ;
       }
    return p ;
   }
 else
     return NULL ;
}
struct name_list *left_list_rel(char *attribute)
{
 RELATIONS *r = RELATIONS_get(attribute) ;
 if(r == NULL)
   return NULL ;
 else
     return r->left_head ;
}
struct name_list *right_list_rel(char *attribute)
```

```
{
 RELATIONS *r = RELATIONS_get(attribute) ;
 if(r == NULL)
   return NULL ;
 else
     return r->right_head ;
}
void display_rel_list(char *attribute, int left_or_right)
{
 struct name_list *p ;
 textcolor(BLACK) ;
 textbackground(WHITE) ;
 if(left_or_right == LEFT)
   {
    cprintf("The following occur in left side of \"%s\" :\n\r\n\r",att
ribute) ;
    p = left_list_rel(attribute) ;
   }
 else
     {
      cprintf("The following occur in right side of \"%s\" :\n\r\n\r",
attribute) ;
      p = right_list_rel(attribute) ;
     }
 disp_rel_list(p) ;
 return ;
}
void disp_rel_list(struct name_list *head)
{
 struct name_list *curr = head ;
 int i = 1 ;
 textcolor(BLACK) ;
 textbackground(BROWN) ;
 cprintf("(#%d)",i++) ;
 textbackground(CYAN) ;
 cprintf(" %s",curr->name) ;
 for(curr = curr->next ; curr ; curr = curr->next)
    {
     textbackground(BROWN) ;
     cprintf(", (#%d)",i++) ;
     textbackground(CYAN) ;
     cprintf(" %s",curr->name) ;
    }
 cprintf("\n\r") ;
 return ;
}
void show_relation_info(void)
{
 struct text_info ti ;
 char *txt ;
 gettextinfo(&ti) ;
 window(4,3,77,23) ;
 textbackground(BLACK) ;
 clrscr() ;
 status_window("Enter the relation ") ;
 txt = get_ch_txt(1,1,20,2) ;
 if(txt == NULL)
   goto finish ;
 else
```

```
      {
       trim(txt) ;
       if(strlen(txt) == 0)
         {
          free(txt) ;
          goto finish ;
         }
       else
           {
            if(RELATIONS_get(txt))
              {
               int choice ;
               status_window("(L)eft side or (R)ight side\a") ;
               do
                 {
                  fflush(stdin) ;
                  choice = toupper(getch()) ;
                 } while(!(choice == 'L' || choice == 'R')) ;
               clrscr() ;
               display_rel_list(txt, choice == 'L' ? LEFT : RIGHT) ;
               free(txt) ;
               status_window("Press any key to continue") ;
               fflush(stdin) ;
               getch() ;
               status_window("") ;
              }
            else
                {
                 free(txt) ;
                 status_window("No such Relation.    ..Press any key to
continue") ;
                 fflush(stdin) ;
                 getch() ;
                 status_window("") ;
                }
           }
      }
 finish : clrscr() ;
          window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
          textattr(ti.attribute) ;
          gotoxy(ti.curx, ti.cury) ;
          return ;
}
int can_left_side(char *txt, char *attribute)
{
 struct name_list *curr, *head = left_list_rel(attribute) ;
 int test ;
 for(curr = head ; curr ; curr = curr->next)
    {
     if((test = stricmp(txt, curr->name)) == 0)
       return YES ;
     else if(test < 0)
            break ;
    }
 return NO ;
}
int can_right_side(char *txt, char *attribute)
{
 struct name_list *curr, *head = right_list_rel(attribute) ;
```

```
 int test ;
 for(curr = head ; curr ; curr = curr->next)
    {
     if((test = stricmp(txt, curr->name)) == 0)
       return YES ;
     else if(test < 0)
             break ;
    }
 return NO ;
}
```

```
/******* NON-LEARNING MODULE *******/
/********   RULES.H    ********/


 typedef struct {
                 char *qual_name ;
                 int no_choices ;
                 struct choice_list *head ;
                } QUALIFIER ;
typedef struct method {
                        char *method ;
                        int probability ;
                        float result_probability ;
                        struct method *next ;
                        struct method *same_next ;
                       } CHOICE ;
typedef struct {
                 int var_expr ;
                 char *expression ;
                 char *left_part ;
                 char *right_part ;
                 char relation [3] ; /* to accomodate "<>" */
                } EXPRESSION ;
typedef struct choice_list {
                             int choice ; /* -ve choice for NOT choices
 */
                             struct choice_list *next ;
                            } SELECTION ;
typedef struct progs {
                       char *name ;
                       int share_count ;
                       int pass_no ;
                       struct name_list *p_head ;
                       int return_no ;
                       struct name_list *r_head ;
                       struct progs *next ;
                      } EXTPROG ;
union qual_math {
                  QUALIFIER *ql ;
                  EXPRESSION *expr ;
                 } ;
union choice_extprog_qual_math {
                                 CHOICE *ch ;
                                 EXTPROG *extprog ;
                                 QUALIFIER *ql ;
                                 EXPRESSION *expr ;
                                } ;
typedef struct ifs {
                     int fired ;
                     int act_rule_no ;
                     int which ;
                     union qual_math qm ;
                     struct ifs *next ;
                     struct ifs *left, *right ;
                     struct ifs *same_next ;
                    } IF ;
typedef struct thens {
                       int fired ;
                       int act_rule_no ;
```

```
                    int which ;
                    union choice_extprog_qual_math ceqm ;
                    struct thens *next ;
                    struct thens *left, *right ;
                    struct thens *same_next ;
                   } THEN ;
typedef struct rule {
                     int fire_rule_no ;
                     int act_rule_no ;
                     int no_ifs ;
                     int tot_ifs_fired ;
                     IF *rule_if ;
                     int no_thens ;
                     int tot_thens_fired ;
                     THEN *rule_then ;
                     int fired ;
                     struct rule *next, *prev ;
                    } RULE ;
typedef struct var_desc {
                         char *name ;
                         char *txt ;
                         double value ;
                         int initialized ;
                         int displayed ;
                         int val_accepted ;
                         int disp_in_res ;
                         struct var_desc *left, *right ;
                        } VARIABLE ;
typedef struct res_stack {
                          int act_rule_no ;
                          struct res_stack *next ;
                         } ELEMENT_RULE_STACK ;
typedef struct {
                ELEMENT_RULE_STACK *tos, *curr_why ;
               } RULE_STACK ;
```

```
/******** NON-LEARNING MODULE ********/
/********* RULES.C *********/


#include "shell.h"
extern int curr_qualifier_no ;
extern char file_name[40] ;
extern VARIABLE *var_head, *var_tail ;
int read_variables = NO ;
FILE *v_c_fp, *v_c_ndx_fp ;
long int  curr_dat = 1 ;
int parsing_solving = PARSING ;
int curr_rule_no = 1 ;
int curr_choice_no = 1 ;
int curr_prog_no = 1 ;
int new_rules_added = NO ;
IF *q_if_root, *m_if_root ;
THEN *q_then_root, *m_then_root, *o_head_then, *o_tail_then ;
EXTPROG *e_head_then, *e_tail_then ;
RULE *rule_head, *rule_tail ;
VARIABLE *var_root ;
CHOICE *head_method, *tail_method ;
VARIABLE *v_tree ;
/* In Other Files */
void status_window(char *) ;
char *get_char_array(int) ;
void draw(int, int, int, int) ;
void ver_line(int, int, int) ;
char *dup_str(char *) ;
void clear_windows(void) ;
char *trim(char *) ;
void disp_frame(FRAME *) ;
FRAME *get_nm_qualifier(char *) ;
FRAME *get_num_qualifier(int) ;
char * clean_expression(char *) ;
EXPRESSION *if_split_expression(char *) ;
char * get_left_part(char *, int *) ;
int if_parse_expression(EXPRESSION *) ;
int parse_expr(char *) ;
void look_new_variables(EXPRESSION *) ;
void process_var(char *) ;
int exist_variable(char *) ;
VARIABLE *get_variable(char *) ;
VARIABLE *add_variable(VARIABLE *, VARIABLE *) ;
VARIABLE *get_new_variable(char *) ;
VARIABLE *get_VARIABLE_node(void) ;
void vars_read(void) ;
void vars_write(void) ;
void free_var_list(VARIABLE *) ;
char *get_cr_txt(int, int, int, int) ;
char *get_val_var_txt(int, int, int, int) ;
char *get_prob_txt(int, int, int, int) ;
char *input_choices(int, int, int, int) ;
struct name_list *get_link_node(void) ;
void list_dispose(struct name_list *) ;
/* In This File */
IF *if_single_get(int) ;
void disp_q_rule(QUALIFIER *) ;
int add_choice(QUALIFIER *, int, int) ;
```

```
IF *get_ifs(int *) ;
THEN *get_thens(int *) ;
THEN *then_single_get(int) ;
void new_get_rules(void) ;
void discard_rule(RULE *) ;
void if_discard(IF *) ;
void if_single_discard(IF *) ;
void then_discard(THEN *) ;
void then_single_discard(THEN *) ;
void QUL_dispose(QUALIFIER *) ;
void selections_dispose(SELECTION *) ;
void SELECTION_dispose(SELECTION *) ;
void MATH_dispose(EXPRESSION *) ;
void CHOICE_dispose(CHOICE *) ;
void EPR_dispose(EXTPROG *) ;
void new_rule_add(RULE *) ;
void add_rule_list(RULE *) ;
void if_new_rule_add(IF *) ;
void each_if_add(IF *) ;
IF *if_q_add_tree(IF *, IF *) ;
IF *if_m_add_tree(IF *, IF *) ;
void then_new_rule_add(THEN *) ;
void each_then_add(THEN *) ;
THEN *then_q_add_tree(THEN *, THEN *) ;
THEN *then_m_add_tree(THEN *, THEN *) ;
void then_c_add_list(THEN *) ;
void then_e_add_list(EXTPROG *) ;
void add_to_method_list(CHOICE *) ;
void disp_m_rule(EXPRESSION *) ;
void disp_c_rule(CHOICE *) ;
void disp_e_rule(EXTPROG *) ;
IF *get_IF_node(void) ;
THEN *get_THEN_node(void) ;
SELECTION *get_SELECTION_node(void) ;
QUALIFIER *get_QUALIFIER_node(void) ;
RULE *get_RULE_node(void) ;
EXTPROG *get_EXTPROG_node(void) ;
CHOICE *get_CHOICE_node(void) ;
EXPRESSION *get_MATH_node(void) ;
void view_rules(void) ;
RULE *no_rule_get(int) ;
RULE *no_disp_rule(int) ;
void disp_rule(RULE *) ;
void disp_ifs(IF *) ;
void view_if_single(IF *) ;
void disp_thens(THEN *) ;
void view_then_single(THEN *) ;
void disp_all_methods(void) ;
void disp_choice(CHOICE *) ;
CHOICE *get_choice_no(int) ;
void rules_write(void) ;
int one_rule_write(RULE *, FILE *) ;
int ifs_write(IF *, FILE *) ;
int thens_write(THEN *, FILE *) ;
int one_if_write(IF *, FILE *) ;
int one_then_write(THEN *, FILE *) ;
int QUL_write(QUALIFIER *, FILE *) ;
int EPR_write(EXTPROG *, FILE *) ;
int CHC_write(CHOICE *, FILE *) ;
```

```
int MATH_write(EXPRESSION *, FILE *) ;
void read_rules(void) ;
RULE *read_one_rule(FILE *) ;
IF *read_ifs(int, FILE *) ;
THEN *read_thens(int, FILE *) ;
IF *if_one_read(FILE *) ;
THEN *then_one_read(FILE *) ;
QUALIFIER * QUL_read(FILE *) ;
EXTPROG *EPR_read(FILE *) ;
CHOICE *CHC_read(FILE *) ;
EXPRESSION *MATH_read(FILE *) ;
void Copy(char *, char *) ;
void ext_all_disp(void) ;
EXTPROG *get_prog_no(int) ;
EXTPROG *name_ext_prog_get(char *) ;
IF *if_single_get(int key)
{
 char *ch_txt ; /*txt[MAX_LENGTH]*/
 char msg[MESG_LENGTH] ;
 struct text_info ti ;
 int correct = YES, done = NO ;
 int qual_no = 1, x, x1, y1, type_choice = YES, no_choices = 0 ;
 IF *rule_if = NULL ;
 QUALIFIER *p = NULL ;
 EXPRESSION *q = NULL ;
 FRAME *f = NULL ;
 gettextinfo(&ti) ;
 window(43,3,77,23) ;
 switch(key)
        {
         case 'Q' : qual_no = 1 ;
                    if(curr_qualifier_no > 1)
                      f = get_num_qualifier(qual_no) ;
                    else
                        {
                         status_window("No Qualifiers added yet. ....Pr
ess any key.") ;
                         fflush(stdin) ;
                         getch() ;
                         status_window("") ;
                         rule_if = NULL ;
                         break ;
                        }
                    clrscr() ;
                    disp_frame(f) ;
                    x1 = wherex() ;
                    y1 = wherey() ;
                    do
                      {
                       done = NO ;
                       correct = YES ;
                       sprintf(msg,"Previous(%c or %c), Next(%c or %c),
 Qualifier #(N), Qualifier name(M), Choices(C)",27,24,26,25) ;
                       status_window(msg) ;
                       fflush(stdin) ;
                       key = toupper(getch()) ;
                       switch(key)
                             {
                              default : correct = NO ;
```

```
                                        break ;
                                case ESC   :
                                case ENTER : done = YES ;
                                             rule_if = NULL ;
                                             break;
                                case 'N' : window(1,1,80,25) ;
                                           status_window("") ;
                                           gotoxy(1,25) ;
                                           cprintf("Enter qualifier #") ;
                                           fflush(stdin) ;
                                           ch_txt = get_prob_txt(wherex()
, wherey(), 5, 1) ;
                                           if(ch_txt == NULL)
                                             {
                                              correct = NO ;
                                              break ;
                                             }
                                           x = atoi(ch_txt) ;
                                           window(43,3,77,23) ;
                                           if(x < 1 || x >= curr_qualifie
r_no)
                                             {
                                              sprintf(msg,"(%s): No such
qualifier no. in the database. ..Press any key",ch_txt) ;
                                              status_window(msg) ;
                                              fflush(stdin) ;
                                              getch() ;
                                              status_window("") ;
                                              correct = NO ;
                                              break ;
                                             }
                                           else
                                               {
                                                qual_no = x ;
                                                f = get_num_qualifier(qua
l_no) ;

                                                clrscr() ;
                                                disp_frame(f) ;
                                                x1 = wherex() ;
                                                y1 = wherey() ;
                                               }
                                           break ;
                                case 'M' : window(1,1,80,25) ;
                                           status_window("") ;
                                           gotoxy(1,25) ;
                                           cprintf("Enter qualifier name
: ") ;
                                           fflush(stdin) ;
                                           ch_txt = get_ch_txt(wherex(),
wherey(), 50-wherex()-2, 1) ;
                                           if(ch_txt == NULL)
                                             {
                                              correct = NO ;
                                              break ;
                                             }
                                           trim(ch_txt) ;
                                           window(43,3,77,23) ;
                                           if(strlen(ch_txt) == 0)
                                             {
```

```
                                        correct = NO ;
                                        break ;
                                       }
                                     f = get_nm_qualifier(ch_txt) ;
                                     if(f == NULL)
                                       {
                                        sprintf(msg,"(%s): No such
qualifier in database. ..Press any key",ch_txt) ;
                                        status_window(msg) ;
                                        fflush(stdin) ;
                                        getch() ;
                                        correct = NO ;
                                        break ;
                                       }
                                     qual_no = f->qualifier_no ;
                                     clrscr() ;
                                     disp_frame(f) ;
                                     x1 = wherex() ;
                                     y1 = wherey() ;
                                     break ;
                          case EXTENDED : key = getch() ;
                                          switch(key)
                                                 {
                                                  default : correct
 = NO ;
                                                            break ;
                                                  case UP_ARR   :
                                                  case LEFT_ARR : i
f(qual_no == 1)
 {
qual_no = curr_qualifier_no -1 ;
putchar('\a') ;
 }
                                                                  e
lse
 --qual_no ;
                                                                  f
 = get_num_qualifier(qual_no) ;
                                                                  c
lrscr() ;
                                                                  d
isp_frame(f) ;
                                                                  x
1 = wherex() ;
                                                                  y
1 = wherey() ;
                                                                  b
reak ;
                                                  case DOWN_ARR  :
                                                  case RIGHT_ARR :
if(qual_no == curr_qualifier_no - 1)
{
```

```
 qual_no = 1 ;
 putchar('\a') ;
}
else
  ++qual_no ;
f = get_num_qualifier(qual_no) ;
clrscr() ;
disp_frame(f) ;
x1 = wherex() ;
y1 = wherey() ;
break ;
/                                                    } /* switch(key) *
                                      break ;
                                case 'C' : status_window("Enter choice no
(s), NOT+number(s), done <ENTER>") ;
                                           ch_txt = input_choices(x1+2,y1
,2c,c) ;
                                           no_choices = 0 ;
                                           if(ch_txt != NULL)
                                             {
                                              trim(ch_txt) ;
                                              if(strlen(ch_txt) != 0)
                                                {
                                                 char *next_token ;
                                                 p = NULL ;
                                                 next_token = strtok(ch_t
xt," ,+") ;
                                                 if(next_token != NULL)
                                                   {
                                                    if(*next_token == 'N'
)
                                                      type_choice = NO ;
                                                    else
                                                       {
                                                        type_choice = YE
s ;
                                                        x = atoi(next_to
ken) ;
                                                        if(x > 0 && x <=
 f->no_childs)
                                                          {
                                                           p = get_QUALI
FIER_node() ;
                                                           p->qual_name
= dup_str(f->name) ;
                                                           no_choices +=
 add_choice(p,x,type_choice) ;
                                                          }
```

```
                                                    }
                                                  while((next_token = s
trtok('\0'," ,")) != NULL)
                                                    {
                                                     x = atoi(next_t
oken) ;
                                                     if(x > 0 && x <
= f->no_childs)
                                                       {
                                                        if(p == NULL
)
                                                          {
                                                           p = get_Q
UALIFIER_node() ;
                                                            p->qual_n
ame = dup_str(f->name) ;
                                                          }
                                                        no_choices +
= add_choice(p,x,type_choice) ;
                                                       }
                                                    }
                                                  if(p == NULL)
                                                    {
                                                     clrscr() ;
                                                     disp_frame(f) ;
                                                     correct = NO ;
                                                     break ;
                                                    }
                                                  else
                                                      done = YES ;
                                                  rule_if = get_IF_node
() ;
                                                  rule_if->which = QUL
;
                                                  p->no_choices = no_ch
oices ;
                                                  rule_if->qm.ql = p ;
                                                 } /* if(next_token !=
NULL) */
                                               } /* if(strlen(ch_txt) !=
 0) */
                                             } /* if(ch_txt != NULL) */
                                           break ; /* case 'C' */
                                       } /* switch(key) */
                                    if(!done)
                                      correct = NO ;
                                    else
                                        correct = YES ;
                                  } while(!correct) ; /* do */
                                break ; /* case 'Q' */
                        case 'M' : do /* while(!done) ; */
                                   {
                                    done = YES ;
                                    status_window("Enter a logical mathematical expr
esion, [Var]") ;
                                    ch_txt = get_ch_txt(x1+2, y1, 26, 14) ;
                                    if(ch_txt == NULL)
                                      break ;
                                    clean_expression(ch_txt) ;
```

```
                        if(*ch_txt == '\0')
                          {
                           free(ch_txt) ;
                           ch_txt = NULL ;
                           done = NO ;
                           continue ;
                          }
                        q = if_split_expression(ch_txt) ;
                        if(q == NULL)
                          {
                           status_window("Erroneous Expression.    ...Pre
ss any key to retry\a") ;
                           fflush(stdin) ;
                           getch() ;
                           free(ch_txt) ;
                           ch_txt = NULL ;
                           done = NO ;
                           continue ;
                          }
                        correct = if_parse_expression(q) ;
                        if(!correct)
                          {
                           MATH_dispose(q) ;
                           q = NULL ;
                           free(ch_txt) ;
                           ch_txt = NULL ;
                           done = NO ;
                           continue ;
                          }
                        look_new_variables(q) ;
                        rule_if = get_IF_node() ;
                        rule_if->which = MATH ;
                        rule_if->am.expr = q ;
                       } while(!done) ;
                    break ; /* case 'M' */
       } /* switch(key) */
 window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
 textattr(ti.attribute) ;
 return rule_if ;
}
void disp_q_rule(QUALIFIER *p)
{
 SELECTION *curr ;
 FRAME *frm ;
 cprintf(" %s ",p->qual_name) ;
 curr = p->head ;
 if(curr->choice < 0)
   cprintf("NOT ") ;
 frm = get_nm_qualifier(p->qual_name) ;
 cprintf("%s",frm->childs[abs(curr->choice)-1]) ;
 curr = curr->next ;
 for(;curr; curr = curr->next)
    cprintf(" or %s",frm->childs[abs(curr->choice)-1]) ;
 cprintf("\n\r") ;
 return ;
}
int add_choice(QUALIFIER *p, int choice, int type)
{
 SELECTION *curr, *prev, *x ;
```

```
 int unique = YES ;
 x = get_SELECTION_node() ;
 x->choice= type == NO ? -choice : choice ;
 if(p->head == NULL)
   p->head = x ;
 else
     {
      if(choice == p->head->choice)
        {
         unique = NO ;
         return unique ;
        }
      if(abs(p->head->choice) > abs(choice))
        {
         x->next = p->head ;
         p->head = x ;
        }
      else
          {
           for(prev = curr = p->head ; abs(choice) > abs(curr->choice)
 && curr ; prev = curr, curr = curr->next)
               ;
           if(abs(choice) != abs(curr->choice)) /* To remove duplicate
s */
             {
              x->next = prev->next ;
              prev->next = x ;
             }
           else
               unique = NO ;
          }
     }
 return unique ;
}
IF *get_ifs(int *no_ifs)
{
 IF *head = NULL, *tail = NULL, *curr = NULL ;
 int if_y ;
 int key ;
 *no_ifs = 0 ;
 if_y = whereY() ;
 do
   {
    status_window("Qualifier(Q), Math(M), done <ENTER>") ;
    do
      {
       fflush(stdin) ;
       key = toupper(getch()) ;
      } while(!(key == 'Q' || key == 'M' || key == ENTER)) ;
    if(key != ENTER)
      {
       curr = if_single_get(key) ;
       if(curr != NULL)
         {
          if(head == NULL)
            head = tail = curr ;
          else
              {
               tail->next = curr ;
```

```
                tail = curr ;
               }
            curr->act_rule_no = curr_rule_no ;
            gotoxy(1,if_y) ;
            switch(curr->which)
                  {
                   case QUL : disp_q_rule(curr->qm.ql) ;
                              break ;
                   case MATH : disp_m_rule(curr->qm.expr) ;
                               break ;
                  }
            ++(*no_ifs) ;
            if_y = wherey() ;
           } /* if(curr != NULL) */
      } /* if(key != ENTER) */
     else
         break ;
   } while(TRUE) ;
 gotoxy(4,if_y) ;
 return head ;
}
THEN *get_thens(int *no_thens)
{
 THEN *head = NULL, *tail = NULL, *curr = NULL ;
 int then_y ;
 int key ;
 struct text_info ti ;
 gettextinfo(&ti) ;
 *no_thens = 0 ;
 then_y = wherey() ;
 do
   {
    status_window("Qualifier(Q), Math(M), choice(C), External program(
E), done <ENTER>") ;
    do
      {
       fflush(stdin) ;
       key = toupper(getch()) ;
      } while(!(key == 'Q' || key == 'M' || key == 'C' || key == 'E' |
| key == ENTER)) ;
    if(key != ENTER)
      {
       curr = then_single_get(key) ;
       if(curr != NULL)
         {
          if(head == NULL)
            head = tail = curr ;
          else
              {
               tail->next = curr ;
               tail = curr ;
              }
          curr->act_rule_no = curr_rule_no ;
          gotoxy(1,then_y) ;
          switch(curr->which)
                {
                 case QUL : disp_q_rule(curr->ceqm.ql) ;
                            break ;
                 case MATH : disp_m_rule(curr->ceqm.expr) ;
```

```
                                break ;
                        case EPR : disp_e_rule(curr->ceqm.extprog) ;
                                   break ;
                        case CHE : disp_c_rule(curr->ceqm.ch) ;
                                   break ;
                      }
                  ++(*no_thens) ;
                  then_y = wherey() ;
                } /* if(curr != NULL) */
            } /* if(key != ENTER) */
          else
              break ;
        } while(TRUE) ;
 window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
 textattr(ti.attribute) ;
 return head ;
}
THEN *then_single_get(int key)
{
 char *ch_txt ; /*txt[MAX_LENGTH]*/
 char msg[MESG_LENGTH] ;
 struct text_info ti ;
 int correct = YES, done = NO ;
 int qual_no = 1, x, x1, y1, type_choice = YES, no_choices = 0 ;
 THEN *rule_then = NULL ;
 QUALIFIER *p = NULL ;
 EXPRESSION *q = NULL ;
 CHOICE *c = NULL ;
 EXTPROG *e = NULL ;
 FRAME *f = NULL ;
 gettextinfo(&ti) ;
 window(43,3,77,23) ;
 switch(key)
       {
        case 'C' : clrscr() ;
                   if(curr_choice_no > 1)
                     {
                      textcolor(BLACK) ;
                      textbackground(WHITE) ;
                      cprintf("Choices :\n\r") ;
                      textattr(WHITE | (BLACK <<4)) ;
                      disp_all_methods() ;
                      status_window("Choice #(N), New(W)") ;
                      do
                        {
                         key = toupper(getch()) ;
                        } while(!(key == 'N' || key == 'W' || key == E
NTER)) ;

                      if(key == ENTER)
                        break ;
                      if(key == 'N')
                        {
                         window(1,1,80,25) ;
                         status_window("") ;
                         gotoxy(1,25) ;
                         cprintf("Enter Choice #") ;
                         fflush(stdin) ;
                         ch_txt = get_prob_txt(wherex(), wherey(), 4,1
) ;
```

ERRATA

S.No.	Page	Present	Read As
1	3	As the ESS SHELL is designed to highly	As the ESS SHELL is designed to be highly
2	32	fig h	fig 3.9
3	44 to 45	Rules : ¦ Methods or Conclusions	[Not required]
4	51	Test Runs : ¦ Results	[Not required]
5	52	Cost of estimated	Cost of project
6	58	buildin	building
7	69	[x] open_towards [y]	[x] open_towards [z]
8	48-50	In rule #11 to #15 Soil type is cohesive (clay type)	Soil type is Non-cohesive (Sandy)

```
                        if(ch_txt == NULL)
                          break ;
                        else
                            {
                             x = atoi(ch_txt) ;
                             c = get_choice_no(x) ;
                             if(c == NULL)
                               {
                                free(ch_txt) ;
                                ch_txt = NULL ;
                                break ;
                               }
                             else
                                 {
                                  CHOICE *temp = c ;
                                  c = get_CHOICE_node() ;
                                  c->method = dup_str(temp->method) ;
                                  do
                                    {
                                     status_window("") ;
                                     gotoxy(1,25) ;
                                     cprintf("Enter any no. betweeen
0(Absolutely False) & 10(Absolutely True) : ") ;
                                     ch_txt = get_prob_txt(wherex(),w
nerey(),3,1) ;
                                     if(ch_txt == NULL)
                                        break ;
                                     x = atoi(ch_txt) ;
                                    } while(x < 0 || x > 10) ;
                                  if(ch_txt == NULL)
                                    {
                                     free(c->method) ;
                                     free(c) ;
                                     c = NULL ;
                                     rule_then = NULL ;
                                     break ;
                                    }
                                  else
                                      {
                                       c->probability = x ;
                                       rule_then = get_THEN_node() ;
                                       rule_then->which = CHE ;
                                       rule_then->ceqm.ch = c ;
                                       add_to_method_list(c) ;
                                      }
                                 }
                            }
                        break ;
                       }
                   }
                 /* i.e. a New Method is being chosen */
                 status_window("Enter a text") ;
                 clrscr() ;
                 ch_txt = get_ch_txt(2,2,26,14) ;
                 if(ch_txt != NULL)
                   {
                    trim(ch_txt) ;
                    if(strlen(ch_txt) != 0)
                      {
```

```
                        c = get_CHOICE_node() ;
                        c->method = ch_txt ;
                        do
                          {
                           window(1,1,80,25) ;
                           status_window("") ;
                           gotoxy(1,25) ;
                           cprintf("Enter any no. betweeen 0(Absolute
ly False) & 10(Absolutely True) : ") ;
                           ch_txt = get_prob_txt(wherex(),wherey(),3,
1) ;
                           if(ch_txt == NULL)
                              break ;
                           x = atoi(ch_txt) ;
                          } while(x < 0 || x > 10) ;
                        if(ch_txt == NULL)
                          {
                           free(c->method) ;
                           free(c) ;
                           c = NULL ;
                           rule_then = NULL ;
                           break ;
                          }
                        else
                            {
                             c->probability = x ;
                             rule_then = get_THEN_node() ;
                             rule_then->which = CHE ;
                             rule_then->ceqm.ch = c ;
                             add_to_method_list(c) ;
                            }
                       } /* if(strlen(ch_txt) != 0) */
                    } /* if(ch_txt != NULL) */
                  window(43,3,77,23) ;
                  break ; /* case 'C' */
        case 'E' : clrscr() ;
                  if(curr_prog_no > 1)
                    {
                     textcolor(BLACK) ;
                     textbackground(WHITE) ;
                     cprintf("External Program(s) :\n\r") ;
                     textattr(WHITE | (BLACK <<4)) ;
                     ext_all_disp() ;
                     status_window("Program #(N), New(W)") ;
                     do
                       {
                        key = toupper(getch()) ;
                       } while(!(key == 'N' || key == 'W' || key == E
NTER)) ;
                     if(key == ENTER)
                       break ;
                     if(key == 'N')
                       {
                        window(1,1,80,25) ;
                        status_window("") ;
                        gotoxy(1,25) ;
                        cprintf("Enter Program #") ;
                        fflush(stdin) ;
                        ch_txt = get_prob_txt(wherex(), wherey(), 4,1
```

```
) ;
                        if(ch_txt == NULL)
                          break ;
                        else
                            {
                             x = atoi(ch_txt) ;
                             e = get_prog_no(x) ;
                             if(e == NULL)
                               {
                                free(ch_txt) ;
                                ch_txt = NULL ;
                                break ;
                               }
                             else
                                 {
                                  ++e->share_count ;
                                  rule_then = get_THEN_node() ;
                                  rule_then->which = EPR ;
                                  rule_then->ceqm.extprog = e ;
                                  break ;
                                 }
                            }
                       }
                  }
                status_window("Enter the Program's name") ;
                clrscr() ;
                ch_txt = get_ch_txt(2,2,26,14) ;
                if(ch_txt != NULL)
                  {
                   trim(ch_txt) ;
                   if(strlen(ch_txt) != 0)
                     {
                      char *temp = NULL ;
                      struct name_list *n = NULL, *end = NULL ;
                      e = name_ext_prog_get(ch_txt) ;
                      if(e != NULL)
                        {
                         ++e->share_count ;
                         rule_then = get_THEN_node() ;
                         rule_then->which = EPR ;
                         rule_then->ceqm.extprog = e ;
                         free(ch_txt) ;
                         ch_txt = NULL ;
                         break ;
                        }
                      e = get_EXTPROG_node() ;
                      e->share_count = 1 ;
                      e->name = dup_str(ch_txt) ;
                      free(ch_txt) ;
                      ch_txt = NULL ;
                      status_window("Enter the list of variables (l
ike [V1],[V2] or [V1] [V2] ) to be passed") ;
                      ch_txt = get_ch_txt(2,2,26,14) ;
                      if(ch_txt != NULL)
                        {
                         trim(ch_txt) ;
                         if(strlen(ch_txt) != 0)
                           {
                            temp = strtok(ch_txt,", ") ;
```

```
                        while(temp)
                             {
                              if(e->p_head == NULL)
                                {
                                 n = get_link_node() ;
                                 n->name = dup_str(temp) ;
                                 process_var(temp) ;
                                 ++e->pass_no ;
                                 e->p_head = end = n ;
                                }
                              else
                                  {
                                   struct name_list *curr = NUL
L ;

                                   for(curr = e->p_head ; curr
; curr = curr->next)

                                      if(stricmp(temp,curr->nam
e) == 0)

                                        break ;
                                   if(curr == NULL)
                                     {
                                      n = get_link_node() ;
                                      n->name = dup_str(temp) ;
                                      process_var(temp) ;
                                      ++e->pass_no ;
                                      end->next = n ;
                                      end = n ;
                                     }
                                  }
                              temp = strtok('\0',", ") ;
                             } /* while(temp) */
                         free(ch_txt) ;
                         ch_txt = NULL ;
                        } /* if(strlen(ch_txt) != 0) */
                    } /* if(ch_txt != NULL) */
                status_window("Enter the list of variables (l
ike [V1],[V2] or [V1] [V2] ) returned by program") ;
                ch_txt = get_ch_txt(2,2,26,14) ;
                if(ch_txt != NULL)
                  {
                   trim(ch_txt) ;
                   if(strlen(ch_txt) != 0)
                     {
                      temp = strtok(ch_txt,", ") ;
                      while(temp)
                             {
                              if(e->r_head == NULL)
                                {
                                 n = get_link_node() ;
                                 n->name = dup_str(temp) ;
                                 process_var(temp) ;
                                 ++e->return_no ;
                                 e->r_head = end = n ;
                                }
                              else
                                  {
                                   struct name_list *curr = NUL
L ;

                                   for(curr = e->r_head ; curr
```

```
; curr = curr->next)
                                                    if(stricmp(temp,curr->nam
e) == 0)
                                                      break ;
                                                  if(curr == NULL)
                                                    {
                                                     n = get_link_node() ;
                                                     n->name = dup_str(temp) ;
                                                     process_var(temp) ;
                                                     ++e->return_no ;
                                                     end->next = n ;
                                                     end = n ;
                                                    }
                                                 }
                                               temp = strtok('\0',", ") ;
                                              } /* while(temp) */
                                           free(ch_txt) ;
                                           ch_txt = NULL ;
                                          } /* if(strlen(ch_txt) != 0) */
                                       } /* if(ch_txt != NULL) */
                                     rule_then = get_THEN_node() ;
                                     rule_then->which = EPR ;
                                     rule_then->ceqm.extprog = e ;
                                     then_e_add_list(e) ;
                                    } /* if(strlen(ch_txt) != 0) */
                                  } /* if(ch_txt != NULL) */
                                break ;
                     case 'Q' : qual_no = 1 ;
                                if(curr_qualifier_no > 1)
                                  f = get_num_qualifier(qual_no) ;
                                else
                                    {
                                     status_window("No Qualifiers added yet. ....Pr
ess any key.") ;
                                     fflush(stdin) ;
                                     getch() ;
                                     status_window("") ;
                                     rule_then = NULL ;
                                     break ;
                                    }
                                clrscr() ;
                                disp_frame(f) ;
                                x1 = wherex() ;
                                y1 = wherey() ;
                                do
                                  {
                                   done = NO ;
                                   correct = YES ;
                                   sprintf(msg,"Previous(%c or %c), Next(%c or %c),
 Qualifier #(N), Qualifier name(M), Choices(C)",27,24,26,25) ;
                                   status_window(msg) ;
                                   fflush(stdin) ;
                                   key = toupper(getch()) ;
                                   switch(key)
                                         {
                                          default : correct = NO ;
                                                    break ;
                                          case ESC   :
                                          case ENTER : done = NO ;
```

```
                                    rule_then = NULL ;
                                    break;
                          case 'N' : window(1,1,80,25) ;
                                     status_window("") ;
                                     gotoxy(1,25) ;
                                     cprintf("Enter qualifier #") ;
                                     ch_txt = get_prob_txt(wherex()
,wherey(),3,1) ;
                                     window(43,3,77,23) ;
                                     if(ch_txt == NULL)
                                       {
                                        correct = NO ;
                                        break ;
                                       }
                                     x = atoi(ch_txt) ;
                                     if(x < 1 || x >= curr_qualifie
r_no)
                                       {
                                        sprintf(msg,"(%s): No such
qualifier no. in the database. ..Press any key",ch_txt) ;
                                        status_window(msg) ;
                                        fflush(stdin) ;
                                        getch() ;
                                        status_window("") ;
                                        correct = NO ;
                                        break ;
                                       }
                                     else
                                         {
                                          qual_no = x ;
                                          f = get_num_qualifier(qua
l_no) ;
                                          clrscr() ;
                                          disp_frame(f) ;
                                          x1 = wherex() ;
                                          y1 = wherey() ;
                                         }
                                     break ;
                          case 'M' : window(1,1,80,25) ;
                                     status_window("") ;
                                     gotoxy(1,25) ;
                                     cprintf("Enter qualifier name
: ") ;
                                     ch_txt = get_ch_txt(wherex(),w
herey(),30-wherex()-2,1) ;
                                     window(43,3,77,23) ;
                                     if(ch_txt == NULL)
                                       {
                                        correct = NO ;
                                        break ;
                                       }
                                     trim(ch_txt) ;
                                     if(strlen(ch_txt) == 0)
                                       {
                                        correct = NO ;
                                        break ;
                                       }
                                     f = get_nm_qualifier(ch_txt) ;
                                     if(f == NULL)
```

```
                                        {
                                         sprintf(msg,"(%s): No such
qualifier in database. ..Press any key",ch_txt) ;
                                         status_window(msg) ;
                                         fflush(stdin) ;
                                         getch() ;
                                         correct = NO ;
                                         break ;
                                        }
                                    qual_no = f->qualifier_no ;
                                    clrscr() ;
                                    disp_frame(f) ;
                                    x1 = wherex() ;
                                    y1 = wherey() ;
                                    break ;
                          case EXTENDED : key = getch() ;
                                          switch(key)
                                                {
                                                 default : correct
 = NO ;
                                                           break ;
                                                 case UP_ARR   :
                                                 case LEFT_ARR : i
f(qual_no == 1)
 {
qual_no = curr_qualifier_no -1 ;
putchar('\a') ;
 }
                                                                 e
lse
 --qual_no ;
                                                                 f
 = get_num_qualifier(qual_no) ;
                                                                 c
lrscr() ;
                                                                 d
isp_frame(f) ;
                                                                 x
1 = wherex() ;
                                                                 y
1 = wherey() ;
                                                                 b
reak ;
                                                    case DOWN_ARR  :
                                                    case RIGHT_ARR :
if(qual_no == curr_qualifier_no - 1)
{
 qual_no = 1 ;
 putchar('\a') ;
}
```

```
   else
     ++qual_no ;
   f = get_num_qualifier(qual_no) ;
   clrscr() ;
   disp_frame(f) ;
   x1 = wherex() ;
   y1 = wherey() ;
   break ;
   /                                                    } /* switch(key) *
                                                      break ;
   (s), NOT+number(s), done <ENTER>") ;          case 'C' : status_window("Enter choice no
   ,26,14) ;                                      ch_txt = input_choices(x1+2,y1
                                                  no_choices = 0 ;
                                                  if(ch_txt != NULL)
                                                    {
                                                     trim(ch_txt) ;
                                                     if(strlen(ch_txt) != 0)
                                                       {
                                                        char *next_token ;
                                                        p = NULL ;
   xt," ,+") ;                                          next_token = strtok(ch_t

                                                        if(next_token != NULL)
                                                          {
   )                                                       if(*next_token == 'N'

                                                             type_choice = NO ;
                                                           else
                                                               {
   s ;                                                          type_choice = YE
   ken) ;                                                       x = atoi(next_to
    f->no_childs)                                               if(x > 0 && x <=

                                                                  {
   FIER_node() ;                                                   p = get_QUALI
   = dup_str(f->name) ;                                            p->qual_name
    add_choice(p,x,type_choice) ;                                  no_choices +=

                                                                  }
                                                              }
   trtok('\0'," ,")) != NULL)                               while((next_token = s

                                                            {
                                                             x = atoi(next_t
```

```
oken) ;
                                                        if(x > 0 && x <
= f->no_childs)
                                                          {
                                                           if(p == NULL
)
                                                             {
                                                              p = get_Q
UALIFIER_node() ;
                                                              p->qual_n
ame = dup_str(f->name) ;
                                                             }
                                                           no_choices +
= add_choice(p,x,type_choice) ;
                                                          }
                                                       }
                                                    if(p == NULL)
                                                      {
                                                       clrscr() ;
                                                       disp_frame(f) ;
                                                       correct = NO ;
                                                       break ;
                                                      }
                                                    else
                                                        done = YES ;
                                                    rule_then = get_THEN_
node() ;
                                                    rule_then->which = QU
L ;
                                                    p->no_choices = no_ch
oices ;
                                                    rule_then->ceqm.ql =
p ;
                                                   } /* if(next_token !=
NULL) */
                                                } /* if(strlen(ch_txt) !=
 0) */
                                             } /* if(ch_txt != NULL) */
                                          break ; /* case 'C' */
                                   } /* switch(key) */
                              if(!done)
                                correct = NO ;
                              else
                                  correct = YES ;
                             } while(!correct) ; /* do */
                           break ; /* case 'Q' */
                case 'M' : do /* while!done) ; */
                           {
                            done = YES ;
                            status_window("Enter a variable's name within []
") ;
                            ch_txt = get_cn_txt(x1+2,y1,26,14) ;
                            if(ch_txt == NULL)
                              break ;
                            clean_expression(ch_txt) ;
                            if(*ch_txt == '\0')
                              {
                               free(ch_txt) ;
                               ch_txt = NULL ;
```

```
                        done = NO ;
                        continue ;
                       }
                     if(!(*ch_txt == '[' && ch_txt[strlen(ch_txt)-1]
== ']'))
                       {
                        free(ch_txt) ;
                        ch_txt = NULL ;
                        done = NO ;
                        continue ;
                       }
                     else
                         {
                          if(strlen(ch_txt) <= 2)
                            {
                             free(ch_txt) ;
                             ch_txt = NULL ;
                             done = NO ;
                             continue ;
                            }
                          else
                              {
                               int j ;
                               for(j = 1 ; ch_txt[j] ; ++j)
                                  {
                                   if(isalnum(ch_txt[j]) || ch_txt[j]
 == '_')
                                     continue ;
                                   else if(ch_txt[j] == ']' && ch_txt
[j+1] == '\0')
                                         break ;
                                     else
                                         {
                                          free(ch_txt) ;
                                          ch_txt = NULL ;
                                          done = NO ;
                                          break ;
                                         }
                                  }
                               if(done == NO)
                                 continue ;
                               else
                                   {
                                    char *temp = ch_txt ;
                                    status_window("Enter Expression/V
alue you want to assign this variable in this rule") ;
                                    clrscr() ;
                                    ch_txt = get_ch_txt(x1+2,y1,26,14
) ;
                                    if(ch_txt == NULL)
                                      {
                                       free(temp) ;
                                       break ;
                                      }
                                    clean_expression(ch_txt) ;
                                    if(*ch_txt == '\0')
                                      {
                                       free(ch_txt) ;
                                       ch_txt = NULL ;
```

```
                                        free(temp) ;
                                        done = NO ;
                                        continue ;
                                       }
                                      done = parse_expr(ch_txt) ;
                                      if(done)
                                        {
                                         q = get_MATH_node() ;
                                         q->left_part = temp ;
                                         q->right_part = ch_txt ;
                                         q->relation[0] = '=',q->relati
on[1] = '\0' ;

                                         look_new_variables(q) ;
                                         rule_then = get_THEN_node() ;
                                         rule_then->which = MATH ;
                                         rule_then->ceqm.expr = q ;
                                        }
                                      else
                                          {
                                           free(ch_txt) ;
                                           free(temp) ;
                                          }
                                     } /* else */
                                } /* else */
                           } /* else */
                      } while(!done) ;
                    break ;
          } /* switch(key) */
 window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
 textattr(ti.attribute) ;
 return rule_then ;
}
void new_get_rules(void)
{
 int choice = YES, again = NO, discard = NO ;
 RULE *new_rule ;
 IF *if_head ;
 THEN *then_head ;
 int i, no_ifs, no_thens, y1, y2 ;
 struct text_info ti ;
 gettextinfo(&ti) ;
 window(4,3,38,23) ;
 textbackground(BLACK) ;
 clear_windows() ;
 textcolor(BLACK) ;
 textbackground(WHITE) ;
 cprintf("Rule #%d\n\r",curr_rule_no) ;
 gotoxy(wherex(),wherey()) ;
 cprintf("\n\r   IF\n\r") ;
 textattr(WHITE | (BLACK << 4)) ;
 if_head = get_ifs(&no_ifs) ;
 if(no_ifs == 0)
   {
    status_window("No \"IF\" part.     ...Press any key") ;
    fflush(stdin) ;
    getch() ;
    return ;
   }
 textcolor(BLACK) ;
```

```
textbackground(WHITE) ;
cprintf("\n\r   THEN\n\r") ;
textattr(WHITE | (BLACK << 4)) ;
y1 = wherey() ;
new_rule = get_RULE_node() ;
new_rule->no_ifs = no_ifs ;
new_rule->rule_if = if_head ;
do
  {
   then_head = get_thens(&no_thens) ;
   again = NO ;
   if(no_thens == 0)
     {
      status_window("No \"THEN\" part.  ...Again ? (Y/N)") ;
      do
        {
         fflush(stdin) ;
         again = toupper(getch()) ;
        } while(!(again == 'Y' || again == 'N')) ;
      again = again == 'Y' ? YES : NO ;
      if(again)
        {
         for(y2 = wherey(),i = y1 ;i <= y2 ; ++i)
            clreol() ;
         gotoxy(1,y1) ;
        }
      else
          {
           free_var_list(var_head) ;
           var_head = var_tail = NULL ;
           discard = YES ;
          }
     }
  } while(again) ;
if(!discard)
  {
   status_window("Are You sure ? (Y/N)") ;
   do
     {
      fflush(stdin) ;
      choice = toupper(getch()) ;
     } while(!(choice == 'Y' || choice == 'N')) ;
  }
if(choice == 'N' || discard == YES)
  {
   discard_rule(new_rule) ;
   free(new_rule) ;
   new_rule = NULL ;
   free_var_list(var_head) ;
   var_head = var_tail = NULL ;
  }
else
    {
     new_rule->rule_then = then_head ;
     new_rule->no_thens = no_thens ;
     new_rule->fire_rule_no = new_rule->act_rule_no = curr_rule_no++
;
     new_rule_add(new_rule) ;
     new_rules_added += YES ;
```

```
        vars_write() ;
       }
 return ;
}
void discard_rule(RULE *p)
{
 if(p->no_ifs)
    {
     if_discard(p->rule_if) ;
     p->rule_if = NULL ;
    }
 if(p->no_thens)
    {
     then_discard(p->rule_then) ;
     p->rule_then = NULL ;
    }
 return ;
}
void if_discard(IF *p)
{
 if(p != NULL)
    {
     if_single_discard(p) ;
     if_discard(p->next) ;
     free(p) ;
     p = NULL ;
    }
 return ;
}
void if_single_discard(IF *p)
{
 switch(p->which)
        {
         case QUL : QUL_dispose(p->qm.ql) ;
                    p->qm.ql = NULL ;
                    break ;
         case MATH : MATH_dispose(p->qm.expr) ;
                     p->qm.expr = NULL ;
                     break ;
        }
 return ;
}
void then_discard(THEN *p)
{
 if(p != NULL)
    {
     then_single_discard(p) ;
     then_discard(p->next) ;
     free(p) ;
     p = NULL ;
    }
 return ;
}
void then_single_discard(THEN *p)
{
 switch(p->which)
        {
         case QUL : QUL_dispose(p->ceqm.ql) ;
                    p->ceqm.ql = NULL ;
```

```
                    break ;
          case MATH  : MATH_dispose(p->ceqm.expr) ;
                       p->ceqm.expr = NULL ;
                       break ;
          case EPR : EPR_dispose(p->ceqm.extprog) ;
                     p->ceqm.extprog = NULL ;
                     break ;
          case CHE : CHOICE_dispose(p->ceqm.ch) ;
                     p->ceqm.ch = NULL ;
                     break ;
        }
 return ;
}
void QUL_dispose(QUALIFIER *p)
{
 free(p->qual_name) ;
 selections_dispose(p->head) ;
 p->head = NULL ;
 return ;
}
void selections_dispose(SELECTION *p)
{
 if(p != NULL)
   {
    selections_dispose(p->next) ;
    SELECTION_dispose(p) ;
    p = NULL ;
   }
 return ;
}
void SELECTION_dispose(SELECTION *p)
{
 free(p) ;
 return ;
}
void MATH_dispose(EXPRESSION *p)
{
 free(p->expression) ;
 free(p->left_part) ;
 free(p->right_part) ;
 return ;
}
void CHOICE_dispose(CHOICE *p)
{
 free(p) ;
 return ;
}
void EPR_dispose(EXTPROG *p)
{
 if(--p->share_count == 0)
   {
    free(p->name) ;
    free(p) ;
    if(p->pass_no)
      {
       p->pass_no = 0 ;
       list_dispose(p->p_head) ;
       p->p_head = NULL ;
      }
```

```
    if(p->return_no)
      {
       p->return_no = 0 ;
       list_dispose(p->r_head) ;
       p->r_head = NULL ;
      }
   }
 return ;
}
void new_rule_add(RULE *new_rule)
{
 add_rule_list(new_rule) ;
 if_new_rule_add(new_rule->rule_if) ;
 then_new_rule_add(new_rule->rule_then) ;
 return ;
}
void add_rule_list(RULE *p)
{
 if(rule_head == NULL)
   rule_head = rule_tail = p ;
 else
     {
      rule_tail->next = p ;
      p->prev = rule_tail ;
      rule_tail = p ;
     }
 return ;


}
void if_new_rule_add(IF *p)
{
 IF *curr ;
 for(curr = p ; curr ; curr = curr->next)
    each_if_add(curr) ;
 return ;
}
void each_if_add(IF *p)
{
 switch(p->which)
       {
        case ALL : q_if_root = if_q_add_tree(p,q_if_root) ;
                   break ;
        case MATH : m_if_root = if_m_add_tree(p,m_if_root) ;
                    break ;
       }
 return ;
}
IF *if_q_add_tree(IF *p, IF *root)
{
 int test ;
 if(root == NULL)
   root = NULL ;
 else
     {
      if((test = stricmp(p->qm.ql->qual_name,root->qm.ql->qual_name))
> 0)
        root->right = if_q_add_tree(p,root->right) ;
      else if(test < 0)
             root->left = if_q_add_tree(p,root->left) ;
           else /* test == 0 */
```

```
                {
                 IF *curr ;
                 for(curr = root ; curr->same_next ; curr = curr->same_
next)
                    ;
                 curr->same_next = p ;
                }
     }
 return root ;
}
IF *if_m_add_tree(IF *p, IF *root)
{
 int test ;
 if(root == NULL)
   root = NULL ;
 else
     {
      if((test = stricmp(p->qm.expr->left_part,root->qm.expr->left_par
t)) > 0)
        root->right = if_m_add_tree(p,root->right) ;
       else if(test < 0)
              root->left = if_m_add_tree(p,root->left) ;
            else /* test == 0 */
                {
                 IF *curr ;
                 for(curr = root ; curr->same_next ; curr = curr->same_
next)
                    ;
                 curr->same_next = p ;
                }
     }
 return root ;
}
void then_new_rule_add(THEN *p)
{
 THEN *curr ;
 for(curr = p ; curr ; curr = curr->next)
    each_then_add(curr) ;
 return ;
}
void each_then_add(THEN *p)
{
 switch(p->which)
       {
        case NUL : q_then_root = then_q_add_tree(p,q_then_root) ;
                   break ;
        case MATH : m_then_root = then_m_add_tree(p,m_then_root) ;
                    break ;
        case EPR  : then_e_add_list(p) ;
                    break ;
       }
 return ;
}
THEN *then_q_add_tree(THEN *p, THEN *root)
{
 int test ;
 if(root == NULL)
   root = p ;
 else
```

```
     {
      if((test = stricmp(p->ceqm.ql->qual_name,root->ceqm.ql->qual_nam
e)) > 0)
        root->right = then_q_add_tree(p,root->right) ;
      else if(test < 0)
             root->left = then_q_add_tree(p,root->left) ;
           else /* test == 0 */
               {
                THEN *curr ;
                for(curr = root ; curr->same_next ; curr = curr->same_
next)
                   ;
                curr->same_next = p ;
               }

     }
 return root ;
}
THEN *then_m_add_tree(THEN *p, THEN *root)
{
 int test ;
 if(root == NULL)
   root = p ;
 else
     {
      if((test = stricmp(p->ceqm.expr->left_part,root->ceqm.expr->left
_part)) > 0)
        root->right = then_m_add_tree(p,root->right) ;
      else if(test < 0)
             root->left = then_m_add_tree(p,root->left) ;
           else /* test == 0 */
               {
                THEN *curr ;
                for(curr = root ; curr->same_next ; curr = curr->same_
next)
                   ;
                curr->same_next = p ;
               }
     }
 return root ;
}
void then_e_add_list(EXTPROG *p)
{
 if(e_head_then == NULL)
   {
    e_head_then = e_tail_then = p ;
    ++curr_prog_no ;
   }
 else
     {
      e_tail_then->next = p ;
      e_tail_then = p ;
      ++curr_prog_no ;
     }
 return ;
}
void disp_m_rule(EXPRESSION *p)
{
 cprintf(" %s %s %s\n\r",p->left_part,p->relation,p->right_part) ;
 return ;
```

```
}
void disp_c_rule(CHOICE *p)
{
 cprintf(" %s : Probability - %d/10",p->method,p->probability) ;
 if(p->probability == 10)
   cprintf(" (Absolutely True)\n\r") ;
 else if(p->probability == 0)
        cprintf(" (Absolutely False)\n\r") ;
      else
         cprintf("\n\r") ;
 return ;
}
void disp_e_rule(EXTPROG *p)
{
 struct name_list *curr = NULL ;
 cprintf("(") ;
 if(p->r_head)
   {
    curr = p->r_head ;
    cprintf("%s",curr->name) ;
    for(curr = curr->next ; curr ; curr = curr->next)
       cprintf(", %s",curr->name) ;
   }
 cprintf(") = \"%s\" (",p->name) ;
 if(p->p_head)
   {
    curr = p->p_head ;
    cprintf(" s",curr->name) ;
    for(curr = curr->next ; curr ; curr = curr->next)
       cprintf(", %s",curr->name) ;
   }
 cprintf(")\n\r") ;
 return ;
}
IF *get_IF_node(void)
{
 IF *p ;
 p = (IF *) malloc(sizeof(IF)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING   ..(get_IF_node)\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->fired = NO ;
 p->act_rule_no = 0 ;
 p->which = FALSE ;
 p->next = p->left = p->right = p->same_next = NULL ;
 return p ;
}
THEN *get_THEN_node(void)
{
 THEN *p ;
 p = (THEN *) malloc(sizeof(THEN)) ;
 if(p == NULL)
   {
    clrscr() ;
```

```
    printf("Out of memory.    ...ABORTING   ..(get_THEN_node)\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->fired = NO ;
 p->act_rule_no = 0 ;
 p->which = FALSE ;
 p->next = p->left = p->right = p->same_next = NULL ;
 return p ;
}
SELECTION *get_SELECTION_node(void)
{
 SELECTION *p ;
 p = (SELECTION *) malloc(sizeof(SELECTION)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING   ..(get_SELECTION_node)\n")
 ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->choice = 0 ;
 p->next = NULL ;
 return p ;
}
QUALIFIER *get_QUALIFIER_node(void)
{
 QUALIFIER *p ;
 p = (QUALIFIER *) malloc(sizeof(QUALIFIER)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING   ..(get_QUALIFIER_node)\n")
 ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->qual_name = NULL ;
 p->no_choices = 0 ;
 p->head = NULL ;
 return p ;
}
RULE *get_RULE_node(void)
{
 RULE *p ;
 p = (RULE *) malloc(sizeof(RULE)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING   ..(get_RULE_node)\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->fire_rule_no = p->act_rule_no = 0 ;
```

```
 p->no_ifs = 0 ;
 p->tot_ifs_fired = 0 ;
 p->rule_if = NULL ;
 p->no_thens = 0 ;
 p->tot_thens_fired = 0 ;
 p->rule_then = NULL ;
 p->fired = NO ;
 p->next = p->prev = NULL ;
 return p ;
}
EXTPROG *get_EXTPROG_node(void)
{
 EXTPROG *p ;
 p = (EXTPROG *) malloc(sizeof(EXTPROG)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING   ..(get_EXTPROG_node)\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->name = NULL ;
 p->share_count = p->pass_no = p->return_no = 0 ;
 p->p_head = p->r_head = NULL ;
 p->next = NULL ;
 return p ;
}
CHOICE *get_CHOICE_node(void)
{
 CHOICE *p ;
 p = (CHOICE *) malloc(sizeof(CHOICE)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING   ..(get_CHOICE_node)\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->method = NULL ;
 p->probability = 0 ;
 p->result_probability = 0.0 ;
 p->next = p->same_next = NULL ;
 return p ;
}
EXPRESSION *get_MATH_node(void)
{
 EXPRESSION *p ;
 p = (EXPRESSION *) malloc(sizeof(EXPRESSION)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.    ...ABORTING   ..(get_MATH_node)\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->expression = p->left_part = p->right_part = NULL ;
```

```
 p->relation[7] = '\0' ;
 return p ;
}
void view_rules(void)
{
 RULE *p = rule_head ;
 struct text_info ti ;
 int rule_no = 1 ;
 char msg1[MAX_LENGTH], *txt ;
 int key, x ;
 if(curr_rule_no == 1)
   return ;
 gettextinfo(&ti) ;
 textcolor(WHITE) ;
 textbackground(BLACK) ;
 clear_windows() ;
 window(4,3,77,23) ;
 clrscr() ;
 disp_rule(p) ;
 sprintf(msg1,"Previous(%c or %c), Next(%c or %c), Rule #(N), Exit(X)"
,27,24,26,25) ;
 do
   {
    status_window(msg1) ;
    fflush(stdin) ;
    key = toupper(getch()) ;
    switch(key)
          {
           case 'X'   :
           case ESC   :
           case ENTER : status_window("") ;
                        window(ti.winleft, ti.wintop, ti.winright, ti.
winbottom) ;
                        textattr(ti.attribute) ;
                        gotoxy(ti.curx, ti.cury) ;
                        return ;
          case 'N' : window(1,1,80,25) ;
                     status_window("") ;
                     gotoxy(1,25) ;
                     cprintf("Enter Rule #") ;
                     fflush(stdin) ;
                     txt = get_prob_txt(wherex(),wherey(),6,1) ;
                     window(4,3,77,23) ;
                     if(txt == NULL)
                       break ;
                     x = atoi(txt) ;
                     if(x < 1 || x >= curr_rule_no)
                       {
                        char msg2[MESG_LENGTH] ;
                        sprintf(msg2,"(%s): No such Rule no.  ...Pres
s any key",txt) ;
                        status_window(msg2) ;
                        fflush(stdin) ;
                        getch() ;
                       }
                     else
                         {
                          rule_no = x ;
                          clrscr() ;
```

```
                        p = no_disp_rule(rule_no) ;
                       }
                     break ;
          case EXTENDED : switch(key = getch())
                          {
                           case UP_ARR   :
                           case LEFT_ARR : if(rule_no == 1)
                                           {
                                            rule_no = curr_ru
le_no - 1 ;

                                            p = rule_tail ;
                                            putchar('\a') ;
                                           }
                                          else
                                              {
                                               --rule_no ;
                                               p = p->prev ;
                                              }
                                          clrscr() ;
                                          disp_rule(p) ;
                                          break ;
                           case DOWN_ARR  :
                           case RIGHT_ARR : if(rule_no == curr_
rule_no - 1)
                                           {
                                            rule_no = 1 ;
                                            p = rule_head ;
                                            putchar('\a') ;
                                           }
                                           else
                                              {
                                               ++rule_no ;
                                               p = p->next ;
                                              }
                                          clrscr() ;
                                          disp_rule(p) ;
                                          break ;
                          }
                     break ;
          } /* switch(key) */
   } while(TRUE) ;
}
RULE * no_rule_get(int fire_no)
{
 RULE *curr ;
 int search_from ;
 if(fire_no < 1 || fire_no >= curr_rule_no)
   return NULL ;
 search_from = fire_no > (curr_rule_no/2) ? END : BEGIN ;
 if(search_from == BEGIN)
   {
    for(curr = rule_head ; curr ; curr = curr->next)
       {
        if(curr->fire_rule_no == fire_no)
          break ;
        else if(curr->fire_rule_no > fire_no)
               {
                curr = NULL ;
                break ;
```

```
                }
            }
        }
    else
        {
         for(curr = rule_tail ; curr ; curr = curr->prev)
            {
             if(curr->fire_rule_no == fire_no)
               break ;
             else if(curr->fire_rule_no < fire_no)
                    {
                     curr = NULL ;
                     break ;
                    }
            }
        }
 return curr ;
}
RULE *no_disp_rule(int fire_no)
{
 RULE *p ;
 p = no_rule_get(fire_no) ;
 if(p != NULL)
   disp_rule(p) ;
 return p ;
}
void disp_rule(RULE *p)
{
 textcolor(BLACK) ;
 textbackground(WHITE) ;
 cprintf("Rule #%d\n\r\n\r   IF\n\r",p->fire_rule_no) ;
 textattr(WHITE | (BLACK <<4)) ;
 disp_ifs(p->rule_if) ;
 textcolor(BLACK) ;
 textbackground(WHITE) ;
 cprintf("\n\r   THEN\n\r") ;
 textattr(WHITE | (BLACK <<4)) ;
 disp_thens(p->rule_then) ;
 return ;
}
void disp_ifs(IF *head)
{
 IF *curr ;
 for(curr = head ; curr ; curr = curr->next)
   view_if_single(curr) ;
 return ;
}
void view_if_single(IF *p)
{
 switch(p->which)
        {
         case QUL : disp_q_rule(p->qm.ql) ;
                    break ;
         case MATH : disp_m_rule(p->qm.expr) ;
                     break ;
        }
 return ;
}
void disp_thens(THEN *head)
```

```
{
 THEN *curr ;
 for(curr = head ; curr ; curr = curr->next)
   view_then_single(curr) ;
 return ;
}
void view_then_single(THEN *p)
{
 switch(p->which)
        {
         case QUL : disp_q_rule(p->ceqm.ql) ;
                    break ;
         case MATH : disp_m_rule(p->ceqm.expr) ;
                     break ;
         case EPR : disp_e_rule(p->ceqm.extprog) ;
                    break ;
         case CHE : disp_c_rule(p->ceqm.ch) ;
                    break ;
        }
 return ;
}
void disp_all_methods(void)
{
 int i ;
 CHOICE *curr ;
 for(curr = head_method, i = 0 ; curr ; curr = curr->next)
    {
     cprintf(" %2d. ",++i) ;
     disp_choice(curr) ;
    }
 return ;
}
void disp_choice(CHOICE *p)
{
 cprintf("%s\n\r",p->method) ;
 return ;
}
CHOICE *get_choice_no(int choice_no)
{
 if(choice_no < 1 || choice_no >= curr_choice_no)
   return NULL ;
 else
     {
      CHOICE *curr = head_method ;
      int i = 1 ;
      for(; curr && i < choice_no ; curr = curr->next, ++i)
         ;
      return curr ;
     }
}
void add_to_method_list(CHOICE *p)
{
 if(head_method == NULL)
   {
    head_method = tail_method = p ;
    ++curr_choice_no ;
   }
 else
     {
```

```
      CHOICE *curr ;
      for(curr = head_method ; curr ; curr = curr->next)
         {
          if(stricmp(curr->method,p->method) == 0)
            {
             p->same_next = curr->same_next ;
             curr->same_next = p ;
             return ;
            }
         }
      ++curr_choice_no ;
      tail_method->next = p ;
      tail_method = p ;
     }
 return ;
}
void rules_write(void)
{
 int choice, success = YES ;
 char txt[MAX_LENGTH] ;
 FILE *rules_fp ;
 Rule *curr ;
 if(new_rules_added == NO)
   return ;
 sprintf(txt,"%s.RUL",file_name) ;
 rules_fp = fopen(txt,"r") ;
 if(rules_fp != NULL)
   {
    char inp[MESG_LENGTH] , out[MESG_LENGTH] ;
    fclose(rules_fp) ;
    sprintf(inp,"%s.RUL",file_name) ;
    sprintf(out,"%s.RBK",file_name) ;
    copy(inp,out) ;
   }
 else
     fclose(rules_fp) ;
 do /* while(!success) ; */
   {
    sprintf(txt,"%s.RUL",file_name) ;
    do /* while(rules_fp == NULL) ; */
      {
       rules_fp = fopen(txt,"w") ;
       if(rules_fp == NULL)
         {
          sprintf(txt,"Error while (over)writing file \"%s.RUL\".   ..
Retry ? (Y/N)",file_name) ;
          do
            {
             fflush(stdin) ;
             choice = toupper(getch()) ;
            } while(!(choice == 'Y' || choice == 'N')) ;
          status_window("") ;
          if(choice == 'N')
            {
             fclose(rules_fp) ;
             return ;
            }
         }
      } while(rules_fp == NULL) ;
```

```
    fprintf(rules_fp,"%d\n",curr_rule_no) ;
    for(curr = rule_head ; curr && success ; curr = curr->next)
       success = one_rule_write(curr,rules_fp) ;
    if(!success)
      {
       sprintf(txt,"Error while writing file \"%s.RUL\".  ..Retry ? (Y
/N)",file_name) ;
       status_window(txt) ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'Y' || choice == 'N')) ;
       if(choice == 'N')
         {
          char inp[MESG_LENGTH] , out[MESG_LENGTH] ;
          fclose(rules_fp) ;
          sprintf(out,"%s.RUL",file_name) ;
          sprintf(inp,"%s.RBK",file_name) ;
          Copy(inp,out) ;
          return ;
         }
      }
   } while(!success) ;
 new_rules_added = NO ;
 fclose(rules_fp) ;
 return ;
}
int one_rule_write(RULE *p, FILE *fp)
{
 int success = YES ;
 fprintf(fp,"%d %d\n",p->fire_rule_no, p->act_rule_no) ;
 fprintf(fp,"%d\n",p->no_ifs) ;
 success = ifs_write(p->rule_if,fp) ;
 if(success)
   {
    fprintf(fp,"%d\n",p->no_thens) ;
    success = thens_write(p->rule_then,fp) ;
   }
 return success ;
}
int ifs_write(IF *head, FILE *fp)
{
 int success = YES ;
 IF *curr ;
 for(curr = head ; curr && success ; curr = curr->next)
    success = one_if_write(curr,fp) ;
 return success ;
}
int thens_write(THEN *head, FILE *fp)
{
 int success = YES ;
 THEN *curr ;
 for(curr = head ; curr && success ; curr = curr->next)
    success = one_then_write(curr,fp) ;
 return success ;
}
int one_if_write(IF *p, FILE *fp)
{
```

```
 int success = YES ;
 fprintf(fp,"%d\n",p->act_rule_no) ;
 fprintf(fp,"%d\n",p->which) ;
 switch(p->which)
        {
         case QUL : success = QUL_write(p->qm.ql,fp) ;
                    break ;
         case MATH : success = MATH_write(p->qm.expr,fp) ;
                     break ;
        }
 return success ;
}
int one_then_write(THEN *p, FILE *fp)
{
 int success = YES ;
 fprintf(fp,"%d\n",p->act_rule_no) ;
 fprintf(fp,"%d\n",p->which) ;
 switch(p->which)
        {
         case QUL : success = QUL_write(p->ceqm.ql,fp) ;
                    break ;
         case MATH : success = MATH_write(p->ceqm.expr,fp) ;
                     break ;
         case EPR : success = EPR_write(p->ceqm.extprog,fp) ;
                    break ;
         case CHE : success = CHE_write(p->ceqm.ch,fp) ;
                    break ;
        }
 return success ;
}
int QUL_write(QUALIFIER *p, FILE *fp)
{
 int success = YES ;
 SELECTION *curr ;
 fprintf(fp,"%s\n",p->qual_name) ;
 fprintf(fp,"%d\n",p->no_choices) ;
 for(curr = p->head ; curr ; curr = curr->next)
    fprintf(fp,"%d\n",curr->choice) ;
 return success ;
}
int EPR_write(EXTPROG *p, FILE *fp)
{
 int success = YES ;
 struct name_list *curr ;
 fprintf(fp,"%s\n",p->name) ;
 fprintf(fp," %d\n",p->pass_no) ;
 for(curr = p->p_head ; curr ; curr = curr->next)
    fprintf(fp,"%s\n",curr->name) ;
 fprintf(fp," %d\n",p->return_no) ;
 for(curr = p->r_head ; curr ; curr = curr->next)
    fprintf(fp,"%s\n",curr->name) ;
 return success ;
}
int CHE_write(CHOICE *p, FILE *fp)
{
 int success = YES ;
 fprintf(fp,"%s\n",p->method) ;
 fprintf(fp,"%d\n",p->probability) ;
 return success ;
```

```
}
int MATH_write(EXPRESSION *p, FILE *fp)
{
 int success = YES ;
 fprintf(fp,"%s\n",p->left_part) ;
 fprintf(fp,"%s\n",p->relation) ;
 fprintf(fp,"%s\n",p->right_part) ;
 return success ;
}
void read_rules(void)
{
 int success = YES, choice ;
 char txt[MESG_LENGTH] ;
 FILE *rules_fp ;
 Rule *p ;
 int x ;
 do /* while(rules_fp == NULL) ; */
   {
    sprintf(txt,"%s.RUL",file_name) ;
    rules_fp = fopen(txt,"r") ;
    if(rules_fp == NULL)
      {
       sprintf(txt,"Error while opening file \"%s.RUL\".   ..Retry ? (
Y/N)",file_name) ;
       status_window(txt) ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'Y' || choice == 'N')) ;
       status_window("") ;
       if(choice == 'N')
         {
          fcloseall() ;
          clrscr() ;
          exit(1) ;
         }
      }
   } while(rules_fp == NULL) ;
 fscanf(rules_fp,"%d\n",&x) ;
 if(x != 1 && feof(rules_fp))
   {
    sprintf(txt,"Abnormal end of \"%s.RUL\".   ..ABORTING    ....Press
 any key",file_name) ;
    status_window(txt) ;
    fflush(stdin) ;
    getch() ;
    fcloseall() ;
    exit(1) ;
   }
 curr_rule_no = x ;
 for(x = 1 ; x < curr_rule_no && success ; ++x)
    {
     p = read_one_rule(rules_fp) ;
     if(p)
       new_rule_add(p) ;
     else
         success = NO ;
    }
```

```
 if(!success)
   {
    sprintf(txt,"File \"%s.RUL\" corrupted.   ..ABORTING    ....Press
 any key",file_name) ;
    status_window(txt) ;
    fflush(stdin) ;
    getch() ;
    fcloseall() ;
    exit(1) ;
   }
 fclose(rules_fp) ;
 return ;
}
RULE *read_one_rule(FILE *fp)
{
 RULE *p ;
 IF *if_head ;
 THEN *then_head ;
 int x, y, success = YES ;
 p = get_RULE_node() ;
 fscanf(fp,"%d %d\n",&x,&y) ;
 p->fire_rule_no = x ;
 p->act_rule_no = y ;
 fscanf(fp,"%d\n",&x) ;
 p->no_ifs = x ;
 if_head = read_ifs(x,fp) ;
 fscanf(fp,"%d\n",&x) ;
 p->no_thens = x ;
 then_head = read_thens(x,fp) ;
 p->rule_if = if_head ;
 p->rule_then = then_head ;
 if(!success)
   return NULL ;
 else
    return p ;
}
IF *read_ifs(int no_ifs, FILE *fp)
{
 IF *head = NULL, *tail = NULL, *curr = NULL ;
 while(no_ifs--)
      {
       curr = if_one_read(fp) ;
       if(curr != NULL)
         {
          if(head == NULL)
            head = tail = curr ;
          else
              {
               tail->next = curr ;
               tail = curr ;
              }
         }
       else
           {
            char txt[MESG_LENGTH] ;
            sprintf(txt,"File \"%s.RUL\" corrupted.   ..ABORTING  ....
Press any key",file_name) ;
            status_window(txt) ;
            fflush(stdin) ;
```

```
            getch() ;
            fcloseall() ;
            clrscr() ;
            exit(1) ;
           }
      } /* while(no_ifs--) */
 return head ;
}
THEN *read_thens(int no_thens, FILE *fp)
{
 THEN *head = NULL, *tail = NULL, *curr = NULL ;
 while(no_thens--)
      {
       curr = then_one_read(fp) ;
       if(curr != NULL)
         {
          if(head == NULL)
            head = tail = curr ;
          else
              {
               tail->next = curr ;
               tail = curr ;
              }
         }
       else
           {
            char txt[MESG_LENGTH] ;
            sprintf(txt,"File \"%s.RUL\" corrupted.   ..ABORTING  ....
Press any key",file_name) ;
            status_window(txt) ;
            fflush(stdin) ;
            getch() ;
            fcloseall() ;
            clrscr() ;
            exit(1) ;
           }
      } /* while(no_thens--) */
 return head ;
}
IF *if_one_read(FILE *fp)
{
 int success = YES ;
 IF *p ;
 int x ;
 p = get_IF_node() ;
 fscanf(fp,"%d\n",&x) ;
 p->act_rule_no = x ;
 fscanf(fp,"%d\n",&x) ;
 p->which = x ;
 switch(p->which)
       {
        case RUL : p->qm.ql = RUL_read(fp) ;
                   if(p->qm.ql == NULL)
                     success = NO ;
                   break ;
        case MATH : p->qm.expr = MATH_read(fp) ;
                   if(p->qm.expr == NULL)
                     success = NO ;
                   break ;
```

```
        }
  if(!success)
    return NULL ;
  else
      return p ;
}
THEN *then_one_read(FILE *fp)
{
 int success = YES ;
 THEN *p ;
 int x ;
 p = get_THEN_node() ;
 fscanf(fp,"%d\n",&x) ;
 p->act_rule_no = x ;
 fscanf(fp,"%d\n",&x) ;
 p->which = x ;
 switch(p->which)
        {
         case QUL : p->ceqm.ql = QUL_read(fp) ;
                    if(p->ceqm.ql == NULL)
                      success = NO ;
                    break ;
         case MATH : p->ceqm.expr = MATH_read(fp) ;
                    if(p->ceqm.expr == NULL)
                      success = NO ;
                    break ;
         case EPR : p->ceqm.extprog = EPR_read(fp) ;
                    if(p->ceqm.extprog == NULL)
                      success = NO ;
                    break ;
         case CHE : p->ceqm.ch = CHE_read(fp) ;
                    if(p->ceqm.ch == NULL)
                      success = NO ;
                    break ;
        }
  if(!success)
    return NULL ;
  else
      return p ;
}
QUALIFIER * QUL_read(FILE *fp)
{
 int x, success = YES, y ;
 QUALIFIER *p ;
 char txt[MAX_LENGTH] ;
 p = get_QUALIFIER_node() ;
 fscanf(fp,"%[^\n]\n",txt) ;
 p->qual_name = dup_str(txt) ;
 fscanf(fp,"%d\n",&x) ;
 p->no_choices = x ;
 for(x = 0 ; x < p->no_choices ; ++x)
    {
     fscanf(fp,"%d\n",&y) ;
     add_choice(p,y,YES) ;
    }
 if(!success)
   return NULL ;
 else
     return p ;
```

```
}
EXTPROG *EPR_read(FILE *fp)
{
 int success = YES, i ;
 EXTPROG *p ;
 char txt[MAX_LENGTH] ;
 fscanf(fp,"%[^\n]\n",txt) ;
 p = name_ext_prog_get(txt) ;
 if(p == NULL)
   {
    struct name_list *curr = NULL, *tail = NULL ;
    p = get_EXTPROG_node() ;
    p->name = dup_str(txt) ;
    p->share_count = 1 ;
    fscanf(fp,"%d\n",&i) ;
    p->pass_no = i ;
    if(p->pass_no)
      if(v_tree == NULL)
        vars_read() ;
    for(i = 0 ; i < p->pass_no ; ++i)
       {
        fscanf(fp,"%[^\n]\n",txt) ;
        curr = get_link_node() ;
        curr->name = dup_str(txt) ;
        if(p->p_head == NULL)
          p->p_head = tail = curr ;
        else
            {
             tail->next = curr ;
             tail = curr ;
            }
       }
    fscanf(fp,"%d\n",&i) ;
    p->return_no = i ;
    if(p->return_no)
      if(v_tree == NULL)
        vars_read() ;
    for(i = 0 ; i <  p->return_no ; ++i)
       {
        fscanf(fp,"%[^\n]\n",txt) ;
        curr = get_link_node() ;
        curr->name = dup_str(txt) ;
        if( p->r_head == NULL)
           p->r_head = tail = curr ;
        else
            {
             tail->next = curr ;
             tail = curr ;
            }
       }
    then_e_add_list(p) ;
   }
 else
     {
      ++p->share_count ;
      fscanf(fp,"%d\n",&i) ;
      for(i = 0 ; i < p->pass_no ; ++i)
         fscanf(fp,"%[^\n]\n",txt) ;
      fscanf(fp,"%d\n",&i) ;
```

```
      for(i = 0 ; i <  p->return_no ; ++i)
         fscanf(fp,"%[^\n]\n",txt) ;
     }
 if(!success)
   return NULL ;
 else
     return p ;
}
CHOICE *CHC_read(FILE *fp)
{
 int success = YES, x ;
 CHOICE *p ;
 char txt[MAX_LENGTH] ;
 p = get_CHOICE_node() ;
 fscanf(fp,"%[^\n]\n",txt) ;
 p->method = dup_str(txt) ;
 fscanf(fp,"%d\n",&x) ;
 p->probability = x ;
 if(!success)
   return NULL ;
 add_to_method_list(p) ;
 return p ;
}
EXPRESSION *MATH_read(FILE *fp)
{
 int success = YES ;
 EXPRESSION *p ;
 char txt[MAX_LENGTH] ;
 p = get_MATH_node() ;
 fscanf(fp,"%[^\n]\n",txt) ;
 p->left_part = dup_str(txt) ;
 fscanf(fp,"%[^\n]\n",txt) ;
 strcpy(p->relation,txt) ;
 fscanf(fp,"%[^\n]\n",txt) ;
 p->right_part = dup_str(txt) ;
 if(strchr(p->left_part,'[') || strchr(p->right_part,'['))
   {
    if(!read_variables)
      {
       vars_read() ;
       read_variables = YES ;
      }
   }
 if(!success)
   return NULL ;
 else
     return p ;
}
void Copy(char *source, char *dest)
{
 FILE *inp, *out ;
 char c ;
 inp = fopen(source,"rb") ;
 out = fopen(dest,"wb") ;
 if(inp == NULL || out == NULL)
   {
    printf("Error in source or destination file\n") ;
    exit(1) ;
   }
```

```
 c = getc(inp) ;
 while(!feof(inp))
       {
        putc(c,out) ;
        c = getc(inp) ;
       }
 flushall() ;
 fclose(inp) ;
 fclose(out) ;
 return ;
}
EXTPROG *get_prog_no(int prog_no)
{
 if(prog_no < 1 || prog_no >= curr_prog_no)
   return NULL ;
 else
      {
       EXTPROG *curr = e_head_then ;
       int i = 1 ;
       for(; curr && i < prog_no ; curr = curr->next, ++i)
          ;
       return curr ;
      }
}
void ext_all_disp(void)
{
 int i ;
 EXTPROG *curr ;
 for(curr = e_head_then, i = 0 ; curr ; curr = curr->next)
     {
      cprintf(" %2d. ",++i) ;
      disp_e_rule(curr) ;
     }
 return ;
}
EXTPROG *name_ext_prog_get(char *name)
{
 EXTPROG *curr = e_head_then ;
 for(; curr ; curr = curr->next)
     if(stricmp(name,curr->name) == 0)
       break ;
 return curr ;
}
void then_o_add_list(THEN *p)
{
 if(o_head_then == NULL)
   o_head_then = o_tail_then = p ;
 else
      {
       o_tail_then->right = p ;
       o_tail_then = p ;
      }
 return ;
}
```

```
/****** NON_LEARNING MODULE ******/
/******* FIRE.C ********/

#include "shell.h"
#include <dir.h>
#include <process.h>
extern RULE *rule_head ;
extern THEN *a_then_root, *m_then_root ;
extern struct name_tree *frame_tree ;
extern jmp_buf jbuf ;
extern int parsing_solving ;
extern char *expression ;
extern VARIABLE *v_tree ;
CHOICE *head_result, *tail_result ;
RULE_STACK *R_STACK ;
/* In Other Files */
void status_window(char *) ;
char *get_char_array(int) ;
char *dup_str(char *) ;
struct name_tree *get_frm_ptr(char *) ;
void disp_no_qualifier(int) ;
FRAME *get_num_qualifier(int) ;
char *input_choices(int, int, int, int) ;
SELECTION *get_SELECTION_node(void) ;
void selections_dispose(SELECTION *) ;
CHOICE *get_CHOICE_node(void) ;
void disp_rule(RULE *) ;
void next_token(void) ;
void E(double *) ;
void V(double *) ;
void ERROR(int) ;
VARIABLE *get_variable(char *) ;
void update_var_in_res(RULE *) ;
void draw(int, int, int, int) ;
void prepare_var(VARIABLE *) ;
/* In This File */
int is_fired(RULE *) ;
int completely_fired(RULE *) ;
void incr_rule_fired(int) ;
RULE *get_act_rule_no(int) ;
void fire_rules(void) ;
void init_for_firing(void) ;
void single_rule_fire(RULE *) ;
void fired_rule(RULE *) ;
int true_if(IF *) ;
int true_expression(EXPRESSION *) ;
int true_qualifier(QUALIFIER *) ;
THEN *can_fire(QUALIFIER *) ;
THEN *get_then_name(char *) ;
void set_choices(QUALIFIER *) ;
void get_choices(struct name_tree *) ;
SELECTION *add_SELECTION(int, int, SELECTION *) ;
int test_qualifier(QUALIFIER *) ;
void release_result_memory(void) ;
void choice_result_dispose(CHOICE *) ;
void sel_dispose(struct name_tree *) ;
fire_thens(THEN *) ;
void fire_then_single(THEN *) ;
```

```
void fire_MATH_then(EXPRESSION *) ;
void fire_EPR_then(EXTPROG *) ;
void fire_QUL_then(QUALIFIER *) ;
void fire_CHC_then(CHOICE *) ;
void res_print(void) ;
void init_ifs(IF *) ;
void if_init_one(IF *) ;
void init_thens(THEN *) ;
void then_init_one(THEN *) ;
int succ_fired(RULE *) ;
void successful_rules_view(void) ;
int whether_failed_rule(RULE *) ;
void fail_rule_detail(void) ;
void init_vars(VARIABLE *) ;
void res_vars(VARIABLE *,int *) ;
THEN *var_get_then_name(char *) ;
ELEMENT_RULE_STACK *R_node_get_stack(void) ;
RULE_STACK *R_get_stack(void) ;
int rule_empty_stack(RULE_STACK *) ;
init_rule_stack(RULE_STACK *) ;
void prepare_for_why(RULE_STACK *) ;
int why_rule_retrieve_stack(RULE_STACK *, int *) ;
int rule_pop_stack(RULE_STACK *, int *) ;
void rule_push_stack(RULE_STACK *, int) ;
void why_answer(RULE_STACK *) ;
int is_fired(RULE *p)
{
 return p->tot_ifs_fired > 0 ? YES : NO ;
}
int completely_fired(RULE *p)
{
 return p->tot_ifs_fired > p->no_ifs ? YES : NO ;     /* if == p->no_if
s+1 All rules were fired */
}                                                     /* if  > p->no_if
s+1 Forced to terminate  */
void incr_rule_fired(int act_rule_no)
{
 RULE *p = get_act_rule_no(act_rule_no) ;
 ++p->tot_ifs_fired ;
 return ;
}
RULE *get_act_rule_no(int act_rule_no)
{
 RULE *curr ;
 for(curr = rule_head ; curr ; curr = curr->next)
    {
     if(curr->act_rule_no == act_rule_no)
       break ;
    }
 return curr ;
}
void fire_rules(void)
{
 RULE *curr ;
 if(R_STACK == NULL)
   R_STACK = R_get_stack() ;
 init_for_firing() ;
 for(curr = rule_head ; curr ; curr = curr->next)
    {
```

```
      if(completely_fired(curr))
        continue ;
      single_rule_fire(curr) ;
     }
 return ;
}
void init_for_firing(void)
{
 RULE *curr ;
 for(curr = rule_head ; curr ; curr = curr->next)
     {
      curr->tot_ifs_fired = 0 ;
      init_ifs(curr->rule_if) ;
      init_thens(curr->rule_then) ;
     }
 init_vars(v_tree) ;
 return ;
}
void single_rule_fire(RULE *p)
{
 int true_rule_ifs = YES, x ;
 IF *curr ;
 rule_push_stack(R_STACK, p->act_rule_no) ;
 for(curr = p->rule_if ; curr && true_rule_ifs ; curr = curr->next)
     {
      ++p->tot_ifs_fired ;
      curr->fired = YES ;
      true_rule_ifs = true_if(curr) ;
     }
 if(!true_rule_ifs)
   fired_rule(p) ;
 else
      {
       ++p->tot_ifs_fired ;
       fire_thens(p->rule_then) ;
       update_var_in_res(p) ;
      }
 rule_pop_stack(R_STACK, &x) ;
 return ;
}
void fired_rule(RULE *p)
{
 p->tot_ifs_fired += p->no_ifs + 1 ; /* To distinguish it from natural
 complete firing */
 return ;
}
int true_if(IF *p)
{
 int true_rule_if = YES ;
 switch(p->which)
        {
         case QUL : true_rule_if = true_qualifier(p->qm.ql) ;
                    break ;
         case MATH : true_rule_if = true_expression(p->qm.expr) ;
                     break ;
        }
 return true_rule_if ;
}
int true_expression(EXPRESSION *p)
```

```
{
/* char mesg[MESG_LENGTH] ;*/
 int i, result = TRUE ;
 double left_side = 0.0 , right_side = 0.0 ;
 char *temp ;
/* sprintf(mesg,"Testing Expression \"%s %s %s\"",p->left_part,p->rela
tion,p->right_part) ;
 status_window(mesg) ;*/
 parsing_solving = SOLVING ;
 temp = dup_str(p->left_part) ;
 expression = temp ;
 i = setjmp(jbuf) ;
 if(i == 0)
   {
    next_token() ;
    E(&left_side) ;
   }
 free(temp) ;
 temp = NULL ;
 if(i)
   {
    ERROR(i) ;
    return NO ;
   }
 temp = dup_str(p->right_part) ;
 expression = temp ;
 i = setjmp(jbuf) ;
 if(i == 0)
   {
    next_token() ;
    E(&right_side) ;
   }
 free(temp) ;
 temp = NULL ;
 if(i)
   {
    ERROR(i) ;
    return NO ;
   }
 switch(*p->relation)
       {
        case '<' : result = left_side < right_side ;
                   break ;
        case '>' : result = left_side > right_side ;
                   break ;
        case '!' : result = left_side != right_side ;
                   break ;
        case '=' : result = left_side == right_side ;
                   break ;
       }
/* sprintf(mesg,"Expression \"%s %s %s\" is %s.   ..Press a key",p->le
ft_part,p->relation,p->right_part,result ? "TRUE" : "FALSE") ;
 status_window(mesg) ;
 fflush(stdin) ;
 getch() ;
 status_window("") ;*/
 return result ;
}
int true_qualifier(QUALIFIER *p)
```

```
{
 int is_true = YES ;
 THEN *to_fire = can_fire(p) ;
 struct name_tree *x = NULL ;
 if(to_fire == NULL) /* i.e. Forward chaining */
   set_choices(p) ;
 else /* i.e. backward chaining */
   {
    RULE *q ;
    THEN *curr = to_fire ;
    for(; curr ; curr = curr->same_next)
       {
        q = get_act_rule_no(curr->act_rule_no) ;
        if(is_fired(q))
          continue ;
        single_rule_fire(q) ;
       }
    x = get_frm_ptr(p->qual_name) ;
    if(x->head == NULL) /* i.e. all the rules failed to set the choice
 */
      set_choices(p) ;
   }

 is_true = test_qualifier(p) ;
 return is_true ;
}
THEN *can_fire(QUALIFIER *p)
{
 struct name_tree *x = get_frm_ptr(p->qual_name) ;
 if(x->head != NULL)  /* i.e. already selected choices by some means *
/
   return NULL ;
 else
     {
      THEN *head = get_then_name(p->qual_name) ;
      if(head == NULL)
        return NULL ;
      else /* Test to break a deadlock or cycle */
          {
           THEN *curr = head ;
           for(; curr ; curr = curr->same_next)
              {
               if(!is_fired(get_act_rule_no(curr->act_rule_no)))
                 return curr ; /* Setter make it "return head ;" */
              }
           return NULL ;
          }
     }
}
THEN *get_then_name(char *qual_name)
{
 int test ;
 THEN *curr = q_then_root ;
 for(; curr ;)
    {
     if((test = stricmp(qual_name,curr->ceqm.ql->qual_name)) > 0)
       curr = curr->right ;
     else if(test < 0)
            curr = curr->left ;
```

```
            else
                break ;
        }
    return curr ;
}
void set_choices(QUALIFIER *p)
{
 struct name_tree *x = get_frm_otr(p->qual_name) ;
 if(x->head == NULL)
   {
    int choice ;
    do
      {
       clrscr() ;
       status_window("Enter (C)hoice, (W)hy") ;
       disp_no_qualifier(x->qualifier_no) ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'C' || choice == 'W')) ;
       if(choice == 'C')
         break ;
       else
           {
            prepare_for_why(R_STACK) ;
            why_answer(R_STACK) ;
           }
      } while(choice == 'W') ;
    get_choices(x) ;
    clrscr() ;
   }
 return ;
}
void get_choices(struct name_tree *p)
{
 char *token, *txt ;
 struct text_info ti ;
 int y1, x, type_choice ;
 SELECTION *head = NULL ;
 FRAME *f = get_num_qualifier(p->qualifier_no) ;
 gettextinfo( ti) ;
 if(ti.winbottom == wherey())
   insline() ;
 y1 = wherey() ;
 status_window("Enter Choices....") ;
 do /* while(head == NULL) ; */
   {
    do
      {
       txt = input_choices(2,y1,ti.winright-ti.winleft,ti.winbottom-ti
.wintop-y1) ;
      } while(txt == NULL) ;
    token = strtok(txt," ,+") ;
    if(*token == 'N')
      type_choice = NO ;
    else
        {
         type_choice = YES ;
```

```
          x = atoi(token) ;
          if(x > 0 && x <= f->no_childs)
            head = add_SELECTION(x,type_choice,head) ;
        }
    while((token = strtok('\0'," ,")) != NULL)
        {
         x = atoi(token) ;
         if(x > 0 && x <= f->no_childs)
           head = add_SELECTION(x,type_choice,head) ;
        }
    free(txt) ;
    txt = NULL ;
   } while(head == NULL) ;
 p->head = head ;
 status_window("") ;
 return ;
}
SELECTION *add_SELECTION(int choice, int type, SELECTION *head)
{
 SELECTION *curr, *prev, *x ;
 x = get_SELECTION_node() ;
 x->choice = type == NO ? -choice : choice ;
 if(head == NULL)
   head = x ;
 else
   {
     if(choice == head->choice)
       {
        free(x) ;
        return head ;
       }
     if(abs(head->choice) > abs(choice))
       {
        x->next = head ;
        head = x ;
       }
     else
         {
          for(prev = curr = head ; abs(choice) > abs(curr->choice) &&
 curr ; prev = curr, curr = curr->next)
              ;
          if(abs(choice) != abs(curr->choice))
            {
             x->next = prev->next ;
             prev->next = x ;
            }
          else
              free(x) ;
         }
    }
 return head ;
}
int test_qualifier(QUALIFIER *p)
{
 int present_negative = NO ;
 SELECTION *currq = NULL, *currs = NULL ;
 struct name_tree *x = get_frm_ptr(p->qual_name) ;
 int negative_conditions = p->head->choice < 0 ? YES : NO ;
 FRAME *f = get_num_qualifier(x->qualifier_no) ;
```

```
 int no_choices_q = 0, no_choices_s = 0 ;
 if(!negative_conditions)
   {
    for(currq = p->head ; currq && !present_negative ; currq = currq->
next)
       {
        for(currs = x->head ; currs ; currs = currs->next)
           {
            if(currs->choice < 0)
              {
               if(abs(currq->choice) != abs(currs->choice))
                 {
                  if(!present_negative)
                     present_negative = YES ;
                 }
               else
                   {
                    if(present_negative)
                       present_negative = NO ;
                    break ;
                   }
               continue ;
              }
            if(currq->choice == currs->choice)
              return TRUE ;
           } /* for(currs .....) */
       } /* for(currq .....) */
    if(present_negative)
        return TRUE ;
    else
         return FALSE ;
   }
 else
     {
      for(currq = p->head ; currq ; currq = currq->next)
         {
          ++no_choices_q ;
          for(currs = x->head ; currs ; currs = currs->next)
             {
              if(currq->choice == currs->choice)
                break ;
              if(abs(currq->choice) == abs(currs->choice) && currs->ch
oice > 0)
                return FALSE ;
             } /* for(currs .....) */
         } /* for(currq .....) */
       for(no_choices_s = 0 , currs = x->head ; currs ; currs = currs->
next)
          {
           ++no_choices_s ;
           if(currs->choice > 0)
             return TRUE ;
          }
       if(no_choices_s == (f->no_childs-no_choices_q))
         return FALSE ;
       else
           return TRUE ;
     }
}
```

```
void release_result_memory(void)
{
 choice_result_dispose(head_result) ;
 head_result = tail_result = NULL ;
 sel_dispose(frame_tree) ;
 /* Rest Later */
 return ;
}
void choice_result_dispose(CHOICE *p)
{
 if(p != NULL)
   {
    choice_result_dispose(p->next) ;
    free(p) ;
    p = NULL ;
   }
 return ;
}
void sel_dispose(struct name_tree *p)
{
 if(p != NULL)
   {
    sel_dispose(p->left_name) ;
    sel_dispose(p->right_name) ;
    if(p->head != NULL)
      {
       selections_dispose(p->head) ;
       p->head = NULL ;
      }
   }
 return ;
}
fire_thens(THEN *head)
{
 THEN *curr = head ;
 RULE *p = get_act_rule_no(head->act_rule_no) ;
 for(; curr ; curr = curr->next)
    {
     ++p->tot_thens_fired ;
     curr->fired = YES ;
     fire_ther_single(curr) ;
    }
 return ;
}
void fire_then_single(THEN *p)
{
 switch(p->which)
       {
        case QUL : fire_QUL_then(p->ceqm.ql) ;
                   break ;
        case MATH : fire_MATH_then(p->ceqm.expr) ;
                    break ;
        case EPR : fire_EPR_then(p->ceqm.extprog) ;
                   break ;
        case CHE : fire_CHE_then(p->ceqm.ch) ;
                   break ;
       }
 return ;
}
```

```
void fire_MATH_then(eXPRESSION *p)
{
 int i ;
/* char mesg[MESG_LENGTH] ;*/
 char *temp ;
 VARIABLE *v = NULL ;
 double right_side = 0.0 ;
/* sprintf(mesg,"Firing Expression \"%s %s %s\"    .Press any key",p->
left_part,p->relation,p->right_part) ;
 status_window(mesg) ;*/
 parsing_solving = SOLVING ;
 temp = dup_str(p->right_part) ;
 expression = temp ;
 i = setjmp(jbuf) ;
 if(i == 0)
   {
    next_token() ;
    E(&right_side) ;
   }
 free(temp) ;
 temp = NULL ;
 if(i)
   {
    ERROR(i) ;
    return ;
   }
 v = get_variable(p->left_part) ;
 v->value = right_side ;
 v->val_accepted = YES ;
/* sprintf(mesg,"Assigned \"%g\" to \"%s\".       ...Press a Key",v->v
alue,v->name) ;
 status_window(mesg) ;
 fflush(stdin) ;
 getch() ;
 status_window("") ;*/
 return ;
}
void fire_EXPR_then(EXTPROG *p)
{
 char mesg[MESG_LENGTH] ;
 int status ;
 VARIABLE *v ;
 struct name_list *curr ;
 FILE *fp ;
 char drive[MAXDRIVE], dir[MAXDIR], fname[MAXFILE], ext[MAXEXT], pathn
ame[MAXPATH] ;
/* sprintf(mesg,"Calling External Program \"%s\"    .Press any key",p-
>name) ;
 status_window(mesg) ;
 fflush(stdin) ;
 getch() ;
 status_window("") ;*/
 fnsplit(p->name, drive, dir, fname, ext) ;
 if(p->pass_no)
   {
    fnmerge(pathname, drive, dir, fname, ".PAS") ;
    do
      {
       fp = fopen(pathname,"w") ;
```

```
       if(fp == NULL)
         {
          int choice ;
          sprintf(mesg,"Error while creating file \"%s\".  Retry ? (Y/
N)",pathname) ;
          status_window(mesg) ;
          do
            {
             fflush(stdin) ;
             choice = toupper(getch()) ;
            } while(!(choice == 'Y' || choice == 'N')) ;
          if(choice == 'N')
            return ;
         }
      } while(fp == NULL) ;
    for(curr = p->p_head ; curr ; curr = curr->next)
       {
        v = get_variable(curr->name) ;
        if(!(v->initialized || v->val_accepted))
          prepare_var(v) ;
        fprintf(fp,"%lg\n",v->value) ;
       }
    fclose(fp) ;
   }
 errno = 0 ;
 textcolor(WHITE) ;
 textbackground(BLACK) ;
 status = spawnlp(P_WAIT,p->name,NULL) ;
 window(1,1,80,25) ;
 textbackground(YELLOW) ;
 clrscr() ;
 textbackground(BLACK) ;
 draw(3,2,79,24) ;
 if(status == -1)
   {
    char *txt ;
    txt = strerror(errno) ;
    txt[strlen(txt)-1] = '\0' ;
    sprintf(mesg,"\"%s\" : %s, ..Press any key",p->name,txt) ;
    status_window(mesg) ;
    fflush(stdin) ;
    getch() ;
    status_window("") ;
   }
 else
     {
      if(p->return_no)
        {
         double values ;
         fnmerge(pathname, drive, dir, fname, ".RET") ;
         do
           {
            fp = fopen(pathname,"r") ;
            if(fp == NULL)
              {
               int choice ;
               sprintf(mesg,"Error while opening file \"%s\".  Retry ?
(Y/N)",pathname) ;
               status_window(mesg) ;
```

```
                do
                  {
                   fflush(stdin) ;
                   choice = toupper(getch()) ;
                  } while(!(choice == 'Y' || choice == 'N')) ;
                if(choice == 'N')
                  return ;
                }
              } while(fp == NULL) ;
            for(curr = p->r_head ; curr ; curr = curr->next)
               {
                v = get_variable(curr->name) ;
                fscanf(fp,"%lg\n",&values) ;
                v->value = values ;
                v->val_accepted = YES ;
               }
            fclose(fp) ;
           }
       }
 return ;
}
void fire_JUL_then(QUALIFIER *p)
{
 struct name_tree *x = get_frm_ptr(p->qual_name) ;
 SELECTION *q, *curr, *prev, *y ;
 for(y = p->head ; y ; y = y->next)
    {
     if(x->head == NULL)
       {
        q = get_SELECTION_node() ;
        q->choice = y->choice ;
        x->head = q ;
       }
     else
         {
          if(abs(x->head->choice) == abs(y->choice))
            {
             if(x->head->choice > 0 && y->choice < 0)
               x->head->choice *= -1 ;
             continue ;
            }
          if(abs(x->head->choice) > abs(y->choice))
            {
             q = get_SELECTION_node() ;
             q->choice = y->choice ;
             q->next = x->head ;
             x->head = q ;
             continue ;
            }
          for(prev = curr = x->head ; curr && abs(y->choice) > abs(cur
r->choice) ; prev = curr, curr = curr->next)
             ;
          if(abs(curr->choice) == abs(y->choice))
            {
             if(curr->choice > 0 && y->choice < 0)
               curr->choice *= -1 ;
            }
          else if(abs(curr->choice) > abs(y->choice))
                 {
```

```
                    q = get_SELECTION_node() ;
                    q->choice = y->choice ;
                    q->next = prev->next ;
                    prev->next = q ;
                   }
          }
     } /* for(y......) */
 return ;
}
void fire_the_then(CHOICE *p)
{
 CHOICE *x ;
 if(head_result == NULL)
   {
    x = get_CHOICE_node() ;
    x->method = p->method ;
    x->probability = p->probability ;
    x->result_probability = (float) x->probability / 10.0 ;
    head_result = tail_result = x ;
   }
 else
     {
      CHOICE *curr = head_result ;
      for(; curr ; curr = curr->next)
         {
          if(stricmp(curr->method,p->method) == 0)
            {
             if(!(curr->probability == 10 || curr->probability == 0))
               {
                if(p->probability == 10 || p->probability == 0)
                  {
                   curr->result_probability = (float) p->probability /
10.0 ;
                   curr->probability = p->probability ;
                  }
                else
                    curr->result_probability *= (float) p->probability
/ 10.0 ;
               }
             return ;
            }
         }
       x = get_CHOICE_node() ;
       x->method = p->method ;
       x->probability = p->probability ;
       x->result_probability = (float) x->probability / 10.0 ;
       tail_result->next = x ;
       tail_result = x ;
      }
 return ;
}
void res_print(void)
{
 CHOICE *curr = head_result ;
 int i = 1 ;
 clrscr() ;
 textcolor(BLACK) ;
 textbackground(WHITE) ;
 cprintf("Result(s) :\n\r") ;
```

```
 textcolor(WHITE) ;
 textbackground(BLACK) ;
 for(; curr ; curr = curr->next,++i)
    {
     cprintf("%2d.  %s  : probability %g",i,curr->method,curr->result_
probability) ;
     if(curr->probability == 10)
       cprintf(" (i.e. Absolutely)\n\r") ;
     else if(curr->probability == 0)
            cprintf(" (i.e. Never)\n\r") ;
          else
              cprintf("\n\r") ;
    }
  textcolor(CYAN) ;
  res_vars(v_tree,&i) ;
  textcolor(WHITE) ;
  status_window("Press any key to continue.............") ;
  fflush(stdin) ;
  if(getch() == EXTENDED)
    getch() ;
  return ;
 }
void init_ifs(IF *head)
{
 IF *curr = head ;
 for(; curr ; curr = curr->next)
    if_init_one(curr) ;
 return ;
}
void if_init_one(IF *p)
{
 p->fired = NO ;
 return ;
}
void init_thens(THEN *head)
{
 THEN *curr = head ;
 for(; curr ; curr = curr->next)
    then_init_one(curr) ;
 return ;
}
void then_init_one(THEN *p)
{
 p->fired = NO ;
 return ;
}
int succ_fired(RULE *p)
{
 return (p->tot_ifs_fired > 0 && p->tot_ifs_fired == p->no_ifs + 1) ?
YES : NO ;
}
void successful_rules_view(void)
{
 RULE *curr = rule_head ;
 int no_success = 0, key ;
 for(; curr ; curr = curr->next)
    {
     if(succ_fired(curr))
       {
```

```
        ++no_success ;
        clrscr() ;
        disp_rule(curr) ;
        status_window("Press any key to see next successful rule .....
") ;
        fflush(stdin) ;
        if((key = getch()) == ESC)
          {
           status_window("") ;
           clrscr() ;
           return ;
          }
        else if(key == EXTENDED)
              getch() ;
      }
    }
 if(no_success == 0)
   {
    clrscr() ;
    status_window("No rule was a success.     ....Press any key to cont
inue") ;
   }
 else
     status_window("No more successful rules.     ....Press any key to
continue") ;
 fflush(stdin) ;
 if(getch() == EXTENDED)
   getch() ;
 status_window("") ;
 return ;
}
int whether_failed_rule(RULE *p)
{
 return p->tot_ifs_fired - (p->no_ifs + 1) ;
}
void fail_rule_detail(void)
{
 RULE *curr = rule_head ;
 int no_failed = 0, fail_premise_no, key ;
 char txt[MESG_LENGTH] ;
 for(; curr ; curr = curr->next)
    {
     if((fail_premise_no = whether_failed_rule(curr)) == 0)
       continue ;
     else
         {
          ++no_failed ;
          clrscr() ;
          disp_rule(curr) ;
          sprintf(txt,"Failed at premise #%d.     ...Press any key to s
ee next failed rule",fail_premise_no) ;
          status_window(txt) ;
          fflush(stdin) ;
          if((key = getch()) == ESC)
            {
             status_window("") ;
             clrscr() ;
             return ;
            }
```

```
            else if(key == EXTENDED)
                    getch() ;
                }
        }
 if(no_failed == 0)
   {
    clrscr() ;
    status_window("No rule failed.     ....Press any key to continue")
;
   }
 else
     status_window("No more failed rules.     ....Press any key to cont
inue") ;
 fflush(stdin) ;
 if(getch() == EXTENDED)
   getch() ;
 status_window("") ;
 return ;
}
void init_vars(VARIABLE *p)
{
 if(p)
   {
    p->val_accepted = NO ;
    p->disp_in_res = NO ;
    init_vars(p->left) ;
    init_vars(p->right) ;
   }
 return ;
}
void res_vars(VARIABLE *p,int *i)
{
 if(p)
   {
    if(p->displayed && p->disp_in_res)
      {
       if(p->initialized || p->val_accepted)
         cprintf("%2d.  %s : %g\n\r",(*i)++,p->txt,p->value) ;
       else
           cprintf("%2d.  %s : ***was not deducable ***\n\r",(*i)++,p-
>txt) ;
      }
    res_vars(p->left,i) ;
    res_vars(p->right,i) ;
   }
 return ;
}
THEN *var_get_then_name(char *name)
{
 int test ;
 THEN *curr = m_then_root ;
 for(; curr ;)
    {
     if((test = stricmp(name,curr->ceam.expr->left_part)) > 0)
       curr = curr->right ;
     else if(test < 0)
            curr = curr->left ;
          else
              break ;
```

```
    }
 return curr ;
}
RULE_STACK *R_get_stack(void)
{
 RULE_STACK *p ;
 p = (RULE_STACK *)malloc(sizeof(RULE_STACK)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.     .....ABORTING       ...(R_get_stack)\n")
;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->tos = p->curr_why = NULL ;
 return p ;
}
ELEMENT_RULE_STACK *R_node_get_stack(void)
{
 ELEMENT_RULE_STACK *p ;
 p = (ELEMENT_RULE_STACK *)malloc(sizeof(ELEMENT_RULE_STACK)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.     .....ABORTING       ...(R_get_stack_node)
\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->act_rule_no = 0 ;
 p->next = NULL ;
 return p ;
}
int rule_empty_stack(RULE_STACK *RS)
{
 return RS->tos == NULL ? YES : NO ;
}
init_rule_stack(RULE_STACK *RS)
{
 RS->tos = RS->curr_why = NULL ;
 return ;
}
void prepare_for_why(RULE_STACK *RS)
{
 RS->curr_why = RS->tos ;
 return ;
}
int why_rule_retrieve_stack(RULE_STACK *RS, int *act_rule_no)
{
 if(RS->curr_why != NULL)
   {
    *act_rule_no = RS->curr_why->act_rule_no ;
    RS->curr_why = RS->curr_why->next ;
    return YES ;
   }
 else
```

```
      {
       *act_rule_no = 0 ;
       return NO ;
      }
}
int rule_pop_stack(RULE_STACK *RS, int *act_rule_no)
{
 if(kS->tos != NULL)
   {
    ELEMENT_RULE_STACK *p = RS->tos ;
    kS->tos = RS->tos->next ;
    *act_rule_no = p->act_rule_no ;
    free(p) ;
    p = NULL ;
    return YES ;
   }
 else
     {
      *act_rule_no = 0 ;
      return NO ;
     }
}
void rule_push_stack(RULE_STACK *RS, int act_rule_no)
{
 ELEMENT_RULE_STACK *p ;
 p = R_node_get_stack() ;
 p->act_rule_no = act_rule_no ;
 p->next = RS->tos ;
 RS->tos = p ;
 return ;
}
void why_answer(RULE_STACK *RS)
{
 int act_rule_no, choice = 'W', x, y, more ;
 RULE *p ;
 clrscr() ;
 do
   {
    more = why_rule_retrieve_stack(RS, &act_rule_no) ;
    if(more)
      {
       p = get_act_rule_no(act_rule_no) ;
       disp_rule(p) ;
       cprintf("\n\r\n\r") ;
       textattr(0x70) ;
       if(!p->tot_thens_fired)
         cprintf("Trying to solve premise #%d of the above rule", p->t
ot_ifs_fired) ;
       else
          cprintf("Trying to solve consequence #%d of the above rule"
, p->tot_thens_fired) ;
       textcolor(WHITE) ;
       textbackground(BLACK) ;
       x = wherex() ;
       y = wherey() ;
       status_window("(W)hy, (C)ontinue with firing rules") ;
       do
         {
          fflush(stdin) ;
```

```
          choice = toupper(getch()) ;
         } while(!(choice == 'C' || choice == 'W')) ;
       if(choice == 'C')
         return ;
       gotoxy(x, y) ;
       cprintf("\n\r\n\r") ;
      }
    else

        {
         status_window("No (Last) Rule for your question.  Press a key
 to continue firing Rules") ;
         fflush(stdin) ;
         getch() ;
        }
  } while(choice == 'W' && more) ;
 return ;
}
```

```
/****** NON-LEARNING MODULE *******/
/*******     VAR.C     ******/


#include "shell.h"
extern int parsing_solving ;
extern VARIABLE *v_tree ;
extern char file_name[40] ;
extern curr_rule_no ;
extern THEN *b_head_then ;
extern RULE_STACK *R_STACK ;
int m_err_no = 0 ;
struct exception excep ;
jmp_buf jbuf ;
int token_type ;
char token[MAX_LENGTH] ;
char *expression ;
VARIABLE *var_head, *var_tail ;
char *math_error_mesg[] = {
                           "No Error",
                           "Argument domain error",
                           "Argument singularity error",
                           "Overflow range error",
                           "Underflow range error",
                           "Total loss of significance"
                          } ;
/* Of Other Files */
EXPRESSION *get_MATH_node(void) ;
char *get_char_array(int) ;
char *dup_str(char *) ;
void status_window(char *) ;
char *get_ch_txt(int, int, int, int) ;
char *trim(char *) ;
char *get_val_var_txt(int, int, int, int) ;
RULE *get_act_rule_no(int) ;
int is_fired(RULE *) ;
int single_rule_fire(RULE *) ;
THEN *var_get_then_name(char *) ;
void prepare_for_why(RULE_STACK *) ;
void why_answer(RULE_STACK *) ;
/* In This File */
char * clean_expression(char *) ;
EXPRESSION *if_split_expression(char *) ;
char * get_left_part(char *, int *) ;
int if_parse_expression(EXPRESSION *) ;
int parse_expr(char *) ;
void look_new_variables(EXPRESSION *) ;
void process_var(char *) ;
int exist_variable(char *) ;
VARIABLE *get_variable(char *) ;
VARIABLE *add_variable(VARIABLE *, VARIABLE *) ;
VARIABLE *get_new_variable(char *) ;
VARIABLE *get_VARIABLE_node(void) ;
void E(double *) ;
void S(double *) ;
void T(double *) ;
void F(double *) ;
void P(double *) ;
void C(double *) ;
```

```
void A(double *) ;
void M(double *) ;
void V(double *) ;
void report_error(int) ;
void next_token(void) ;
void operation(double *, double *, char) ;
void unary_operation(double *, char) ;
void func_operation(double *, char) ;
void skip_spaces(void) ;
void ERROR(int) ;
void vars_write(void) ;
void add_to_write_var(VARIABLE *) ;
void var_write_add(VARIABLE *) ;
void vars_read(void) ;
void free_var_list(VARIABLE *) ;
VARIABLE *in_var_list(char *) ;
void update_var_in_res(RULE *) ;
void can_var_fire(char *, THEN **, THEN **) ;
int return_prog(char *, EXTPROG *) ;
void prepare_var(VARIABLE *) ;
char *clean_expression(char *str)
{
 int i, k = 0 ;
 char txt[MAX_LENGTH] ;
 for(i=strlen(str)-1 ; i >= 0 ; --i)
    {
     if(isspace(str[i]) || iscntrl(str[i]))
       str[i] = '\0' ;
     else
       break ;
    }
 for(i=0 ; str[i] && (isspace(str[i]) || iscntrl(str[i])) ; ++i)
    ;
 strcpy(txt,str+i) ;
 for(k=0, i=0 ; txt[i] ; ++i)
    {
     if(!(isspace(txt[i]) || iscntrl(txt[i])))
        str[k++] = txt[i] ;
    }
 str[k] = '\0' ;
 return str ;
}
EXPRESSION *if_split_expression(char *expr)
{
 int posn ;
 EXPRESSION *p = NULL ;
 char *left_part = NULL, *right_part = NULL, relation[3] ;
 left_part = get_left_part(expr,&posn) ;
 if(!left_part)
   return NULL ;
 if(expr[posn] == '!')
   {
    if(expr[++posn] != '=')
      {
       free(left_part) ;
       return NULL ;
      }
    else
       {
```

```
            relation[0] = '!', relation[1] = '=', relation[2] = '\0' ;
            ++posn ;
           }
    }
  else
      {
       relation[0] = expr[posn++], relation[1] = '\0' ;
      }
  if(!expr[posn])
    {
     free(left_part) ;
     return NULL ;
    }
  right_part = get_char_array(strlen(expr) - posn) ;
  strcpy(right_part, expr+posn) ;
  p = get_MATH_node() ;
  p->expression = dup_str(expr) ;
  p->left_part = left_part ;
  p->right_part = right_part ;
  strcpy(p->relation, relation) ;
  return p ;
}
char * get_left_part(char *expr, int *posn)
{
  for(*posn = 0 ; expr[*posn] ; ++*posn)
      {
       if(strchr("<>!=",expr[*posn]))
         break ;
      }
  if(*posn == 0 || *posn >= (strlen(expr) -1))
    return NULL ;
  else
      {
       char *lp = get_char_array(*posn+1) ;
       strncpy(lp, expr, *posn) ;
       lp[*posn] = '\0' ;
       return lp ;
      }
}
int if_parse_expression(EXPRESSION *p)
{
  int correct = YES ;
  correct = parse_expr(p->left_part) ;
  if(!correct)
    return NO ;
  correct = parse_expr(p->right_part) ;
  return correct ;
}
int parse_expr(char *expr)
{
  double result ;
  int i ;
  char *temp = dup_str(expr) ;
  expression = temp ;
  i = setjmp(jbuf) ;
  if(i == 0)
    {
     next_token() ;
     if(*token)
```

```
        {
         parsing_solving = PARSING ;
         E(result) ;
         if(*token)
           i = 12 ; /* New Error # */
        }
    }
 if(i)
    {
     ERROR(i) ;
     return NO ;
    }
 else
     return YES ;
}
void look_new_variables(EXPRESSION *p)
{
 char *temp ;
 char *token, var_name[MAX_LENGTH] ;
 int i, j = 0, var_only = YES ;
 temp = p->left_part ;
 for(i=0 ; *temp ; ++i, temp++)
    {
     if(*temp == '[')
       {
        token = temp++ ;
        j = i++ ;
        while(*temp != ']')
              {
               temp++ ;
               i++ ;
              }
        if(i-j == 1)
          ERROR(13) ; /* Empty [], i.e. Variable name missing */
        strncpy(var_name,token,i-j+1) ;
        var_name[i-j+1] = '\0' ;
        process_var(var_name) ;
       }
     else
         var_only = NO ;
    }
 if(var_only)
   p->var_expr = VAR ;
 else
    p->var_expr = EXPR ;
 temp = p->right_part ;
 for(i=0 ; *temp ; ++i, temp++)
    {
     if(*temp == '[')
       {
        token = temp++ ;
        j = i++ ;
        while(*temp != ']')
             {
              temp++ ;
              i++ ;
             }
        strncpy(var_name, token,i-j+1) ;
        var_name[i-j+1] = '\0' ;
```

```
            process_var(var_name) ;
          }
      }
  return ;
}
void process_var(char *var_name)
{
 int present = NO ;
 VARIABLE *p = NULL ;
 present = exist_variable(var_name) ;
 if(present)
   return ;
 else
     {
      p = get_new_variable(var_name) ;
      if(p)
        add_to_write_var(p) ;
     }
 return ;
}
int exist_variable(char *var_name)
{
 return get_variable(var_name) == NULL ? NO : YES ;
}
VARIABLE *get_variable(char *var_name)
{
 VARIABLE *curr = v_tree ;
 int test ;
 for(; curr ;)
    {
     if((test = stricmp(var_name,curr->name)) > 0)
       curr = curr->right ;
     else if(test < 0)
            curr = curr->left ;
          else
              break ;
    }
 if(curr == NULL)
   curr = in_var_list(var_name) ;
 return curr ;
}
VARIABLE *add_variable(VARIABLE *p, VARIABLE *root)
{
 if(root == NULL)
   root = p ;
 else
     {
      if(stricmp(p->name, root->name) > 0)
        root->right = add_variable(p, root->right) ;
      else
          root->left = add_variable(p, root->left) ;
     }
 return root ;
}
VARIABLE *get_new_variable(char *var_name)
{
 int answer= NO ;
 char *txt ;
 VARIABLE *p = get_VARIABLE_node() ;
```

```
 char mesg[MESG_LENGTH] ;
 struct text_info ti ;
 gettextinfo( ti) ;
 sprintf(mesg, "Enter the text associated with the New variable : %s",
var_name) ;
 status_window(mesg) ;
 window(43,3,77,23) ;
 clrscr() ;
 do /* while(txt == NULL || strlen(txt) == 0) ; */
   {
    txt = get_ch_txt(2,2,26,14) ;
    if(txt != NULL)
      trim(txt) ;
    if(*txt == '\0')
      free(txt) ;
   } while(txt == NULL || strlen(txt) == 0) ;
 p->name = dup_str(var_name) ;
 p->txt = txt;
 clrscr() ;
 status_window("Do You wish to initialize this variable ? (Y/N)") ;
 do
   {
    fflush(stdin) ;
    answer = toupper(getch()) ;
   } while(!(answer == 'Y' || answer == 'N')) ;
 if(answer == 'Y')
   {
    p->initialized = YES ;
    status_window("Enter the initial value of the variable") ;
    do /* while(txt == NULL) ; */
      {
       txt = get_val_var_txt(2,2,26,14) ;
      } while(txt == NULL) ;
    p->value = strtod(txt, NULL) ;
   }
 else
     p->initialized = NO ;
 clrscr() ;
 status_window("Do You wish to display the value of this variable at t
he end  ? (Y/N)") ;
 do
   {
    fflush(stdin) ;
    answer = toupper(getch()) ;
   } while(!(answer == 'Y' || answer == 'N')) ;
 if(answer == 'Y')
    p->displayed = YES ;
 else
     p->displayed = NO ;
 return p ;
}
VARIABLE *get_VARIABLE_node(void)
{
 VARIABLE *p ;
 p = (VARIABLE *) malloc(sizeof(VARIABLE)) ;
 if(p == NULL)
   {
    clrscr() ;
    printf("Out of memory.  ..ABORTING          ...(get_VARIABLE_node)\
```

```
n") ;
     fflush(stdin) ;
     getch() ;
     exit(1) ;
    }
 p->name = p->txt = NULL ;
 p->value = 0.0 ;
 p->initialized = YES ;
 p->val_accepted = NO ;
 p->displayed = NO ;
 p->disp_in_res = NO ;
 p->left = p->right = NULL ;
 return p ;
}
void E(double *result)  /* E -> S { +S | -S } */
{
 double hold ;
 char op ;
 S(result) ;
 while((op = *token) == '+' || op == '-')
       {
        next_token() ;
        S(&hold) ;
        operation(result,&hold,op) ;
       }
 return ;
}
void S(double *result)  /* S -> T { *T | /T | %T } */
{
 double hold ;
 char op ;
 T(result) ;
 while((op = *token) == '*' || op == '/' || op == '%')
       {
        next_token() ;
        T(&hold) ;
        operation(result,&hold,op) ;
       }
 return ;
}
void T(double *result)   /* T -> F | ^T */
{
 double hold ;
 F(result) ;
 if(*token == '^')
   {
    next_token() ;
    T(&hold) ;
    operation(result,&hold,'^') ;
   }
return ;
}
void F(double *result)   /* F -> [ + | - ] P */
{
 char op = 0;
 if((*token == '+' || *token == '-') && token_type == DELIMITER)
   {
    op = *token ;
    next_token() ;
```

```
    }
  P(result) ;
  if(op)
    unary_operation(result,op) ;
  return ;
}
void P(double *result)     /* P -> (E) | C | A  */
{
  if(*token == '(' && token_type == DELIMITER)
    {
     next_token() ;
     E(result) ;
     if(*token != ')')
       report_error(0) ;
     next_token() ;
    }
  else if(isdigit(*token) || *token == '.')
         C(result) ;
       else
          A(result) ;
  return ;
}
void C(double *result) /* C -> A floating or integer constant */
{
  if(token_type == NUMBER)
    {
     *result = atof(token) ;
     next_token() ;
    }
  return ;
}
void A(double *result)     /* A -> [V] | M */
{
  if(*token == '[')
    {
     next_token() ;
     V(result) ; /* Don't need if only parsing is required */
     if(*token != ']')
       report_error(1) ;
     next_token() ;
    }
  else
      M(result) ;
  return ;
}
void M(double *result)    /* M -> SIN(E) | COS(E) | TAN(E) | ASIN(E) |
ACOS(E) | ATAN(E) */
{
  char which = 0 ;
  if(token_type != VAR_FUNC)
    report_error(11) ;
  if(toupper(*token) == 'A')
    {
     if(stricmp(token,"ASIN") == 0)
       which = ASIN ;
     else if(stricmp(token,"ACOS") == 0)
            which = ACOS ;
          else if(stricmp(token,"ATAN") == 0)
                 which = ATAN ;
```

```
                else if(stricmp(token,"ABS") == 0)
                        which = ABS ;
                    else
                        report_error(2) ;
    }
  else
      {
       if(stricmp(token,"SIN") == 0)
         which = SIN ;
       else if(stricmp(token,"COS") == 0)
              which = COS ;
            else if(stricmp(token,"TAN") == 0)
                   which = TAN ;
                 else if(stricmp(token,"FLOOR") == 0)
                        which = FLOOR ;
                      else if(stricmp(token,"CEIL") == 0)
                             which = CEIL ;
                           else if(stricmp(token,"EXP") == 0)
                                  which = EXP ;
                                else if(stricmp(token,"LOG") == 0)
                                       which = LOG ;
                                     else if(stricmp(token,"SQRT") == 0
)
                                            which = SQRT ;
                                          else
                                            report_error(2) ;
       }
  next_token() ;
  if(*token != '(')
    report_error(10) ;
  next_token() ;
  E(result) ;
  func_operation(result,which) ;
  if(*token != ')')
    report_error(9) ;
  next_token() ;
  return ;
}
void ERROR(int e_no)
{
  char mesg[MAX_LENGTH] ;
 static char *e_message[] = {
         /* 0 */                 "Unbalanced Parenthesis, missing \")\"",
         /* 1 */                 "Variable not terminated with \"]\"",
         /* 2 */                 "Undefined function called",
         /* 3 */                 "Attempt to divide by Zero",
         /* 4 */                 "Attemp to find \"%\" with second operand
 Zero",
         /* 5 */                 "Invalid constant",
         /* 6 */                 "Illegal character in the expression",
         /* 7 */                 "Cannot raise -ve no\'s to any power",
         /* 8 */                 "Cannot have operand of ARC sine,cos,tan
> 1 or < -1",
         /* 9 */                 "Unbalanced parenthesis of a function, mi
ssing \")\"",
         /* 10 */                "Missing bracket \"(\" with a function",
         /* 11 */                "Error in expression",
         /* 12 */                "Unwanted character(s) ",
         /* 13 */                "Empty [], Variable name missing",
```

```
        /* 14 */            "Cannot find Square Root of a -ve number"
,
        /* 15 */            "Cannot find LOG of a non-positive number
"
                           } ;
 switch(e_no)
       {
        case 2  :
        case 11 :
        case 12 : sprintf(mesg,"%s : ERROR : %s. ...Press a key\a",tok
en,e_message[e_no]) ;
                  break ;
        case 6 : sprintf(mesg,"%c : ERROR : %s. ...Press a key\a",*exp
ression,e_message[e_no]) ;
                 break ;
        default : if(e_no < MATH_ERROR_STARTING)
                    sprintf(mesg,"ERROR : %s. ...Press a key\a",e_mess
age[e_no]) ;
                  else
                      sprintf(mesg,"%s : %s",excep.name,math_error_me
sg[excep.type]) ;
                  break ;
       }
 status_window(mesg) ;
 fflush(stdin) ;
 getch() ;
 status_window("") ;
 return ;
}
void next_token(void)
{
 char *temp = token ;
 if(isspace(*expression))
   skip_spaces() ;
 if(strchr("*/%+-^[]()",*expression))
   {
    token_type = DELIMITER ;
    *temp++ = *expression++ ;
   }
 else if(isalpha(*expression))
        {
         token_type = VAR_FUNC ;
         do
           {
            *temp++ = *expression++ ;
           }while(isalnum(*expression) || *expression == '_') ;
        }
      else if(isdigit(*expression) || *expression == '.')
             {
              int had_decimal = NO ;
              token_type = NUMBER ;
              do
                {
                 *temp = *expression++ ;
                 if(*temp == '.')
                   {
                    if(!had_decimal)
                      had_decimal = YES ;
                    else
```

```
                    report_error(5) ;
                  }
                ++temp ;
              } while(isdigit(*expression) || *expression == '.') ;
            }
          else
            report_error(6) ;
 *temp = NULL ;
 return ;
}
void operation(double *op1, double *op2, char op)
{
 switch(op)
       {
        case '+' : if(parsing_solving != SOLVING)
                     {
                      *op1 = 13.0 ;
                      break ;
                     }
                   m_err_no = 0 ;
                   *op1 += *op2 ;
                   if(m_err_no)
                     report_error(m_err_no) ;
                   break ;
        case '-' : if(parsing_solving != SOLVING)
                     {
                      *op1 = 13.0 ;
                      break ;
                     }
                   m_err_no = 0 ;
                   *op1 -= *op2 ;
                   if(m_err_no)
                     report_error(m_err_no) ;
                   break ;
        case '*' : if(parsing_solving != SOLVING)
                     {
                      *op1 = 13.0 ;
                      break ;
                     }
                   m_err_no = 0 ;
                   *op1 *= *op2 ;
                   if(m_err_no)
                     report_error(m_err_no) ;
                   break ;
        case '/' : if(parsing_solving == SOLVING)
                     {
                      if(*op2 == 0.0)
                        report_error(3) ;
                      m_err_no = 0 ;
                      *op1 /= *op2 ;
                      if(m_err_no)
                        report_error(m_err_no) ;
                     }
                   else
                       *op1 = 13.0 ;
                   break ;
        case '%' : if(parsing_solving == SOLVING)
                     {
                      if(*op2 == 0.0)
```

```
                        report_error(5) ;
                    }
                  ++temp ;
                } while(isdigit(*expression) || *expression == '.') ;
            }
          else
            report_error(6) ;
 *temp = NULL ;
 return ;
}
void operation(double *op1, double *op2, char op)
{
 switch(op)
       {
        case '+' : if(parsing_solving != SOLVING)
                     {
                      *op1 = 13.0 ;
                      break ;
                     }
                   m_err_no = 0 ;
                   *op1 += *op2 ;
                   if(m_err_no)
                     report_error(m_err_no) ;
                   break ;
        case '-' : if(parsing_solving != SOLVING)
                     {
                      *op1 = 13.0 ;
                      break ;
                     }
                   m_err_no = 0 ;
                   *op1 -= *op2 ;
                   if(m_err_no)
                     report_error(m_err_no) ;
                   break ;
        case '*' : if(parsing_solving != SOLVING)
                     {
                      *op1 = 13.0 ;
                      break ;
                     }
                   m_err_no = 0 ;
                   *op1 *= *op2 ;
                   if(m_err_no)
                     report_error(m_err_no) ;
                   break ;
        case '/' : if(parsing_solving == SOLVING)
                     {
                      if(*op2 == 0.0)
                        report_error(3) ;
                      m_err_no = 0 ;
                      *op1 /= *op2 ;
                      if(m_err_no)
                        report_error(m_err_no) ;
                     }
                   else
                       *op1 = 13.0 ;
                   break ;
        case '%' : if(parsing_solving == SOLVING)
                     {
                      if(*op2 == 0.0)
```

```
                        report_error(4) ;
                      m_err_no = 0 ;
                      *op1 = fmod(*op1,*op2) ;
                      if(m_err_no)
                        report_error(m_err_no) ;
                     }
                   else
                       *op1 = 13.0 ;
                   break ;
        case '^' : if(parsing_solving == SOLVING)
                     {
                      if(*op1 != 0.0)
                        {
                         if(*op1 < 0.0)
                           report_error(7) ;
                         m_err_no = 0 ;
                         *op1 = exp(*op2 * log(*op1)) ;
                         if(m_err_no)
                           report_error(m_err_no) ;
                        }
                     }
                   else
                       *op1 = 13.0 ;
                   break ;
       }
 return ;
}
void unary_operation(double *result, char op)
{
 m_err_no = 0 ;
 if(op == '-')
   *result *= -1 ;
 if(m_err_no)
   report_error(m_err_no) ;
 return ;
}
void func_operation(double *result, char op)
{
 switch(op)
       {
        case SIN : if(parsing_solving != SOLVING)
                     {
                      *result = .7657 ;
                      break ;
                     }
                   m_err_no = 0 ;
                   *result = sin(*result) ;
                   if(m_err_no)
                     report_error(m_err_no) ;
                   break ;
        case COS : if(parsing_solving != SOLVING)
                     {
                      *result = .7657 ;
                      break ;
                     }
                   m_err_no = 0 ;
                   *result = cos(*result) ;
                   if(m_err_no)
                     report_error(m_err_no) ;
```

```
                break ;
case TAN :  if(parsing_solving != SOLVING)
              {
               *result = .7657 ;
               break ;
              }
            m_err_no = 0 ;
            *result = tan(*result) ;
            if(m_err_no)
              report_error(m_err_no) ;
            break ;
case ABS :  if(parsing_solving != SOLVING)
              {
               *result = 13.0 ;
               break ;
              }
            m_err_no = 0 ;
            *result = fabs(*result) ;
            if(m_err_no)
              report_error(m_err_no) ;
            break ;
case ASIN :if(parsing_solving == SOLVING)
              {
               if(fabs(*result) > 1)
                 report_error(8) ;
               m_err_no = 0 ;
               *result = asin(*result) ;
               if(m_err_no)
                 report_error(m_err_no) ;
              }
             else
                 *result = 13.0 ;
            break ;
case ACOS : if(parsing_solving == SOLVING)
             {
              if(fabs(*result) > 1)
                report_error(8) ;
              m_err_no = 0 ;
              *result = acos(*result) ;
              if(m_err_no)
                report_error(m_err_no) ;
             }
            else
                *result = 13.0 ;
            break ;
case ATAN : if(parsing_solving == SOLVING)
             {
              if(fabs(*result) > 1)
                report_error(8) ;
              m_err_no = 0 ;
              *result = atan(*result) ;
              if(m_err_no)
                report_error(m_err_no) ;
             }
            else
                *result = 13.0 ;
            break ;
case FLOOR : if(parsing_solving == SOLVING)
              {
```

```
                        m_err_no = 0 ;
                        *result = floor(*result) ;
                        if(m_err_no)
                          report_error(m_err_no) ;
                       }
                      else
                        *result = 13.0 ;
                      break ;
        case CEIL : if(parsing_solving != SOLVING)
                      {
                       *result = 13.0 ;
                       break ;
                      }
                    m_err_no = 0 ;
                    *result = ceil(*result) ;
                    if(m_err_no)
                      report_error(m_err_no) ;
                     break ;
        case SQRT : if(parsing_solving == SOLVING)
                      {
                       if(*result < 0.0)
                         report_error(14) ;
                       m_err_no = 0 ;
                       *result = sqrt(*result) ;
                       if(m_err_no)
                         report_error(m_err_no) ;
                      }
                    else
                      *result = 13.0 ;
                    break ;
        case EXP : if(parsing_solving == SOLVING)
                     {
                      m_err_no = 0 ;
                      *result = exp(*result) ;
                      if(m_err_no)
                        report_error(m_err_no) ;
                     }
                   else
                     *result = 13.0 ;
                   break ;
        case LOG : if(parsing_solving == SOLVING)
                     {
                      if(*result <= 0.0)
                        report_error(15) ;
                      m_err_no = 0 ;
                      *result = log(*result) ;
                      if(m_err_no)
                        report_error(m_err_no) ;
                     }
                   else
                     *result = 13.0 ;
                   break ;
       }
 return ;
}
void skip_spaces(void)
{
 while(isspace(*expression))
      expression++ ;
```

```
 return ;
}
void V(double *result)
{
 VARIABLE *p ;
 char txt[MAX_LENGTH] ;
 sprintf(txt,"[%s]",token) ;
 if(parsing_solving == PARSING)
   {
    *result = 13.0 ;
    next_token() ;
    return ;
   }
 p = get_variable(txt) ;
 if(p->initialized)
   *result = p->value ;
 else
     {
      if(!p->val_accepted)
         prepare_var(p) ;
      *result = p->value ;
     }
 clrscr() ;
 status_window("") ;
 next_token() ;
 return ;
}
void can_var_fire(char *name, THEN **m_to_fire, THEN **e_to_fire)
{
 VARIABLE *x = get_variable(name) ;
 *m_to_fire = *e_to_fire = NULL ;
 if(!(x->val_accepted || x->initialized))  /* i.e. not initialized or
already assigned value by some means */
   {
    THEN *head = var_get_then_name(name), *curr = NULL ;
    for(curr = head ; curr ; curr = curr->same_next) /* Test to break
a deadlock or cycle */
       if(!is_fired(get_act_rule_no(curr->act_rule_no)))
         {
          *m_to_fire = curr ;
          break ;
         }
    for(curr = o_head_then ; curr ; curr = curr->right)
       if(return_prog(name,curr->ceam.extprog))
         {
          if(!is_fired(get_act_rule_no(curr->act_rule_no)))
            {
             *e_to_fire = curr ;
             break ;
            }
         }
   }
 return ;
}
int return_prog(char *name, EXTPROG *p)
{
 if(p->return_no == 0)
   return NO ;
 else
```

```
      {
       struct name_list *curr = p->r_head ;
       for(; curr ; curr = curr->next)
          {
           if(stricmp(curr->name,name) == 0)
             return YES ;
          }
       return NO ;
      }
}
void report_error(int e_no)
{
 longjmp(jbuf,e_no) ;
}
void var_write_add(VARIABLE *p)
{
 FILE *var_fp = NULL ;
 char txt[MAX_LENGTH] ;
 int choice, success = YES ;
 sprintf(txt,"%s.VAR",file_name) ;
 do /*'while(var_fp == NULL) ; */
   {
    var_fp = fopen(txt,"a+") ;
    if(var_fp == NULL)
      {
       sprintf(txt,"Error while (appending)writing to file \"%s.VAR\".
   ..Retry(R), Exit(X)",file_name) ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'R' || choice == 'X')) ;
       status_window("") ;
       if(choice == 'X')
         {
          fcloseall() ;
          clrscr() ;
          exit(1) ;
         }
      }
   } while(var_fp == NULL) ;
/* fseek(var_fp,0,SEEK_END) ;*/
 choice = fprintf(var_fp,"%s\n",p->name) ;
 choice = fprintf(var_fp,"%s\n",p->txt) ;
 choice = fprintf(var_fp,"%d\n",p->initialized) ;
 if(p->initialized)
   choice = fprintf(var_fp,"%lg\n",p->value) ;
 choice = fprintf(var_fp,"%d\n",p->displayed) ;
 if(!success)
   {
    sprintf(txt,"Error while writing file \"%s.VAR\".  ..ABORTING  ...
.Press any key",file_name) ;
    status_window(txt) ;
    fflush(stdin) ;
    getch() ;
    fcloseall() ;
    exit(1) ;
   }
 flushall() ;
```

```
 fclose(var_fp) ;
 return ;
}
void vars_read(void)
{
 FILE *var_fp ;
 char txt[MAX_LENGTH] ;
 int choice, success = YES ;
 VARIABLE *p ;
 sprintf(txt,"%s.VAR",file_name) ;
 do /* while(var_fp == NULL) ; */
   {
    var_fp = fopen(txt,"r") ;
    if(var_fp == NULL)
      {
       sprintf(txt,"Error while opening file \"%s.VAR\".   ..Retry(R),
Exit(X)",file_name) ;
       status_window(txt) ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'R' || choice == 'X')) ;
       status_window("") ;
       if(choice == 'X')
         {
          fcloseall() ;
          clrscr() ;
          exit(1) ;
         }
      }
   } while(var_fp == NULL) ;
 fscanf(var_fp,"%[^\n]\n",txt) ;
 while(!feof(var_fp) && success)
      {
       p = get_VARIABLE_node() ;
       p->name = dup_str(txt) ;
       fscanf(var_fp,"%[^\n]\n",txt) ;
       p->txt = dup_str(txt) ;
       fscanf(var_fp,"%d\n",&(p->initialized)) ;
       if(p->initialized)
         {
          double x ;
          fscanf(var_fp,"%lg\n",&x) ;
          p->value = x ;
         }
       fscanf(var_fp,"%d\n",&(p->displayed)) ;
       if(!success)
         {
          sprintf(txt,"Error while reading file \"%s.VAR\".  ..ABORTIN
G  ....Press any key",file_name) ;
          status_window(txt) ;
          fflush(stdin) ;
          getch() ;
          clrscr() ;
          exit(1) ;
         }
       v_tree = add_variable(p,v_tree) ;
       fscanf(var_fp,"%[^\n]\n",txt) ;
```

```
      } /* while(!feof(var_fp) && success) */
 fclose(var_fp) ;
 return ;
}
void vars_write(void)
{
 VARIABLE *curr = var_head, *next = var_head->right;
 for(; curr ; curr = next)
    {
     var_write_add(curr) ;
     next = curr->right ;
     curr->right = NULL ;
     v_tree = add_variable(curr,v_tree) ;
    }
 var_head = var_tail = NULL ;
 return ;
}
void add_to_write_var(VARIABLE *p)
{
 if(var_head == NULL)
   var_head = var_tail = p ;
 else
     {
      var_tail->right = p ;
      var_tail = p ;
     }
 return ;
}
void free_var_list(VARIABLE *p)
{
 if(p)
   {
    free_var_list(p->right) ;
    free(p) ;
    p = NULL ;
   }
 return ;
}
VARIABLE *in_var_list(char *var_name)
{
 VARIABLE *curr = var_head ;
 for(; curr ; curr = curr->right)
    if(stricmp(var_name,curr->name) == 0)
      break ;
 return curr ;
}
int matherr(struct exception *e)
{
 excep = *e ;
 m_err_no = MATH_ERROR_STARTING+e->type ;
 return m_err_no ;
}
void update_var_in_res(RULE *r)
{
 VARIABLE *p ;
 int j ;
 char *tkn, var_name[MAX_LENGTH], *temp, *expr ;
 IF *curri ;
 THEN *currt ;
```

```
EXTPROG *e ;
for(curri = r->rule_if ; curri ; curri = curri->next)
    {
     if(curri->which == MATH)
       {
        temp = expr = dup_str(curri->qm.expr->left_part) ;
        while((tkn = strchr(expr,'[')) != NULL)
             {
              j = 0 ;
              do
                {
                 var_name[j++] = *tkn++ ;
                } while(*tkn != ']') ;
              var_name[j++] = ']' ;
              var_name[j] = '\0' ;
              expr = ++tkn ;
              p = get_variable(var_name) ;
              p->disp_in_res = YES ;
             }
        free(temp) ;
        expr = NULL ;
        temp = expr = dup_str(curri->qm.expr->right_part) ;
        while((tkn = strchr(expr,'[')) != NULL)
             {
              j = 0 ;
              do
                {
                 var_name[j++] = *tkn++ ;
                } while(*tkn != ']') ;
              var_name[j++] = ']' ;
              var_name[j] = '\0' ;
              expr = ++tkn ;
              p = get_variable(var_name) ;
              p->disp_in_res = YES ;
             }
        free(temp) ;
        expr = NULL ;
       }
    }
for(currt = r->rule_then ; currt ; currt = currt->next)
    {
     if(currt->which == MATH)
       {
        temp = expr = dup_str(currt->ceqm.expr->left_part) ;
        while((tkn = strchr(expr,'[')) != NULL)
             {
              j = 0 ;
              do
                {
                 var_name[j++] = *tkn++ ;
                } while(*tkn != ']') ;
              var_name[j++] = ']' ;
              var_name[j] = '\0' ;
              expr = ++tkn ;
              p = get_variable(var_name) ;
              p->disp_in_res = YES ;
             }
        free(temp) ;
        expr = NULL ;
```

```
        temp = expr = dup_str(currt->ceqm.expr->right_part) ;
        while((tkn = strchr(expr,'[')) != NULL)
             {
              j = 0 ;
              do
                {
                 var_name[j++] = *tkn++ ;
                } while(*tkn != ']') ;
              var_name[j++] = ']' ;
              var_name[j] = '\0' ;
              expr = ++tkn ;
              p = get_variable(var_name) ;
              p->disp_in_res = YES ;
             }
         free(temp) ;
         expr = NULL ;
        }
       else if(currt->which == EPR)
            {
             struct name_list *x ;
             e = currt->ceqm.extprog ;
             for(x = e->p_head ; x ; x = x->next)
                {
                 p = get_variable(x->name) ;
                 p->disp_in_res = YES ;
                }
             for(x = e->r_head ; x ; x = x->next)
                {
                 p = get_variable(x->name) ;
                 p->disp_in_res = YES ;
                }
            }
    }
 return ;
}
void prepare_var(VARIABLE *p)
{
 char *val ;
 THEN *m_to_fire = NULL, *e_to_fire = NULL ;
 can_var_fire(p->name, &m_to_fire, &e_to_fire) ;
 if(m_to_fire != NULL || e_to_fire != NULL) /* i.e. Backward chaining
 , else Forward Chaining */
   {
    RULE *q ;
    THEN *curr_fire = NULL ;
    char *tmp = dup_str(expression) ;
    int min ;
    for(; m_to_fire || e_to_fire ;)
       {
        min = curr_rule_no ;
        if(m_to_fire)
          {
           q = get_act_rule_no(m_to_fire->act_rule_no) ;
           if(min > q->fire_rule_no)
             {
              min = q->fire_rule_no ;
              curr_fire = m_to_fire ;
             }
           if(e_to_fire == NULL)
```

```
            m_to_fire = m_to_fire->same_next ;
          }
        if(e_to_fire)
          {
           q = get_act_rule_no(e_to_fire->act_rule_no) ;
           if(min > q->fire_rule_no)
             {
              THEN *temp ;
              min = q->fire_rule_no ;
              curr_fire = e_to_fire ;
              for(temp = e_to_fire->right ; temp ; temp = temp->right)
                 {
                  if(return_prog(p->name,temp->ceqm.extprog))
                    {
                     if(!is_fired(get_act_rule_no(temp->act_rule_no)))
                       break ;
                    }
                 }
              e_to_fire = temp ;
             }
           else if(m_to_fire)
                m_to_fire = m_to_fire->same_next ;
          }
        q = get_act_rule_no(curr_fire->act_rule_no) ;
        if(is_fired(q))
          continue ;
        single_rule_fire(q) ;
       }
    expression = tmp ;
   }
 if(!p->val_accepted) /* variable could not be initialized */
   {
    int choice ;
    do
      {
       clrscr() ;
       status_window("Enter (V)alue of variable, (W)hy") ;
       textcolor(WHITE) ;
       textbackground(BLACK) ;
       cprintf("Enter \%s\ \"%s\"",p->txt,p->name) ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'V' || choice == 'W')) ;
       if(choice == 'V')
         break ;
       else
           {
            prepare_for_why(R_STACK) ;
            why_answer(R_STACK) ;
           }
      } while(choice == 'W') ;
    status_window("") ;
    clrscr() ;
    gotoxy(2,2) ;
    textcolor(BLACK) ;
    textbackground(WHITE) ;
    cprintf("Enter \%s\ \"%s\" : ",p->txt,p->name) ;
```

```
/******** NON-LEARNING MODULE ******/
/******** INTERFAC.C **********/


#include <stdio.h>
#include <ctype.h>
#include <conio.h>
/************** Needed For Communication Between Two Programs ********
******/
#include <dir.h>
char drive[MAXDRIVE], dir[MAXDIR], filename[MAXFILE], ext[MAXEXT] ;
int return_vars(double) ;  /* Returns 0 on no error and any positive v
alue otherwise */
/*****************************************************************
******/
main(int argc, char *argv[])
{
  int no ;
  float l,b,h,q ;
  double Grand_area = 0 ;
  char w_nm[10]="Teju",more ;
  int error = 0 ;
/******* For parameters receiving and sending ******/
  fnsplit(argv[0], drive, dir, filename, ext) ;
/************************************************/
  clrscr() ;
  for(;;)
     {
        cprintf("\n\r%s",w_nm) ;
        cprintf("\n\rNumber of walls : ") ;
        scanf("%d",&no) ;
        cprintf("\n\rWall length : ") ;
        scanf("%f",&l) ;
        cprintf("\n\rWall Breadth : ") ;
        scanf("%f",&b) ;
        cprintf("\n\rWall height : ") ;
        scanf("%f",&h) ;
        q = no*l*b*h ;
        Grand_area += q ;
        cprintf("Total Area : %g Units , and Grand Area : %lg Units" ,
q,Grand_area) ;
        cprintf("\n\r\n\r                    More  Wall Types : ") ;
        fflush(stdin) ;
        more = getche() ;
        if(toupper(more) == 'Y')
          continue ;
        else
          break ;
     }
  error = return_vars(Grand_area) ;
  if(error)
    {        /* Any Action to performed in case of failure to return v
alue */
     cprintf("\n\rFailed To Return Values Due To some PROBLEM  ....Pre
ss any Key") ;
     fflush(stdin) ;
     getch() ;
    }
```

```
  return 0 ;
}
/*********************************************************/
/*                                                       */
/*  This   function  Writes  Values  Intended for some */
/*  Program in file with the name of this program with */
/*  extension  ".RET".                                   */
/*                                                       */
/*********************************************************/
int return_vars(double Grand_area)
{
 FILE *fp ;
 char pathname[MAXPATH] ;
 fnmerge(pathname, drive, dir, filename,".RET") ;
 do
   {
    fp = fopen(pathname,"w") ;
    if(fp == NULL)
      {
       int choice ;
       cprintf("\n\rError while creating file \"%s\".  Retry ? (Y/N)",
pathname) ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getche()) ;
          cprintf("\n\r") ;
         } while(!(choice == 'Y' || choice == 'N')) ;
       if(choice == 'N')
         return 1 ;
      }
   } while(fp == NULL) ;
 fprintf(fp,"%lg\n",Grand_area) ;
 fclose(fp) ;
 return 0 ;
}
```

```
/******* LEARNING-MODULE ********/
/******* RULES.H *********/

typedef struct if_and_then {
                              int left_part ; /* VARIABLE or OBJECT */
                              char *left_object ;
                              int right_part ;
                              char *right_object ;
                              char *attribute ;
                              struct if_and_then *next, *prev ;
                             } IF_THEN ;
typedef struct rule {
                      int no_ifs ;
                      int tot_ifs_fired ;
                      IF_THEN *rule_if ;
                      int no_thens ;
                      int tot_thens_fired ;
                      IF_THEN *rule_then ;
                      struct rule *next, *prev ;
                     } RULE ;
typedef struct vars_in_rules {
                               char *name ;
                               char *object ;
                               struct vars_in_rules *next ;
                              } RULE_VARS ;
```

```
/***** LEARNING-MODULE *****/
/********* RULES.C *********/

#include "shell.h"
extern char file_name[] ;
extern struct qual_name *qualifier_tree ;
RULE *rule_head = NULL, *rule_tail = NULL ;
int curr_rule_no = 1 ;
int new_rules_added = FALSE ;
int curr_fired_rule = 0 ;
RULE_VARS *var_list = NULL ;
/* In Other Files */
void clear_windows(void) ;
void status_window(char *) ;
char *get_ch_txt(int, int, int, int) ;
void trim(char *) ;
char *dup_str(char *) ;
void draw(int, int, int, int) ;
ver_line(int ,int, int) ;
void disp_one_if_then(IF_THEN *) ;
char *get_prob_txt(int, int, int, int) ;
char *get_char_array(int) ;
int can_left_side(char *, char *) ;
int can_right_side(char *, char *) ;
struct name_list *left_list_rel(char *) ;
struct name_list *right_list_rel(char *) ;
FRAME *get_nm_qualifier(char *) ;
struct name_tree *get_frm_ptr(char *) ;
FRAME *getnode(void) ;
void process_frame(FRAME *) ;
void w_frm_pr_ind(void) ;
void write_relations(void) ;
void free_FRAME(FRAME *) ;
struct qual_name *get_object(char *, struct qual_name *) ;
int get_verb_ind(char *) ;
STACK *get_stack(void) ;
int empty_stack(STACK *) ;
void clear_stack(STACK *) ;
void free_STACK(STACK *) ;
void push(char *, STACK *) ;
/* In This File */
int get_rules(void) ;
IF_THEN *get_if_thens(int *) ;
RULE *get_rule_node(void) ;
IF_THEN *get_IF_THEN_node(void) ;
void add_if_then_list(IF_THEN *, IF_THEN **, IF_THEN **) ;
void add_to_rule_list(RULE *) ;
IF_THEN *get_one_if_then(void) ;
void new_get_rules(void) ;
void view_rules(void) ;
void rules_write(void) ;
void read_rules(void) ;
void fire_rules(void) ;
void res_print(void) ;
void release_result_memory(void) ;
void successful_rules_view(void) ;
void fail_rule_detail(void) ;
void disp_rule(RULE *, int) ;
RULE *no_disp_rule(int) ;
```

```
void if_or_then_disp(IF_THEN *) ;
void Copy(char *, char *) ;
int one_rule_write(RULE *, FILE *) ;
int if_write_then(IF_THEN *, FILE *) ;
int one_if_then_write(IF_THEN *, FILE *) ;
void read_rules(void) ;
RULE *read_one_rule(FILE *) ;
IF_THEN *read_if_thens(int, FILE *) ;
IF_THEN *if_one_read_then(FILE *) ;
int single_rule_fire(RULE *) ;
int solve_premise(IF_THEN *) ;
void result_display(IF_THEN *) ;
RULE_VARS *get_variable(char *) ;
RULE_VARS *get_var_node(void) ;
void add_var(RULE_VARS *) ;
char *guess_var(char *, int) ;
char *get_disp_options(struct name_list *) ;
int used_object_in_var(char *) ;
void free_var_list(void) ;
void free_each_var(RULE_VARS *) ;
int succ_fired(RULE *) ;
int whether_failed_rule(RULE *) ;
void new_rule_save_option(void) ;
RULE *dup_RULE(RULE *) ;
IF_THEN *dup_IF_THENS(IF_THEN *) ;
void init_for_firing(void) ;
void RULE_dispose(RULE *) ;
void IF_dispose_THEN(IF_THEN *) ;
void update_semantic_net(IF_THEN *) ;
void save_success_var_desc(void) ;
RULE_VARS *read_success_var_desc(void) ;
void find_differences(RULE_VARS *, STACK *) ;
void var_difference_find(char *, char *, STACK *) ;
int disp_explainations(STACK *) ;
void new_reasons(void) ;
void get_complementary_premises(int, STACK *, IF_THEN **, IF_THEN **)
;
char *get_var_for_object(char *) ;
void reverse_consequence(IF_THEN *) ;
void all_reverse_consequence(IF_THEN *) ;
int get_rules(void)
{
 int no_ifs = 0, no_thens = 0 ;
 IF_THEN *if_rule = NULL, *then_rule = NULL ;
 struct text_info ti ;
 gettextinfo(&ti) ;
 window(4,3,33,23) ;
 textbackground(BLACK) ;
 clear_windows() ;
 textcolor(BLACK) ;
 textbackground(WHITE) ;
 cprintf("RULE #%d.\n\r\n\r",curr_rule_no) ;
 cprintf("IF\n\r") ;
 if_rule = get_if_thens(&no_ifs) ;
 if(no_ifs)
   {
    cprintf("\n\rTHEN\n\r") ;
    then_rule = get_if_thens(&no_thens) ;
    if(no_thens)
```

```
        {
         RULE *p = get_rule_node() ;
         ++curr_rule_no ;
         p->no_ifs = no_ifs ;
         p->rule_if = if_rule ;
         p->no_thens = no_thens ;
         p->rule_then = then_rule ;
         add_to_rule_list(p) ;
         ++new_rules_added ;
         return YES ;
        }
      else
          {
           IF_dispose_THEN(if_rule) ;
           if_rule = NULL ;
          }
    }
 textattr(ti.attribute) ;
 return NO ;
}
IF_THEN *get_if_thens(int *no_if_thens)
{
 IF_THEN *p = NULL, *head = NULL, *tail = NULL ;
 struct text_info ti ;
 int x,y ;
 x = wherex() ;
 y = wherey() ;
 gettextinfo(&ti) ;
 window(41,3,77,23) ;
 textcolor(WHITE) ;
 textbackground(BLACK) ;
 clrscr() ;
 do
   {
    p = get_one_if_then() ;
    if(p)
      {
       +++no_if_thens ;
       add_if_then_list(p, &head, &tail) ;
       window(4,3,38,23) ;
       gotoxy(x,y) ;
       disp_one_if_then(p) ;
       x = wherex() ;
       y = wherey() ;
       window(43,3,77,23) ;
      }
   } while(p != NULL) ;
 window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
 textattr(ti.attribute) ;
 gotoxy(x,y) ;
 return head ;
}
RULE *get_rule_node(void)
{
 RULE *p = NULL ;
 p = (RULE *) malloc(sizeof(RULE)) ;
 if(p == NULL)
   {
    printf("Out of memory.  ...ABORTING        ...(get_rule_node)\n") ;
```

```
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->no_ifs = p->no_thens = 0 ;
 p->tot_ifs_fired = p->tot_thens_fired = 0 ;
 p->rule_if = p->rule_then = NULL ;
 p->next = p->prev = NULL ;
 return p ;
}
IF_THEN *get_IF_THEN_node(void)
{
 IF_THEN *p = NULL ;
 p = (IF_THEN *) malloc(sizeof(IF_THEN)) ;
 if(p == NULL)
   {
    printf("Out of memory.  ...ABORTING        ...(get_IF_THEN_node)\n"
) ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->left_part = p->right_part = VARIABLE ;
 p->left_object = p->right_object = p->attribute = NULL ;
 p->next = p->prev = NULL ;
 return p ;
}
void add_to_rule_list(RULE *p)
{
 if(rule_head == NULL)
   rule_head = rule_tail = p ;
 else
     {
      rule_tail->next = p ;
      p->prev = rule_tail ;
      rule_tail = p ;
     }
 return ;
}
void add_if_then_list(IF_THEN *p, IF_THEN **head, IF_THEN **tail)
{
 if(*head == NULL)
   *head = *tail = p ;
 else
     {
      (*tail)->next = p ;
      p->prev = *tail ;
      *tail = p ;
     }
 return ;
}
IF_THEN *get_one_if_then(void)
{
 int choice ;
 IF_THEN *p = NULL ;
 char *txt = NULL ;
 int x, y ;
 status_window("Left part is (V)ariable or (O)bject,  <ENTER> or <ESC>
 to finish") ;
```

```
 do
   {
    fflush(stdin) ;
    choice = toupper(getch()) ;
   } while(!(choice == 'V' || choice == 'O' || choice == ENTER || choi
ce == ESC)) ;
 if(choice == ENTER || choice == ESC)
   return NULL ;
 else
    {
     switch(choice)
           {
            case 'V' : status_window("Enter a Variable name within []
,  e.g. [X]") ;
                       txt = get_ch_txt(2,1,26,14) ;
                       if(txt == NULL)
                         return NULL ;
                       trim(txt) ;
                       if(strlen(txt) == 0)
                         {
                          free(txt) ;
                          return NULL ;
                         }
                       p = get_IF_THEN_node() ;
                       p->left_part = VARIABLE ;
                       p->left_object = dup_str(txt) ;
                       free(txt) ;
                       txt = NULL ;
                       break ;
            case 'O' : status_window("Enter an Object's name") ;
                       txt = get_ch_txt(2,1,26,14) ;
                       if(txt == NULL)
                         return NULL ;
                       trim(txt) ;
                       if(strlen(txt) == 0)
                         {
                          free(txt) ;
                          return NULL ;
                         }
                       p = get_IF_THEN_node() ;
                       p->left_part = OBJECT ;
                       p->left_object = dup_str(txt) ;
                       free(txt) ;
                       txt = NULL ;
                       break ;
           }
    }
 gotoxy(2,16) ;
 cprintf("%s ",p->left_object) ;
 x = wherex() ;
 y = wherey() ;
 status_window("Enter the attribute, For NOT conditions -> '!attribute
_name'") ;
 do
   {
    txt = get_ch_txt(2,1,26,14) ;
    if(txt == NULL)
      continue ;
    else {
```

```
            trim(txt) ;
            if(strlen(txt) == 0)
              {
               free(txt) ;
               txt = NULL ;
               continue ;
              }
            else
                break ;
           }
    } while(TRUE) ;
  p->attribute = get_char_array(strlen(txt)+1) ;
  if(*txt == '!')
   {
    p->attribute[0] = '-' ;
    strcpy(p->attribute+1,txt+1) ;
   }
  else
      {
       p->attribute[0] = '+' ;
       strcpy(p->attribute+1,txt) ;
      }
  free(txt) ;
  txt = NULL ;
  gotoxy(x,y) ;
  cprintf("%s ",p->attribute+1) ;
  if(p->attribute[0] == '-')
    cprintf("NOT ") ;
  x = wherex() ;
  y = wherey() ;
  status_window("Right part is (V)ariable or (O)bject") ;
  do
    {
     fflush(stdin) ;
     choice = toupper(getch()) ;
    } while(!(choice == 'V' || choice == 'O')) ;
  switch(choice)
        {
         case 'V' : status_window("Enter a Variable name within [],  e.
g. [X]") ;
                    do
                      {
                       txt = get_ch_txt(2,1,26,14) ;
                       if(txt == NULL)
                         continue ;
                       else {
                              trim(txt) ;
                              if(strlen(txt) == 0)
                                {
                                 free(txt) ;
                                 txt = NULL ;
                                 continue ;
                                }
                              else
                                  break ;
                             }
                      } while(TRUE) ;
                    p->right_part = VARIABLE ;
                    p->right_object = dup_str(txt) ;
```

```
                    free(txt) ;
                    txt = NULL ;
                    break ;
          case 'O' : status_window("Enter an Object's name") ;
                    do
                      {
                       txt = get_ch_txt(2,1,26,14) ;
                       if(txt == NULL)
                         continue ;
                       else {
                             trim(txt) ;
                             if(strlen(txt) == 0)
                               {
                                free(txt) ;
                                txt = NULL ;
                                continue ;
                               }
                             else
                                 break ;
                           }
                      } while(TRUE) ;
                    p->right_part = OBJECT ;
                    p->right_object = dup_str(txt) ;
                    free(txt) ;
                    txt = NULL ;
                    break ;
        }
 clrscr() ;
 return p ;
}
void disp_one_if_then(IF_THEN *p)
{
 cprintf("   %s %s%s %s\n\r",p->left_object, p->attribute+1,p->attribu
te[0] == '-' ? " NOT" : "", p->right_object) ;
 return ;
}
void view_rules(void)
{
 RULE *p = rule_head ;
 struct text_info ti ;
 int rule_no = 1 ;
 char msg1[MAX_LENGTH], *txt ;
 int key, x ;
 if(curr_rule_no == 1)
   return ;
 gettextinfo(&ti) ;
 textcolor(WHITE) ;
 textbackground(BLACK) ;
 clear_windows() ;
 window(4,3,77,23) ;
 clrscr() ;
 disp_rule(p,rule_no) ;
 sprintf(msg1,"Previous(%c or %c), Next(%c or %c), Rule #(N), Exit(X)"
,27,24,26,25) ;
 do
   {
    status_window(msg1) ;
    fflush(stdin) ;
    key = toupper(getch()) ;
```

```
    switch(key)
          {
           case 'X'    :
           case ESC    :
           case ENTER : status_window("") ;
                        window(ti.winleft, ti.wintop, ti.winright, ti.
winbottom) ;
                        textattr(ti.attribute) ;
                        gotoxy(ti.curx, ti.cury) ;
                        return ;
           case 'N' : window(1,1,80,25) ;
                      status_window("") ;
                      gotoxy(1,25) ;
                      cprintf("Enter Rule #") ;
                      fflush(stdin) ;
                      txt = get_prob_txt(wherex(),wherey(),6,1) ;
                      window(4,3,77,23) ;
                      if(txt == NULL)
                        break ;
                      x = atoi(txt) ;
                      if(x < 1 || x >= curr_rule_no)
                        {
                         char msg2[MESG_LENGTH] ;
                         sprintf(msg2,"(%s): No such Rule no.  ...Pres
s any key",txt) ;
                         status_window(msg2) ;
                         fflush(stdin) ;
                         getch() ;
                        }
                      else
                          {
                           rule_no = x ;
                           clrscr() ;
                           p = no_disp_rule(rule_no) ;
                          }
                      break ;
           case EXTENDED : switch(key = getch())
                                {
                                 case UP_ARR   :
                                 case LEFT_ARR : if(rule_no == 1)
                                                   {
                                                    rule_no = curr_ru
le_no - 1 ;
                                                    p = rule_tail ;
                                                    putchar('\a') ;
                                                   }
                                                 else
                                                     {
                                                      --rule_no ;
                                                      p = p->prev ;
                                                     }
                                                 clrscr() ;
                                                 disp_rule(p, rule_no
) ;
                                                 break ;
                                 case DOWN_ARR  :
                                 case RIGHT_ARR : if(rule_no == curr_
rule_no - 1)
                                                    {
```

```
                                                    rule_no = 1 ;
                                                    p = rule_head ;
                                                    putchar('\a') ;
                                                  }
                                                else
                                                    {
                                                     ++rule_no ;
                                                     p = o->next ;
                                                    }
                                                clrscr() ;
                                                disp_rule(p, rule_n
o) ;
                                                break ;
                                    }
                                break ;
          } /* switch(key) */
   } while(TRUE) ;
}
RULE *no_rule_get(int rule_no)
{
 RULE *curr ;
 int search_from, count = 0 ;
 if(rule_no < 1 || rule_no >= curr_rule_no)
   return NULL ;
 search_from = rule_no > (curr_rule_no/2) ? END : BEGIN ;
 if(search_from == BEGIN)
   {
    for(curr = rule_head, count = 1 ; curr && count < rule_no ; curr =
 curr->next, ++count)
       ;
   }
 else
    {
     for(curr = rule_tail, count = curr_rule_no-(rule_no+1) ; curr &&
 count ; curr = curr->prev, --count)
        ;
    }
 return curr ;
}
RULE *no_disp_rule(int rule_no)
{
 RULE *p ;
 p = no_rule_get(rule_no) ;
 if(p != NULL)
   disp_rule(p, rule_no) ;
 return p ;
}
void disp_rule(RULE *p, int rule_no)
{
 textcolor(BLACK) ;
 textbackground(WHITE) ;
 cprintf("Rule #%d\n\r\n\r   IF\n\r", rule_no) ;
 textattr(WHITE | (BLACK <<4)) ;
 if_or_then_disp(p->rule_if) ;
 textcolor(BLACK) ;
 textbackground(WHITE) ;
 cprintf("\n\r   THEN\n\r") ;
 textattr(WHITE | (BLACK <<4)) ;
 if_or_then_disp(p->rule_then) ;
```

```
 return ;
}
void if_or_then_disp(IF_THEN *head)
{
 IF_THEN *curr ;
 for(curr = head ; curr ; curr = curr->next)
   disp_one_if_then(curr) ;
 return ;
}
void rules_write(void)
{
 int choice, success = YES ;
 char txt[MAX_LENGTH] ;
 FILE *rules_fp ;
 RULE *curr ;
 if(new_rules_added == NO)
   return ;
 sprintf(txt,"%s.RUL",file_name) ;
 rules_fp = fopen(txt,"r") ;
 if(rules_fp != NULL)
   {
    char inp[MESG_LENGTH] , out[MESG_LENGTH] ;
    fclose(rules_fp) ;
    sprintf(inp,"%s.RUL",file_name) ;
    sprintf(out,"%s.RBK",file_name) ;
    copy(inp,out) ;
   }
 else
     fclose(rules_fp) ;
 do /* while(!success) ; */
   {
    sprintf(txt,"%s.RUL",file_name) ;
    do /* while(rules_fp == NULL) ; */
      {
       rules_fp = fopen(txt,"w") ;
       if(rules_fp == NULL)
         {
          sprintf(txt,"Error while (over)writing file \"%s.RUL\".   ..
Retry ? (Y/N)",file_name) ;
          do
            {
             fflush(stdin) ;
             choice = toupper(getch()) ;
            } while(!(choice == 'Y' || choice == 'N')) ;
          status_window("") ;
          if(choice == 'N')
            {
             fclose(rules_fp) ;
             return ;
            }
         }
      } while(rules_fp == NULL) ;
    fprintf(rules_fp,"%d\n",curr_rule_no) ;
    for(curr = rule_head ; curr && success ; curr = curr->next)
       success = one_rule_write(curr,rules_fp) ;
    if(!success)
      {
       sprintf(txt,"Error while writing file \"%s.RUL\".  ..Retry ? (Y
/N)",file_name) ;
```

```
        status_window(txt) ;
        do
          {
           fflush(stdin) ;
           choice = toupper(getch()) ;
          } while(!(choice == 'Y' || choice == 'N')) ;
        if(choice == 'N')
          {
           char inp[MESG_LENGTH] , out[MESG_LENGTH] ;
           fclose(rules_fp) ;
           sprintf(out,"%s.RUL",file_name) ;
           sprintf(inp,"%s.RBK",file_name) ;
           copy(inp,out) ;
           return ;
          }
       }
    } while(!success) ;
 new_rules_added = NO ;
 fclose(rules_fp) ;
 return ;
}
int one_rule_write(RULE *p, FILE *fp)
{
 int success = YES ;
 fprintf(fp,"%d\n",p->no_ifs) ;
 success = if_write_then(p->rule_if,fp) ;
 if(success)
   {
    fprintf(fp,"%d\n",p->no_thens) ;
    success = if_write_then(p->rule_then,fp) ;
   }
 return success ;
}
int if_write_then(IF_THEN *head, FILE *fp)
{
 int success = YES ;
 IF_THEN *curr ;
 for(curr = head ; curr && success ; curr = curr->next)
    success = one_if_then_write(curr,fp) ;
 return success ;
}
int one_if_then_write(IF_THEN *p, FILE *fp)
{
 int success = YES ;
 fprintf(fp,"%d\n",p->left_part) ;
 fprintf(fp,"%s\n",p->left_object) ;
 fprintf(fp,"%s\n",p->attribute) ;
 fprintf(fp,"%d\n",p->right_part) ;
 fprintf(fp,"%s\n",p->right_object) ;
 return success ;
}
void read_rules(void)
{
 int success = YES, choice ;
 char txt[MESG_LENGTH] ;
 FILE *rules_fp ;
 RULE *p ;
 int x ;
 do /* while(rules_fp == NULL) ; */
```

```
   {
    sprintf(txt,"%s.RUL",file_name) ;
    rules_fp = fopen(txt,"r") ;
    if(rules_fp == NULL)
      {
       sprintf(txt,"Error while opening file \"%s.RUL\".    ..Retry ? (
Y/N)",file_name) ;
       status_window(txt) ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'Y' || choice == 'N')) ;
       status_window("") ;
       if(choice == 'N')
         {
          fcloseall() ;
          clrscr() ;
          exit(1) ;
         }
      }
   } while(rules_fp == NULL) ;
 fscanf(rules_fp,"%d\n",&x) ;
 if(x != 1 && feof(rules_fp))
   {
    sprintf(txt,"Abnormal end of \"%s.RUL\".    ..ABORTING     ....Press
 any key",file_name) ;
    status_window(txt) ;
    fflush(stdin) ;
    getch() ;
    fcloseall() ;
    exit(1) ;
   }
 curr_rule_no = x ;
 for(x = 1 ; x < curr_rule_no && success ; ++x)
    {
     p = read_one_rule(rules_fp) ;
     add_to_rule_list(p) ;
    }
 if(!success)
   {
    sprintf(txt,"File \"%s.RUL\"  corrupted.    ..ABORTING     ....Press
 any key",file_name) ;
    status_window(txt) ;
    fflush(stdin) ;
    getch() ;
    fcloseall() ;
    exit(1) ;
   }
 fclose(rules_fp) ;
 return ;
}
RULE *read_one_rule(FILE *fp)
{
 RULE *p ;
 IF_THEN *if_then_head ;
 int x, success = YES ;
 p = get_rule_node() ;
 fscanf(fp,"%d\n",&x) ;
```

```
 p->no_ifs = x ;
 if_then_head = read_if_thens(x,fp) ;
 p->rule_if = if_then_head ;
 fscanf(fp,"%d\n",&x) ;
 p->no_thens = x ;
 if_then_head = read_if_thens(x,fp) ;
 p->rule_then = if_then_head ;
 if(!success)
   return NULL ;
 else
     return p ;
}
IF_THEN *read_if_thens(int no_if_thens, FILE *fp)
{
 IF_THEN *head = NULL, *tail = NULL, *curr = NULL ;
 while(no_if_thens--)
       {
        curr = if_one_read_then(fp) ;
        if(curr != NULL)
          {
           if(head == NULL)
             head = tail = curr ;
           else
               {
                tail->next = curr ;
                tail = curr ;
               }
          }
        else
            {
             char txt[MESG_LENGTH] ;
             sprintf(txt,"File \"%s.RUL\" corrupted.   ..ABORTING  ....
Press any key",file_name) ;
             status_window(txt) ;
             fflush(stdin) ;
             getch() ;
             fcloseall() ;
             clrscr() ;
             exit(1) ;
            }
       } /* while(no_if_thens--) */
 return head ;
}
IF_THEN *if_one_read_then(FILE *fp)
{
 int success = YES ;
 IF_THEN *p ;
 int x ;
 char txt[MESG_LENGTH] ;
 p = get_IF_THEN_node() ;
 fscanf(fp,"%d\n",&x) ;
 p->left_part = x ;
 fscanf(fp,"%[^\n]\n",txt) ;
 p->left_object = dup_str(txt) ;
 fscanf(fp,"%[^\n]\n",txt) ;
 p->attribute = dup_str(txt) ;
 fscanf(fp,"%d\n",&x) ;
 p->right_part = x ;
 fscanf(fp,"%[^\n]\n",txt) ;
```

```
 p->right_object = dup_str(txt) ;
  if(!success)
    return NULL ;
  else
      return p ;
}
void Copy(char *source, char *dest)
{
 FILE *inp, *out ;
 char c ;
 inp = fopen(source,"rb") ;
 out = fopen(dest,"wb") ;
 if(inp == NULL || out == NULL)
   {
    printf("Error in source or destination file\n") ;
    exit(1) ;
   }
 c = getc(inp) ;
 while(!feof(inp))
       {
        putc(c,out) ;
        c = getc(inp) ;
       }
 flushall() ;
 fclose(inp) ;
 fclose(out) ;
 return ;
}
void free_var_list(void)
{
 free_each_var(var_list) ;
 var_list = NULL ;
 return ;
}
void free_each_var(RULE_VARS *p)
{
 if(p)
   {
    free_each_var(p->next) ;
    free(p->name) ;
    free(p->object) ;
    free(p) ;
    p = NULL ;
   }
 return ;
}
void fire_rules(void)
{
 RULE *curr ;
 struct text_info ti ;
 gettextinfo(&ti) ;
 ver_line(39,3,24) ;
 window(4,3,38,23) ;
 init_for_firing() ;
 for(curr_fired_rule = 0, curr = rule_head ; curr ; curr = curr->next)
    {
     ++curr_fired_rule ;
     single_rule_fire(curr) ;
    }
```

```
 window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
 textattr(ti.attribute) ;
 gotoxy(ti.curx,ti.cury) ;
 return ;
}
int single_rule_fire(RULE *p)
{
 IF_THEN *curr ;
 int success = YES ;
 clrscr() ;
 disp_rule(p,curr_fired_rule) ;
 for(curr = p->rule_if ; curr && success ; curr = curr->next)
    {
     p->tot_ifs_fired++ ;
     success = solve_premise(curr) ;
    }
 if(!success)
   p->tot_ifs_fired += p->no_ifs+1 ;
 else
    {
     int choice ;
     ++p->tot_ifs_fired ;
     result_display(p->rule_then) ;
     textcolor(WHITE) ;
     textbackground(BLACK) ;
     status_window("Correct ? (Y/N)") ;
     do
      {
       fflush(stdin) ;
       choice = toupper(getch()) ;
      } while(!(choice == 'Y' || choice == 'N')) ;
     if(choice == 'N')
       {
        RULE *r = NULL ;
        RULE_VARS *head_history = NULL ;
        IF_THEN *new_consequences = NULL ;
        /* char prev[100], present[100] ; */
        /* struct text_info ti ; */
        STACK *expl_stack = NULL ;
        int reason_no = -1, no_thens = 0 ;
        r = dup_RULE(p) ;
        /*************************************************************/
        /******** TAKE NEW PREMISES & MODIFY PREVIOUS RULES ********/
        /*************************************************************/
        head_history = read_success_var_desc() ;
        expl_stack = get_stack() ;
        find_differences(head_history,expl_stack) ;
        reason_no = disp_explainations(expl_stack) ;
        if(reason_no == 0 || reason_no == -1)
          {
           IF_THEN *new_premises = NULL ;
           int no_ifs = 0 ;
           status_window("Press any key to enter more premises, Prevoi
us ones displayed above") ;
           clrscr() ;
           disp_rule(p, curr_fired_rule) ;
           textcolor(BLACK) ;
           textbackground(BROWN) ;
           cprintf("\n\r   IF (New to be appended)\n\r") ;
```

```
            textcolor(WHITE) ;
            textbackground(BLACK) ;
            fflush(stdin) ;
            getch() ;
            new_premises = get_if_thens(&no_ifs) ;
            p->no_ifs += no_ifs ;
            if(no_ifs)
              {
               IF_THEN *head_compl_premises = NULL, *curr_compl_premise
s = NULL, *curr , *tail_compl_premises = NULL ;
               for(curr = new_premises ; curr ; curr = curr->next)
                  {
                   curr_compl_premises = dup_IF_THENS(curr) ;
                   add_if_then_list(curr_compl_premises,&head_compl_pre
mises, &tail_compl_premises) ;
                  }
               all_reverse_consequence(head_compl_premises) ;
               for(curr = p->rule_if ; curr->next ; curr = curr->next)
                  ;
               curr->next = new_premises ;
               for(curr = r->rule_if ; curr->next ; curr = curr->next)
                  ;
               curr->next = head_compl_premises ;
               r->no_ifs = p->no_ifs ;
              }
          }
        else
            {
             IF_THEN *premise1 = NULL, *premise2 = NULL, *tail = NULL
;
             get_complementary_premises(reason_no, expl_stack, &premis
e1, &premise2) ;
             for(tail = p->rule_if ; tail->next ; tail = tail->next)
                ;
             ++p->no_ifs ;
             add_if_then_list(premise1,&p->rule_if,&tail) ;
             for(tail = r->rule_if ; tail->next ; tail = tail->next)
                ;
             ++r->no_ifs ;
             add_if_then_list(premise2,&r->rule_if,&tail) ;
            }
        status_window("Press any key to enter new consequences, Prevoi
us ones displayed above") ;
        clrscr() ;
        disp_rule(r, curr_rule_no) ;
        textcolor(BLACK) ;
        textbackground(BROWN) ;
        cprintf("\n\r   THEN (New i.e. Replacements)\n\r") ;
        textcolor(WHITE) ;
        textbackground(BLACK) ;
        fflush(stdin) ;
        getch() ;
        new_consequences = get_if_thens(&no_thens) ;
        if(new_consequences == NULL)
          all_reverse_consequence(r->rule_then) ;
        else
            {
             IF_dispose_THEN(r->rule_then) ;
             r->rule_then = new_consequences ;
```

```
                r->no_thens = no_thens ;
               }
/*        disp_rule(p, curr_fired_rule) ;
          disp_rule(r, curr_rule_no) ; */
          ++curr_rule_no ;
          add_to_rule_list(r) ;
          ++new_rules_added ;
/*        gettextinfo(&ti) ;
          window(41,3,77,23) ;
          status_window("Compare Objects ? (Y/N)") ;
          fflush(stdin) ;
          while(toupper(getch()) == 'Y')
               {
                clrscr() ;
                status_window("Enter The Previous Object") ;
                gets(prev) ;
                clrscr() ;
                status_window("Enter The Present Object") ;
                gets(present) ;
                if(!empty_stack(expl_stack))
                  clear_stack(expl_stack) ;
                var_difference_find(prev, present, expl_stack) ;
                reason_no = disp_explainations(expl_stack) ;
                status_window("Compare more Objects ? (Y/N)") ;
                fflush(stdin) ;
               }
          window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
          textattr(ti.attribute) ;
          gotoxy(ti.curx,ti.cury) ; */
          free_STACK(expl_stack) ;
          expl_stack = NULL ;
          free_each_var(head_history) ;
          head_history = NULL ;
          /*  Fire modified rule also */
          p->tot_ifs_fired = 0 ;
          single_rule_fire(p) ;
         }
     else
          {
           save_success_var_desc() ;
           update_semantic_net(p->rule_then) ;
          }
    }
 return success ;
}
int solve_premise(IF_THEN *premise)
{
 char *left_part, *right_part ;
 int len, i ;
 FRAME *frm = NULL ;
 if(premise->left_part == VARIABLE)
   {
    RULE_VARS *v = get_variable(premise->left_object) ;
    if(v == NULL)
      {
       char *name ;
       struct text_info ti ;
       gettextinfo(&ti) ;
       window(41,3,77,23) ;
```

```
        textbackground(BLACK) ;
        clrscr() ;
        textcolor(BLACK) ;
        textbackground(WHITE) ;
        cprintf("Select From One Of The Below For The Variable \"%s\" :
\n\r\n\r",premise->left_object) ;
        name = guess_var(premise->attribute+1, LEFT) ;
        window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
        textattr(ti.attribute) ;
        gotoxy(ti.curx,ti.cury) ;
        v = get_var_node() ;
        v->name = dup_str(premise->left_object) ;
        v->object = name ;
        add_var(v) ;
       }
     len = strlen(v->object) + strlen(premise->attribute) ;
     left_part = get_char_array(len+1) ;
     sprintf(left_part,"%s %s",v->object,premise->attribute+1) ;
    }
 else
     {
      len = strlen(premise->left_object) + strlen(premise->attribute)
;
      left_part = get_char_array(len+1) ;
      sprintf(left_part,"%s %s",premise->left_object,premise->attribut
e+1) ;
     }
 if(premise->right_part == VARIABLE)
   {
    RULE_VARS *v = get_variable(premise->right_object) ;
    if(v == NULL)
      {
       char *name ;
       struct text_info ti ;
       gettextinfo(&ti) ;
       window(41,3,77,23) ;
       textbackground(BLACK) ;
       clrscr() ;
       textcolor(BLACK) ;
       textbackground(WHITE) ;
       cprintf("Select From One Of The Below For The Variable \"%s\" :
\n\r\n\r",premise->right_object) ;
       name = guess_var(premise->attribute+1, RIGHT) ;
       window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
       textattr(ti.attribute) ;
       gotoxy(ti.curx,ti.cury) ;
       v = get_var_node() ;
       v->name = dup_str(premise->right_object) ;
       v->object = name ;
       add_var(v) ;
      }
    len = strlen(v->object) ;
    right_part = get_char_array(len+1) ;
    sprintf(right_part,"%s",v->object) ;
   }
 else
     {
      len = strlen(premise->right_object) ;
      right_part = get_char_array(len+1) ;
```

```
      sprintf(right_part,"%s",premise->right_object) ;
     }
 frm = get_nm_qualifier(left_part) ;
 for(i = 0 ; i < frm->no_childs ; ++i)
    if(stricmp(frm->childs[i], right_part) == 0)
      break ;
 free(left_part) ;
 free(right_part) ;
 if(i < frm->no_childs)
   return premise->attribute[0] == '+' ? YES : NO ;
 else
     return premise->attribute[0] == '+' ? NO : YES ;
}
RULE_VARS *get_variable(char *name)
{
 RULE_VARS *curr = var_list ;
 int test ;
 for(; curr ; curr = curr->next)
    {
     if((test = stricmp(name,curr->name)) == 0)
       return curr ;
     else if(test < 0)
            break ;
    }
 return NULL ;
}
RULE_VARS *get_var_node(void)
{
 RULE_VARS *p ;
 p = (RULE_VARS *) malloc(sizeof(RULE_VARS)) ;
 if(p == NULL)
   {
    printf("Out of memory.  ...ABORTING        ...(get_var_node)\n") ;
    fflush(stdin) ;
    getch() ;
    exit(1) ;
   }
 p->name = p->object = NULL ;
 p->next = NULL ;
 return p ;
}
void add_var(RULE_VARS *p)
{
 RULE_VARS *curr, *prev ;
 if(var_list == NULL)
   var_list = p ;
 else
     {
      if(stricmp(p->name, var_list->name) < 0)
        {
         p->next = var_list ;
         var_list = p ;
        }
      else
          {
           for(curr = prev = var_list ; curr ; prev = curr, curr = cur
r->next)
               {
                if(stricmp(p->name,curr->name) < 0)
```

```
                    break ;
                  }
               p->next = prev->next ;
               prev->next = p ;
              }
       }
 return ;
}
char *guess_var(char *attribute, int left_or_right)
{
 struct name_list *head ;
 char *name ;
 if(left_or_right == LEFT)
   head = left_list_rel(attribute) ;
 else
     head = left_list_rel(attribute) ;
 name = get_disp_options(head) ;
 return name ;
}
char *get_disp_options(struct name_list *head)
{
 int i,j, total = 0 ;
 struct name_list *curr ;
 char *name, mesg[120] ;
 char *txt ;
 struct text_info ti1 ;
 int x ;

 for(curr = head ; curr ; curr = curr->next)
    {
     if(!used_object_in_var(curr->name))
       {
        ++total ;
        textcolor(BLACK) ;
        textbackground(BROWN) ;
        if(total == 1)
          cprintf("(#%d)",total) ;
        else
            cprintf(", (#%d)",total) ;
        textbackground(CYAN) ;
        cprintf(" %s",curr->name) ;
       }
    }
 sprintf(mesg,"Enter the object no. (1 to %d) : ",total) ;
 gettextinfo(&ti1) ;
 window(1,1,80,25) ;
 status_window("") ;
 textcolor(WHITE) ;
 textbackground(BROWN) ;
 gotoxy(1,25) ;
 cprintf(mesg) ;
 x = wherex() ;
 i = -1 ;
 do
   {
    fflush(stdin) ;
    do
      {
       txt = get_ch_txt(x,25,80-x-2,1) ;
```

```
      } while(txt == NULL) ;
    trim(txt) ;
    if(strlen(txt))
      i = atoi(txt) ;
    free(txt) ;
    txt = NULL ;
   } while(i < 1 || i > total) ;
 window(ti1.winleft, ti1.wintop, ti1.winright, ti1.winbottom) ;
 textattr(ti1.attribute) ;
 gotoxy(ti1.curx,ti1.cury) ;
 for(j = 0, curr = head ; curr ; curr = curr->next)
    {
     if(!used_object_in_var(curr->name))
       {
        if(++j == i)
          break ;
       }
    }
 name = dup_str(curr->name) ;
 return name ;
}
int used_object_in_var(char *object)
{
 RULE_VARS *curr = var_list ;
 for(; curr ; curr = curr->next)
    {
     if(stricmp(object, curr->object) == 0)
       break ;
    }
 return curr == NULL ? NO : YES ;
}
void result_display(IF_THEN *consequence)
{
 int i ;
 IF_THEN *curr ;
 textcolor(BROWN) ;
 textbackground(WHITE) ;
 cprintf("\n\r%esult : \n\r\n\r") ;
 textcolor(WHITE) ;
 textbackground(BROWN) ;
 for(i = 1, curr = consequence ; curr ; ++i, curr = curr->next)
    {
     cprintf("%2d. ",i) ;
     if(curr->left_part == VARIABLE)
       {
        RULE_VARS *v = get_variable(curr->left_object) ;
        cprintf("%s ",v->object) ;
       }
     else
         {
          cprintf("%s ",curr->left_object) ;
         }
     cprintf("%s%s ", curr->attribute+1, curr->attribute[0] == '-' ? "
NOT" : "") ;
     if(curr->right_part == VARIABLE)
       {
        RULE_VARS *v = get_variable(curr->right_object) ;
        cprintf("%s ",v->object) ;
       }
```

```
      else
          {
           cprintf("%s ",curr->right_object) ;
          }
      cprintf("\n\r") ;
     }
 return ;
}
void res_print(void)
{
 RULE_VARS *curr = var_list ;
 clrscr() ;
 for(; curr ; curr=curr->next)
    cprintf("Variable \"%s\" was assigned Object \"%s\"\n\r",curr->nam
e, curr->object) ;
 status_window("Press any key to continue ") ;
 fflush(stdin) ;
 getch() ;
 return ;
}
void release_result_memory(void)
{
 free_var_list() ;
 return ;
}
int succ_fired(RULE *p)
{
 return (p->tot_ifs_fired > 0 && p->tot_ifs_fired == p->no_ifs + 1) ?
YES : NO ;
}
void successful_rules_view(void)
{
 RULE *curr = rule_head ;
 int no_success = 0, key, i ;
 for(i = 1 ; curr ; ++i, curr = curr->next)
    {
     if(succ_fired(curr))
       {
        ++no_success ;
        clrscr() ;
        disp_rule(curr,i) ;
        status_window("Press any key to see next successful rule .....
") ;
        fflush(stdin) ;
        if((key = getch()) == ESC)
          {
           status_window("") ;
           clrscr() ;
           return ;
          }
        else if(key == EXTENDED)
              getch() ;
       }
    }
 if(no_success == 0)
   {
    clrscr() ;
    status_window("No rule was a success.     ....Press any key to cont
inue") ;
```

```
    }
 else
     status_window("No more successful rules.     ....Press any key to
continue") ;
 fflush(stdin) ;
 if(getch() == EXTENDED)
   getch() ;
 status_window("") ;
 return ;
}
int whether_failed_rule(RULE *p)
{
 return p->tot_ifs_fired - (p->no_ifs + 1) ;
}
void fail_rule_detail(void)
{
 RULE *curr = rule_head ;
 int no_failed = 0, fail_premise_no, key, i ;
 char txt[MESG_LENGTH] ;
 for(i = 1 ; curr ; ++i, curr = curr->next)
    {
     if((fail_premise_no = whether_failed_rule(curr)) == 0)
       continue ;
     else
         {
          ++no_failed ;
          clrscr() ;
          disp_rule(curr,i) ;
          sprintf(txt,"Failed at premise #%d.     ...Press any key to s
ee next failed rule",fail_premise_no) ;
          status_window(txt) ;
          fflush(stdin) ;
          if((key = getch()) == ESC)
            {
             status_window("") ;
             clrscr() ;
             return ;
            }
          else if(key == EXTENDED)
                getch() ;
         }
    }
 if(no_failed == 0)
   {
    clrscr() ;
    status_window("No rule failed.     ....Press any key to continue")
;
   }
 else
     status_window("No more failed rules.     ....Press any key to cont
inue") ;
 fflush(stdin) ;
 if(getch() == EXTENDED)
   getch() ;
 status_window("") ;
 return ;
}
void new_rule_save_option(void)
{
```

```
 if(new_rules_added)
   {
    int choice ;
    status_window("Last chance to save new rule(s),  Save ? (Y/N)\a")
;
    do
      {
       fflush(stdin) ;
       choice = toupper(getch()) ;
      } while(!(choice == 'Y' || choice == 'N')) ;
    if(choice == 'Y')
      rules_write() ;
   }
 return ;
}
RULE *dup_RULE(RULE *p)
{
 RULE *r = NULL ;
 IF_THEN *it = NULL, *curr = NULL, *tail = NULL ;
 r = get_rule_node() ;
 r->no_ifs = p->no_ifs ;
 for(curr = p->rule_if ; curr ; curr = curr->next)
    {
     it = dup_IF_THENS(curr) ;
     add_if_then_list(it, &r->rule_if, &tail) ;
    }
 r->no_thens = p->no_thens ;
 for(curr = p->rule_then, tail = NULL ; curr ; curr = curr->next)
    {
     it = dup_IF_THENS(curr) ;
     add_if_then_list(it, &r->rule_then, &tail) ;
    }
 return r ;
}
IF_THEN *dup_IF_THENS(IF_THEN *p)
{
 IF_THEN *it = NULL ;
 it = get_IF_THEN_node() ;
 it->left_part = p->left_part ;
 it->left_object = dup_str(p->left_object) ;
 it->attribute = dup_str(p->attribute) ;
 it->right_part = p->right_part ;
 it->right_object = dup_str(p->right_object) ;
 return it ;
}
void init_for_firing(void)
{
 RULE *curr ;
 for(curr = rule_head ; curr ; curr = curr->next)
    {
     curr->tot_ifs_fired = curr->tot_thens_fired = 0 ;
    }
 return ;
}
void RULE_dispose(RULE *r)
{
 IF_dispose_THEN(r->rule_if) ;
 IF_dispose_THEN(r->rule_then) ;
 r->rule_if = NULL ;
```

```
 r->rule_then = NULL ;
 free(r) ;
 r = NULL ;
 return ;
}
void IF_dispose_THEN(IF_THEN *p)
{
 if(p)
   {
    IF_dispose_THEN(p->next) ;
    free(p->left_object) ;
    free(p->attribute) ;
    free(p->right_object) ;
    free(p) ;
    p = NULL ;
   }
 return ;
}
void update_semantic_net(IF_THEN *consequence)
{
 char *left_part, *right_part, temp[200] ;
 IF_THEN *curr ;
 for(curr = consequence ; curr ; curr = curr->next)
    {
     if(curr->left_part == VARIABLE)
       {
        RULE_VARS *v = get_variable(curr->left_object) ;
        if(curr->attribute[0] == '+')
          sprintf(temp,"%s ",v->object) ;
       }
     else
        {
         if(curr->attribute[0] == '+')
           sprintf(temp,"%s ",curr->left_object) ;
        }
     if(curr->attribute[0] == '+')
       {
        strcat(temp,curr->attribute+1) ;
        left_part = dup_str(temp) ;
       }
     if(curr->right_part == VARIABLE)
       {
        RULE_VARS *v = get_variable(curr->right_object) ;
        if(curr->attribute[0] == '+')
            right_part = dup_str(v->object) ;
       }
     else
        {
         if(curr->attribute[0] == '+')
           right_part = dup_str(curr->right_object) ;
        }
     if(curr->attribute[0] == '+')
       {
        struct name_tree *n = get_frm_ptr(left_part) ;
        if(n == NULL)
          {
           FRAME *frm = NULL ;
           frm = getnode() ;
           frm->name = left_part ;
```

```
            frm->no_childs = 1 ;
            frm->childs[0] = right_part ;
            process_frame(frm) ;
            w_frm_pr_ind() ;
            write_relations() ;
           }
        }
     }
 return ;
}
void save_success_var_desc(void)
{
 RULE_VARS *curr = var_list ;
 FILE *fp = NULL ;
 char temp[MESG_LENGTH] ;
 sprintf(temp,"%s.SAV",file_name) ;
 do
   {
    fp = fopen(temp,"w") ;
    if(fp == NULL)
      {
       int choice ;
       status_window("Unable to save History....  (R)etry or (A)bort")
 ;
       do
         {
          fflush(stdin) ;
          choice = toupper(getch()) ;
         } while(!(choice == 'R' || choice == 'A')) ;
       if(choice == 'A')
         {
          fcloseall() ;
          window(1,1,80,25) ;
          clrscr() ;
          exit(1) ;
         }
      }
   } while(fp == NULL) ;
 for(curr = var_list ; curr ; curr=curr->next)
     fprintf(fp,"%s\n%s\n",curr->name, curr->object) ;
 fclose(fp) ;
 return ;
}
RULE_VARS *read_success_var_desc(void)
{
 RULE_VARS *curr, *head = NULL, *tail = NULL ;
 FILE *fp = NULL ;
 char temp[MESG_LENGTH] ;
 sprintf(temp,"%s.SAV",file_name) ;
 do
   {
    fp = fopen(temp,"r") ;
    if(fp == NULL)
      {
       int choice ;
       status_window("Unable to open History file....  (R)etry or (A)b
ort") ;
       do
         {
```

```
                {
                 if(stricmp(histf->childs[i], currf->childs[j]) == 0)
                   {
                    same = YES ;
                    break ;
                   }
                }
             if(!same)
               {
                push(histf->childs[i],expl_stack) ;
                push("-",expl_stack) ;
                push(currf->name,expl_stack) ;
               }
           }
         else
             {
              push(histf->childs[i],expl_stack) ;
              push("-",expl_stack) ;
              push(txt,expl_stack) ;
             }
       }
     free_FRAME(currf) ;
     free_FRAME(histf) ;
     histf = currf = NULL ;
   }
for(p = curro->head ; p ; p = p->next)
   {
     currf = get_nm_qualifier(p->name) ;
     sprintf(txt,"%s %s",prev,p->name + get_verb_ind(p->name)) ;
     histf = get_nm_qualifier(txt) ;
     for(i = 0 ; i < currf->no_childs ; ++i)
        {
         if(histf)
           {
            for(j = 0, same = NO ; j < histf->no_childs ; ++j)
               {
                if(stricmp(histf->childs[j], currf->childs[i]) == 0)
                  {
                   same = YES ;
                   break ;
                  }
               }
             if(!same)
               {
                push(currf->childs[i],expl_stack) ;
                push("+",expl_stack) ;
                push(currf->name,expl_stack) ;
               }
           }
         else
             {
              push(currf->childs[i],expl_stack) ;
              push("+",expl_stack) ;
              push(currf->name,expl_stack) ;
             }
       }
     free_FRAME(histf) ;
     free_FRAME(currf) ;
     histf = currf = NULL ;
```

```
            fflush(stdin) ;
            choice = toupper(getch()) ;
          } while(!(choice == 'R' || choice == 'N')) ;
        if(choice == 'N')
          {
           fcloseall() ;
           window(1,1,80,25) ;
           clrscr() ;
           exit(1) ;
          }
       }
    } while(fp == NULL) ;
 fscanf(fp,"%[^\n]\n",temp) ;
 while(!feof(fp))
       {
        curr = get_var_node() ;
        curr->name = dup_str(temp) ;
        fscanf(fp,"%[^\n]\n",temp) ;
        curr->object = dup_str(temp) ;
        if(head == NULL)
          head = tail = curr ;
        else
            {
             tail->next = curr ;
             tail = curr ;
            }
        fscanf(fp,"%[^\n]\n",temp) ;
       }
 fclose(fp) ;
 return head ;
}
void find_differences(RULE_VARS *head_var_history, STACK *expl_stack)
{
 RULE_VARS *hist = head_var_history, *curr = var_list ;
 for(; curr && hist ; curr = curr->next, hist = hist->next)
    {
     var_difference_find(hist->object, curr->object, expl_stack) ;
    }
 return ;
}
void var_difference_find(char *prev, char *present, STACK *expl_stack)
{
 struct qual_name *histo = NULL, *curro = NULL ;
 FRAME *histf = NULL, *currf = NULL ;
 struct name_list *p = NULL ;
 int i, j , same = NO ;
 char txt[100] ;
 histo = get_object(prev, qualifier_tree) ;
 curro = get_object(present, qualifier_tree) ;
 for(p = histo->head ; p ; p = p->next)
    {
     histf = get_nm_qualifier(p->name) ;
     sprintf(txt,"%s %s",present,p->name + get_verb_ind(p->name)) ;
     currf = get_nm_qualifier(txt) ;
     for(i = 0 ; i < histf->no_childs ; ++i)
        {
         if(currf)
           {
            for(j = 0, same = NO ; j < currf->no_childs ; ++j)
```

```
                {
                 if(stricmp(histf->childs[i], currf->childs[j]) == 0)
                   {
                    same = YES ;
                    break ;
                   }
                }
            if(!same)
              {
               push(histf->childs[i],expl_stack) ;
               push("-",expl_stack) ;
               push(currf->name,expl_stack) ;
              }
           }
         else
             {
              push(histf->childs[i],expl_stack) ;
              push("-",expl_stack) ;
              push(txt,expl_stack) ;
             }
       }
     free_FRAME(currf) ;
     free_FRAME(histf) ;
     histf = currf = NULL ;
    }
for(p = curro->head ; p ; p = p->next)
    {
     currf = get_nm_qualifier(p->name) ;
     sprintf(txt,"%s %s",prev,p->name + get_verb_ind(p->name)) ;
     histf = get_nm_qualifier(txt) ;
     for(i = 0 ; i < currf->no_childs ; ++i)
        {
         if(histf)
           {
            for(j = 0, same = NO ; j < histf->no_childs ; ++j)
                {
                 if(stricmp(histf->childs[j], currf->childs[i]) == 0)
                   {
                    same = YES ;
                    break ;
                   }
                }
            if(!same)
              {
               push(currf->childs[i],expl_stack) ;
               push("+",expl_stack) ;
               push(currf->name,expl_stack) ;
              }
           }
         else
             {
              push(currf->childs[i],expl_stack) ;
              push("+",expl_stack) ;
              push(currf->name,expl_stack) ;
             }
       }
     free_FRAME(histf) ;
     free_FRAME(currf) ;
     histf = currf = NULL ;
```

```
    }
 return ;
}
int disp_explainations(STACK *expl_stack)
{
 int total = 0, choice ;
 struct name_list *curr = NULL ;
 struct text_info ti ;
 gettextinfo(&ti) ;
 window(41,3,77,23) ;
 clrscr() ;
 textcolor(BLACK) ;
 textbackground(WHITE) ;
 cprintf("EXPLAINATIONS :\n\r\n\r") ;
 textcolor(WHITE) ;
 textbackground(BROWN) ;
 for(curr = expl_stack->tos ; curr ; curr = curr->next)
    {
     cprintf("%2d.  %s",++total, curr->name) ;
     curr = curr->next ;
     if(*curr->name == '-')
       cprintf(" NOT") ;
     curr = curr->next ;
     cprintf(" %s\n\r",curr->name) ;
    }
 if(total == 0)
   {
    status_window("No differences encountered.    ...Press any key") ;
    fflush(stdin) ;
    getch() ;
    choice = -1 ;
   }
 else
     {
      char *txt ;
      struct text_info ti1 ;
      int x ;
      gettextinfo(&ti1) ;
      window(1,1,80,25) ;
      status_window("") ;
      gotoxy(1,25) ;
      cprintf("Enter the reason no. (Enter \"0\" for a new reason) : "
) ;
      x = wherex() ;
      choice = -1 ;
      do
        {
         fflush(stdin) ;
         do
           {
            txt = get_ch_txt(x,25,80-x-2,1) ;
           } while(txt == NULL) ;
         trim(txt) ;
         if(strlen(txt))
           choice = atoi(txt) ;
         free(txt) ;
         txt = NULL ;
        } while(choice < 0 || choice > total) ;
      window(ti1.winleft, ti1.wintop, ti1.winright, ti1.winbottom) ;
```

```
        textattr(ti1.attribute) ;
        gotoxy(ti1.curx,ti1.cury) ;
       }
 window(ti.winleft, ti.wintop, ti.winright, ti.winbottom) ;
 textattr(ti.attribute) ;
 gotoxy(ti.curx,ti.cury) ;
 return choice ;
}
void get_complementary_premises(int reason_no, STACK *S, IF_THEN **pre
mise1, IF_THEN **premise2)
{
 struct name_list *curr = NULL ;
 char *right_part, *left_part, *attribute1, *attribute2, verb[50], *va
riable ;
 int i ;
 char type ;
 for(curr = S->tos, i = 1 ; i < reason_no ; ++i)
    {
     curr = curr->next->next->next ;
    }
 sprintf(verb,"%s",curr->name+get_verb_ind(curr->name)) ;
 curr->name[get_verb_ind(curr->name)-1] = '\0' ;
 left_part = curr->name ;
 curr = curr->next ;
 type = *curr->name ;
 curr = curr->next ;
 right_part = curr->name ;
 attribute1 = get_char_array(strlen(verb)+1) ;
 attribute2 = get_char_array(strlen(verb)+1) ;
 if(type == '+')
   {
    sprintf(attribute1,"-%s",verb) ;
    sprintf(attribute2,"+%s",verb) ;
   }
 else
     {
      sprintf(attribute1,"+%s",verb) ;
      sprintf(attribute2,"-%s",verb) ;
     }
 variable = get_var_for_object(left_part) ;
 *premise1 = get_IF_THEN_node() ;
 *premise2 = get_IF_THEN_node() ;
 (*premise1)->left_part = VARIABLE ;
 (*premise1)->left_object = dup_str(variable) ;
 (*premise1)->attribute = attribute1 ;
 (*premise1)->right_part = OBJECT;
 (*premise1)->right_object = dup_str(right_part) ;
 (*premise2)->left_part = VARIABLE ;
 (*premise2)->left_object = dup_str(variable) ;
 (*premise2)->attribute = attribute2 ;
 (*premise2)->right_part = OBJECT;
 (*premise2)->right_object = dup_str(right_part) ;
 return ;
}
char *get_var_for_object(char *object)
{
 RULE_VARS *curr = var_list ;
 for(; curr ; curr = curr->next)
    if(stricmp(object, curr->object) == 0)
```

```
        return curr->name ;
    return "[T]" ;
    * Later ********************* > return NULL ; */

    id reverse_consequence(IF_THEN *consequence)

    if(consequence->attribute[0] == '+')
      consequence->attribute[0] = '-' ;
    else
        consequence->attribute[0] = '+' ;
    return ;
}
void all_reverse_consequence(IF_THEN *head_then)
{
    IF_THEN *curr ;
    for(curr = head_then ; curr ; curr = curr->next)
      reverse_consequence(curr) ;
    return ;
}
void new_reasons(void)
{
}
```