National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL 339 (r. 88/04) c

Canada

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

ISBN 0-315-59164-1

Programming an ISDN Intelligent Personal Workstation: An Architecture and Language

Robert D. Rourke

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montréal, Québec, Canada

June 1990

ABSTRACT

PROGRAMMING AN ISDN INTELLIGENT PERSONAL WORKSTATION: AN ARCHITECTURE AND LANGUAGE

Robert D. Rourke

The conventional approach to providing user access to ISDN, *e.g.*, an ISDN terminal, is based on a personal computer with ISDN access provided as an add-on feature. A serious shortcoming of this approach is its inability to come to grips with the issue of how ISDN users can effectively program it. As a consequence, the conventional approach does not allow ISDN users to fully exploit the information services that will be accessible *via* ISDN. Most of these services require a heuristic-based programming environment to facilitate the development of programs that deal with uncertainty and imprecision.

This thesis is that the shortcoming of the conventional approach can be overcome with a software architecture based on a knowledge-based system. An Intelligent ISDN Personal Workstation, which may be viewed as a software augmentation of the conventional ISDN terminal, provides an effective platform for creating and running user-defined knowledge-based applications that exploit ISDN information services. This report presents a software architecture for an Intelligent ISDN Personal Workstation, and, a knowledge-based language to program it.

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Dr. Tho Le-Ngoc for his guidance throughout this research, and for his advice and suggestions during the preparation of this thesis.

I was very fortunate to be working with Zenon Slodki on the ISDN Personal Workstation project. Without the operating system he built, the development of the knowledge-based system would not have been possible. I also appreciate his honesty and friendship which permitted him to constructively criticise my work and offer many useful ideas.

I thank John Rourke and Scott McKenzie for proof-reading my many *final* drafts.

I greatly appreciate the financial support provided for this project through a Canada NSERC grant A5987 and a Québec FCAR grant ER-0106, and support given to me by Teleglobe Canada through a graduate fellowship.

To users of ISDN

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

During the last decade, the amount of information available to the user through personal computers (PC) has exploded. With a PC and the appropriate network interface hardware, users can access local information through local area networks, and remote information through packet-switched networks like X.25. There is no shortage of information, but its usefulness is limited because the current level of computation available with database systems, does not provide sufficient support for informed decision making.

This is because user applications leave the bulk of processing to the user. Most of the information available to network users is stored in the form of relational databases. PC's limit the manipulation of information in a database to query functions. From a symbolic processing point of view, this functionality relates to the *word* and *relation* level of computation [1]. A database organises information as records, with each record comprising a set of fixed fields. Each field stores the same kind of data in every record. For example, a database of people would have, at least, one field for first name and another field for last name. A user, therefore, could query the database for information on a given person, but the user must interpret the data to derive a meaning from it.

We feel the user's primary need for the information contained in databases is informed decision making. One can find the level of computation required for intelligent decision making in current knowledge-based systems, and expert systems. There are already many decision making applications using this type of program, especially in medicine and engineering, but users must enter the data rather than the program obtaining the data from a network.

We foresee a day when knowledge-based applications will provide users with improved information processing and data handling capabilities. The purpose of this thesis is to research the type of processing this will entail, so we could provide an appropriate platform for these applications. We have made the assumption that in the future most people will have access to the standardised user digital network--Integrated Services Digital Network (ISDN)--*via* a basic rate interface (BRI) [2]. Consequently, the architecture of the intelligent personal workstation shall include an ISDN interface.

Having established the reason why an intelligent personal workstation should be developed, let us consider more deeply the reasons for it by examining the motivations affecting its design.

1.1. Design Motivations

In this section we will examine the two principle factors motivating the research into intelligent personal workstation. Both are related to features we feel users will expect from a personal computer connected to an ISDN network. The first feature is the ability to *exploit* (*i.e.*, fully make use of) ISDN information services. The second is the ability to support autonomous control of the local environment in an intelligent home of the future [3].

ISDN provides the workstation with a gateway for both information and communication services. From this fact stem our strongest motivations. These services (or capabilities) are made possible by the fact that ISDN will have common service access points, based on the OSI reference model. With common protocols internationally standardised, users of ISDN can expect practically universal access to a network of various commercial third-party information services. An important example of this type of service is data retrieval from huge databases.

So why is the feature, exploiting ISDN information services, an important consideration in the design of the workstation? Certainly its usefulness to the subscriber is obvious. The answer lies in the fact that this feature causes special processing requirements. An information-based application running on a workstation will have access to copious amounts of information, *via* the ISDN in order to solve problems for the user. However, it has been shown that when the amount of knowledge or data is very large, heuristics must be applied in defining a restricted set of knowledge or data to use in problems solving [1,4]. For the workstation to effectively run information-based applications that use heuristic processing techniques, the architecture of the operating system must be able to support the integration of these applications with the communication facilities.

The second motivating feature in the design of the workstation is to give users primitives for autonomous control. A computer that supports autonomous control can solve problems on its own, without the user's intervention. It can also control its local environment, which defines the physical boundaries over which the workstation can sample and/or exert control. We identify two types of autonomous control that the workstation should provide the user:

1) Remote digital control via ISDN

2) Autonomous multitasking

With ISDN providing the workstation with communication services, the potential for remote control exists. In particular, ISDN supports a low bit-rate packet switched connection that is ideal for transmitting low volume, bursty information[1]. Two computers can use this connection to exchange the small size packets needed for command and telemetry information that are essential in remote control. A user would take advantage of this ISDN-based remote control by accessing home status information and initiating control functions from another station. Clearly, the architecture of the workstation must include facilities for centralised control of the devices in its local environment to achieve the first type of autonomous control.

Autonomous multitasking, the second type of autonomous control, is the term we use to define a special class of operating system services. It is a conglomeration of capabilities that permits the workstation to perform control operations on its own. These types of operations would require some intervention if they were running on personal computers with a regular multitasking operating system. Applications requiring intelligence could use the operating system capabilities to either run in the background or run at pre-specified times, even in the absence of an operator. For example, the user could run a workstation application that transfers a large file across the ISDN late at

---

[1] The CCITT defines the D-channel in ISDN basic rate service as a 16 kb/s, packet switched channel. One use the CCITT intended for the D-channel was to carry packets containing telemetry information [2].

night. An application could also use the background capability to gather large amounts of information through the ISDN to solve a problem.

Services to support operations such as these are more than a centralisation of control devices needed for remote control. Rather, they correspond to provisions in the software architecture for an abstract set of primitives to exploit these devices. When teamed-up with a set of real-time constructs, this architecture gives workstation users a platform capable of concurrent, intelligent problem solving.

By examining the motivations behind the design of the intelligent personal workstation, two key requirements for its software architecture have been precipitated. One requirement is architectural support for heuristic processing of the information available through ISDN third-party information services. The other is to provide, in the software architecture, the management of the centralised control. We feel that a software architecture built around an **expert system shell**[1] provides a solid foundation to achieve these goals.

1.2. Design Criteria

In this section we present the design requirements for the intelligent personal workstation. The approach taken involves making a detailed examination of what we expect the workstation to be like from three principle points of view. Since the most important is the user, we start by describing what it will look like from the user's point

---

[1] We loosely use the term *expert system shell* to denote the operating system support for intelligent processing in the intelligent personal workstation. We do not use the term inference engine since, in its true meaning, it would refer to something less sophisticated.

of view. We then describe the integration of the workstation operating system and an expert system shell. This is followed by an examination of how the expert system shell fits into the *OSI* model for network protocols.

1.2.1. User's Point of View

From the user's point of view, the ISDN workstation must strike a proper balance between *usability* and *ease of use*. These two factors, in many ways, represent opposing concerns. Typically, in the world of machines, people are faced with a conundrum--the more things a device can do, the harder it is to use. For example, a wordproccesor is harder to use than a typewriter, but can do much more. The tension between these opposing concerns strongly influenced the design of the workstation.

Figure 1 illustrates how we envision the ISDN workstation appearing from the user's point of view. This diagram contains both user interface devices for *ease of use*, and external interface components for *useability* (*e.g.*, the ISDN Access and Home Control). These components for useability create the potential for automated home control and access to remote information services via ISDN. The task we face is to ensure that these potentials are fulfilled, by providing a user with a means for *effective* and *simple* control.

1.2.2. OS Point of View

Figure 2 demonstrates the hierarchical organisation of an operating system capable of supporting an expert system shell. It contains three layers. This diagram is not meant

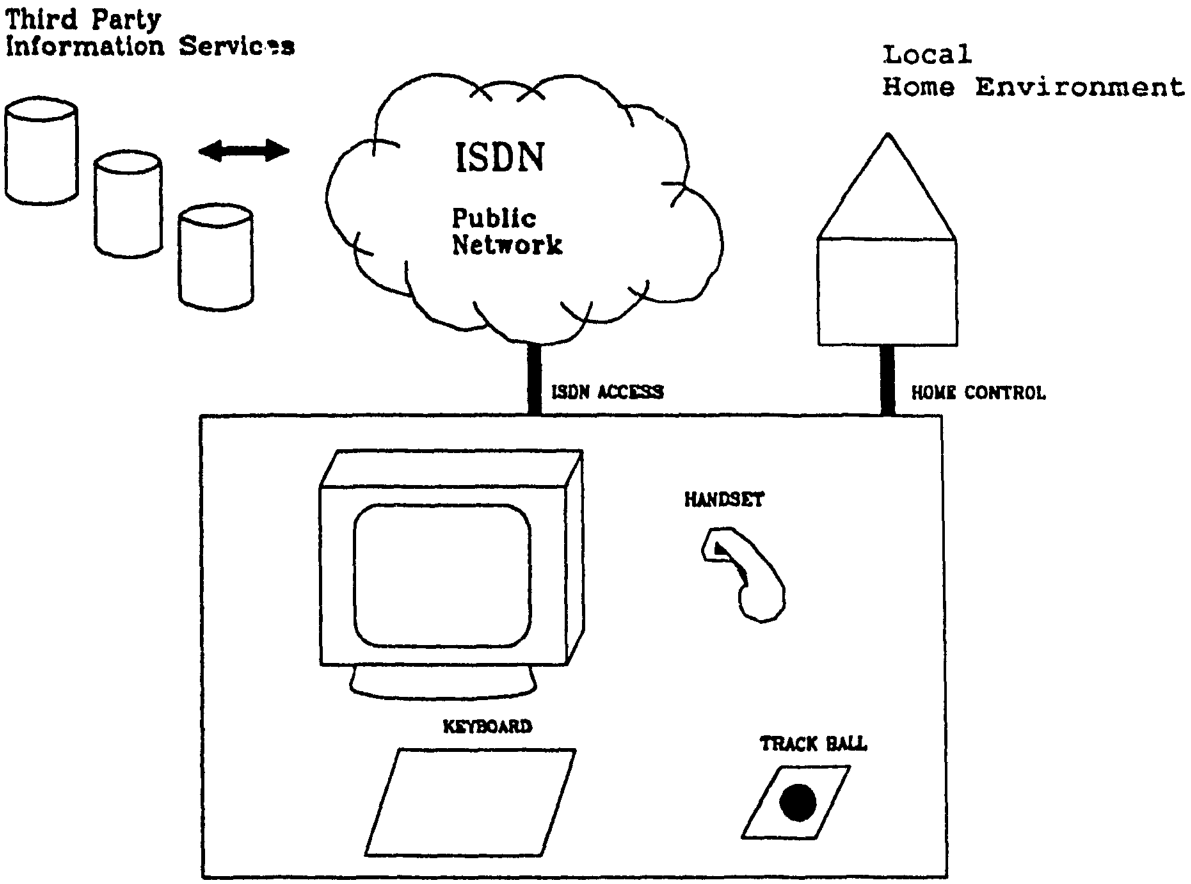
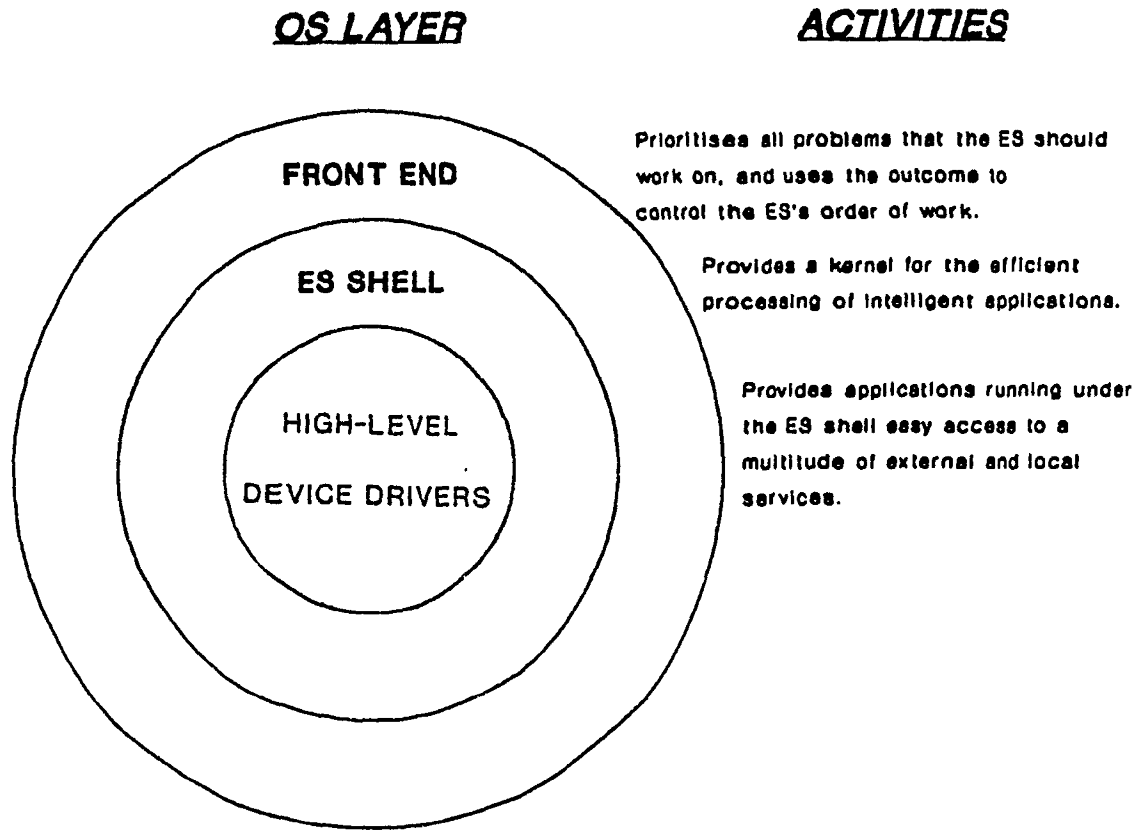


Figure 1 The user point of view of the ISDN workstation.

to be a detailed description of the actual operating system needed for the ISDN workstation. Rather, it highlights the organisation of an operating system from the point of view of the expert system shell.

We desire a workstation that is a real-time system so that it can respond to rapid changes in its environment. The most important consideration in designing an *event driven* real-time expert system is that it must be *data-driven* [5]. Described another way, the primary factor directing the activity in the expert system shell (problem solving) must be the state of its external environment. Being data-driven also implies that any change



Prioritises all problems that the ES should work on, and uses the outcome to control the ES's order of work.

Provides a kernel for the efficient processing of intelligent applications.

Provides applications running under the ES shell easy access to a multitude of external and local services.

Figure 2 Organisation of the operating system to support the expert system shell.

in *state* may precipitate a change in problem ordering. To accommodate these needs, the **Front End** must have the ability to sample the state, and use this to decide the order of problem solving in the expert system shell.

The concept behind the central layer, the **High-Level Device Drivers** layer, comes from the idea of **Device Independent I/O** [6] that is common in most layered operating system designs. A Device Independent I/0 layer of software in an operating system provides higher layers a common access point for using any I/O device that is independent of the device. Such an interface removes the complexity of using different types of devices in a system.

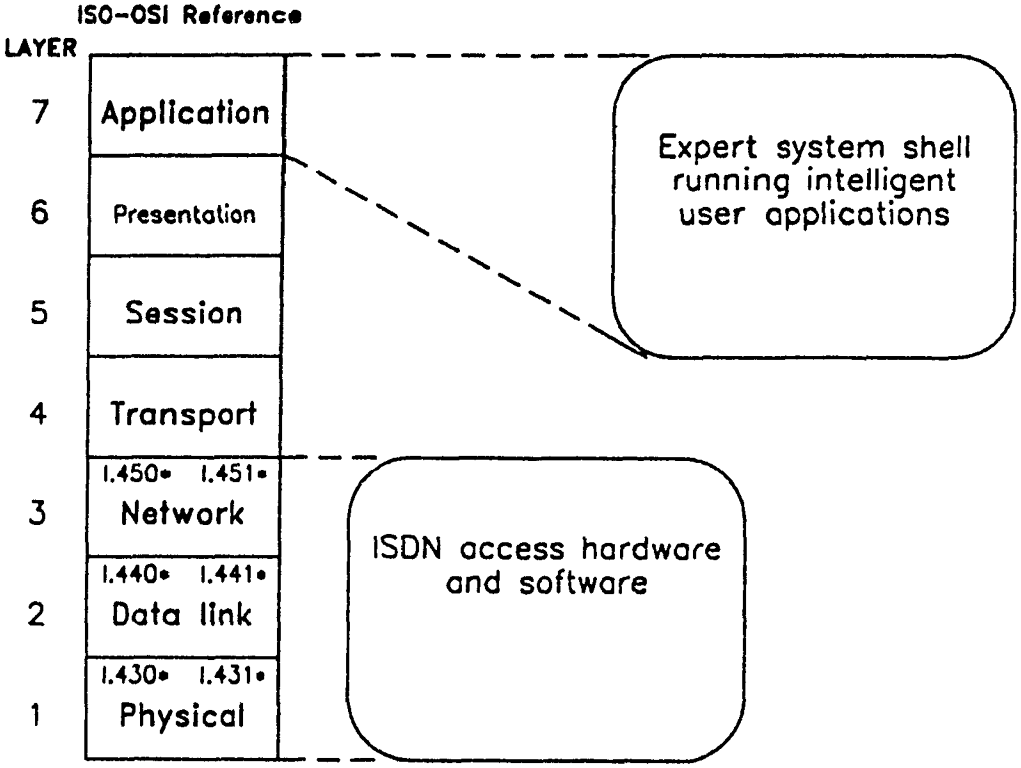
The High-Level Device Driver layer extends the idea of device independent I/O incorporating all services in a workstation. These are services that a knowledge-based system application must use (hardware, software, or communications). This layer embodies the detailed procedural-oriented software for controlling devices.

This separation, in processing between the expert system shell and the operating system, is not just done for design simplicity. A *rule-based production system* [7], the basic component of most expert systems, is a good medium for representing and using *heuristic* knowledge (*i.e.*, knowledge that gives a system *intelligence*). It is not, however, very easy to directly control hardware devices with a rule-based production system.

Production systems may not be able to control hardware devices directly, but they have been shown to provide an excellent *interface* between procedural knowledge (*e.g* device drivers) and heuristic knowledge [1]. Thus, the burden of procedural processing is off-loaded to the operating system. As a result, knowledge-based system applications running on the intelligent personal workstation will employ the lower layer of the operating system to use workstation resources.

1.2.3. Communications Point of View

The design of our expert system shell and ISDN interface falls into the ISO-OSI Reference Model. The **International Standards Organisation** (ISO) has proposed a model for layered network protocols called the ISO-OSI (**Open System Interconnection**) **Reference Model**. Further, the standards established for ISDN by the **Committee Consultative Téléphonique et Télégraphique** (CCITT) can be mapped onto the ISO model [8]. To illustrate this point, in Figure 3 we group the ISDN specifications for network access [2] into the first three layers of the OSI model.



• *NOTE: Refers to CCITT ISDN recommendations (1984)*

Figure 3 Mapping of the Workstation resources onto the OSI-ISO Reference Model.

The application layer of the ISO-OSI model contains routines for performing general-purpose and special-purpose tasks [8]. In the ISDN workstation, the inference engine is a general-purpose task for establishing what and when something should be done on the network. It also augments basic services with concise *rule-sets* (*i.e.*, a small knowledge-based system program). These rule-sets can gather specific classes of information for user applications, from the network. Thus, the expert system shell maps into layer 7 of the ISO-OSI reference model.

The CCITT have made no formal ISDN definition for layers 4, 5 & 6. These depend on third party protocols for accessing information services like databases, home shopping, or even access to other networks [8,9]. It is not in the scope of this thesis to discuss the problems with the standardisation of these layers, but we feel this problem certainly is not trivial.

1.3. Plan and scope

In this section we present the scope of the thesis research, the initial research plan, and the thesis outline.

1.3.1. Limitations of an ISDN terminal

Underlying the solution we adopt for designing an intelligent personal workstation is the belief that current ISDN terminals are not suited to deal with the problem. Most ISDN terminals are just regular personal computer (PC) augmented with an ISDN BRI interface [10,11,12,13]. Consequently, ISDN terminals operate under the control of a regular PC operating system.

As we previously discussed, PC operating systems do not run applications that can provide users with intelligent decision making support. This is because PC operating systems do not have the architectural framework needed to sufficiently integrate these applications with an ISDN communication facility. Further, PC operating systems do not directly support the level of symbolic processing needed by intelligent applications. Thus, the problem of creating an ISDN workstation shall not be solved by imbedding ISDN access hardware and software into a PC, but requires a modification to this architecture.

1.3.2. Solution

The solution we do adopt is to develop a new software architecture for the basic ISDN terminal. We call the new system an **intelligent personal workstation** to emphasise our desire to provide intelligent services to the end-user. One may consider this type of system an augmentation of the capabilities of a conventional ISDN terminal. Before starting the research into the software architecture of an intelligent personal workstation, we foresaw a number of problems that must be addressed in the thesis, they are:

1. The development of a mechanism that permits the gathering of external information for intelligent decision-making applications.
2. Investigate the problem associated with using this external information in a knowledge-based system. In particular we must consider factors like temporal dependency and uncertainty.
3. The system must have a mechanism that facilitates the transfer of problem solving skills from human experts into the intelligent personal workstation. This will require the development of a programming facility.
4. We must structure the software architecture so that it can operate on its own in an autonomous mode.

1.3.3. Outline

We proceed as follows. In Chapter 2, the characteristics, and a formal architecture of the intelligent personal workstation is presented. In Chapter 3, we present details of the **KOOLA** (Knowledge-Based Object-Oriented Language) language for programming the workstation. In Chapter 4, the formal language constructs for the KOOLA production system are given, and ends with an example of a KOOLA application. In Chapter 5, we examine the implementation of an experimental intelligent personal workstation that we call the *ISDN Workstation*. In Chapter 6, we summarise our conclusions, and discuss future work.

CHAPTER 2

ARCHITECTURE OF AN INTELLIGENT PERSONAL WORKSTATION

In this chapter we present the software architecture of an intelligent personal workstation by examining its characteristics, then its realisation.

The idea of an intelligent personal workstation was presented in the Introduction. Generally, we defined it as a personal computer--connected to an ISDN network--with a software architecture built around a knowledge-based system. In this chapter, we will examine the characteristics of an intelligent personal workstation and then its software architecture.

2.1. Characteristics of an Intelligent Personal Workstation

The two primary features expected from an intelligent personal workstation, that were presented in the first chapter, are:

- An intelligent workstation should help users exploit ISDN services. In particular, this feature includes the ability to access and use third party information services.
- An intelligent personal workstation should have the capability of supporting autonomous control of the user's local environment (*i.e.*, devices in his or her own home). To accomplish this, an intelligent workstation must have the intelligence to work in the absence of the user.

From these features we extract the characteristics of an intelligent personal workstation. These characteristics naturally precipitate when we consider these features more deeply. We will now examine, qualitatively, the four characteristics of an intelligent workstation.

2.1.1. Control Workstation Resources

The most important characteristic of an intelligent personal workstation for the autonomous control feature is its ability to control resources. In particular, a knowledge-based system running on the workstation, used for autonomous control, would have to control the local environment. To do this, however, requires special hardware and software interfaces in the workstation. At the same time, the knowledge-based system would have to be able to use those interfaces.

A knowledge-based system's inference engine processes *declarative knowledge* on how to solve problems [1,14,7]. It is not an appropriate place to embody procedural knowledge [1]. The software interfaces needed for autonomous control is an example of the type of procedure knowledge which the workstation must use. Therefore, not all of the knowledge processing can be done by an inference engine--there must be another mechanism for processing procedural knowledge.

A characteristic of an intelligent personal workstation is that it must facilitate the use of both declarative knowledge and procedural knowledge. But since decisions made by one part of such a system (an inference engine for example) can affect the operations in another part of the system (*e.g.*, a device driver), the different parts must communicate effortlessly.

The feasibility of any intelligent workstation design hinges on the whole system's ability to control the local environment. No matter how much intelligence a system obtains internally, if it cannot control external objects, then from the point of view of a computer for autonomous control, the whole system is quite impotent.

2.1.2. Handles Real-Time Information

Much of the information handled by an intelligent workstation connected to an ISDN is time dependent, or *real-time* [4]. This is especially true for data used in an information application.

A system that manages ISDN information services should strive to minimise the expense of using the network (*i.e.*, be *frugal*). If designed this way, the intelligent personal workstation can be considered a *frugal network user* [15]. When the cost of network communications is not free, a frugal network user will try to minimise its use of this resource.

The intelligent personal workstation should minimise its use of the ISDN resource. It can accomplish this by not requesting information that it previously received. The only difficulty in following such a regiment is; that information already received may become outdated, and no longer accurate.

Facts used in decision making remain valid (true) for a finite length of time. For example, information about the availability and cost of certain resources may change daily. Other information, however, may only change in the order of weeks or even months, still some in the order of minutes or hours such as stock-market information. The length of time a fact remains valid depends on the domain and meaning of the fact.

Any part of an intelligent workstation that processes time dependent data should be characterised as having real-time primitives. It is also the case that a knowledge-based system would make much use of time dependent external data in the form of facts. Consequently, the characteristics of its fact handling parts, must be built with the ability to perform real-time information processing.

2.1.3. Easy to Use and Program

From the point of view of the end-user, the degree of simplicity with which he or she can operate or program a computer is most important. Therefore an intelligent workstation should be easy to use and easy to program.

We quantify "easy to use" in terms of how many sub-steps a system can perform on its own, on behalf of the user. For example, consider the case of a user wishing to receive a file from an external database. Performing such a transfer starts with the creation of a communications link, then establishing end-point protocols for the file format, and finally actually sending a file (similar to using a PC and a modem). If the system takes on the burden of doing these sub-steps, then receiving a file is greatly simplified for the user. In fact, it may even be impossible for novice users without intelligent assistance . A characteristic of an intelligent workstation, therefore, is that it should have the intelligence to autonomously perform most common computations on its own.

The user also requires a means to easily write programs which control the local environment and access workstation facilities. To do this, she or he needs a special programming language. The language must be able to exploit the high-level control primitives of the intelligent workstation and the real-time primitives of the operating system. For example, a program to send a file at night would need primitives for controlling the ISDN connection. Also, real-time constructs for specifying the time to send the file are needed.

A second characteristic of an intelligent workstation's language is that it should be universally applicable throughout the entire workstation *environment*. This applies to both the system's programming environment and its interactive command interface. There should be a single, intelligent interface that can be utilised to modify applications or write new ones. We feel the language should have features similar to those found in interpretive BASIC on an IBM PC [16], where the same set of commands are used interactively and in writing programs.

2.1.4. Support for Multiple Problem Solving

The local environment of an intelligent workstation will be the source of many *events* and *problems*. In terms of the local environment, we define an *event* as an unpredictable state change in the local environment, culminating in some form of signal to the workstation. The state change could be a broken window in the home, and the corresponding signal, an alarm. Considered at a higher level of abstraction, an event becomes a *problem* that must be processed. This is in accordance with the previous definition of a problem as the processing required to diagnose an event and determine what actions (if any) to initiate in response to the state change. A common characteristic of events is, from the point of view of software running on the workstation, that they are *stochastic* (random with respect to time).

The changes in local state, signalled *via* the occurrence of events, may have different levels of importance. A fire alarm, for example, is more important than a signal that someone just entered the home. There is no guarantee that the inter-arrival time of events (*front-end granularity*) will not exceed the speed with which the intelligent workstation can process all of them. To deal with this conundrum, the front-end must be able to prioritize problems and process them concurrently. Thus, the fourth characteristic of an intelligent workstation is the ability to process asynchronous problems concurrently, while considering their importance.

2.2. Architecture of an Intelligent Workstation

We now propose a possible architecture for an intelligent personal workstation, based on a distributed knowledge-based system design. In our implementation, major software components are distributed, with some communication ability needed between certain components. Central to the design of each component is its main method of knowledge representation.

2.2.1. Basic Structure

The structure of the architecture is similar to the **Expert Manager** proposed by J. Pasquale [17], which he developed to manage large distributed computer networks. In particular, we base the architecture of the workstation on the control structure he calls an *observe-reason-act*[1] loop, shown in Figure 4. Each of the blocks in the structure (observe, reason or act) corresponds to major steps in the control system. **Observe** corresponds to detecting and measuring external information. **Reason**, is the high-level manipulation of the observed information. **Act** is the process of performing an external action based on the outcome of the reasoning step.

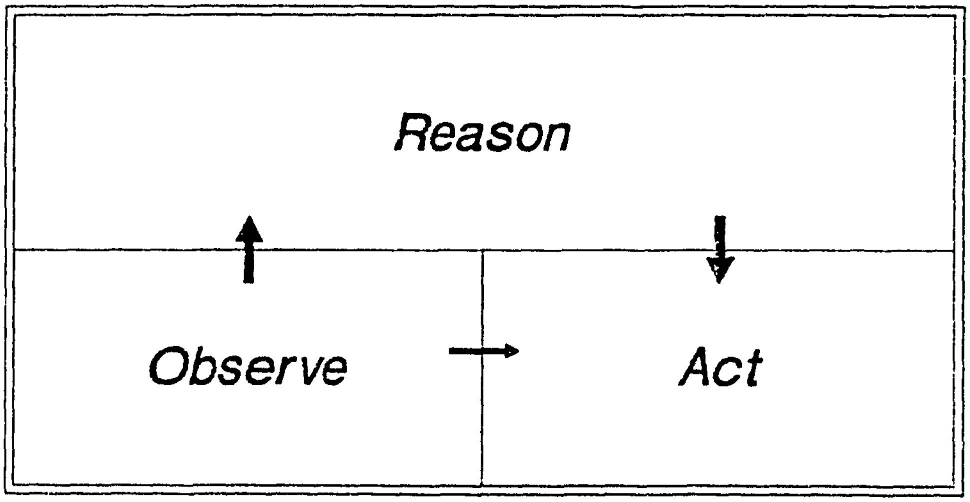


Figure 4 The observe-reason-act loop on which the intelligent workstation architecture is based.

---

[1] The *observe-reason-act* control structure is an extension of the classical *observe-act* structure used in control systems. For more information on this structure the reader is directed to Pasquale's Ph.D. dissertation [15].

There are two possible paths for the flow of activity in the contol structure shown in Figure 4. One path goes directly from *observe* to *act*. This would correspond to conventional processing in the workstation that would be supported by an operating system. The second path (shown with heavier arrows) corresponds to the workstation's intelligence, and constitutes the discussion in the remainder of this section.

We are most concerned with how such a system will help people make better use of controllable devices in their homes, especially an ISDN subscription. Consequently, the goal of this architecture is to provide people with a framework for constructing intelligent autonomous applications that help them exploit ISDN information services. For this reason, the ability of the system to interface with hardware devices is as important to this architecture as its intelligence.

Figure 5 illustrates the architecture of our intelligent workstation. It is based on the observe-reason-act loop defined above. The Belief Manager (BM) and the Experiment Generator (EG) correspond to *observe*. The Inference Engine (IE) provides the system *reasoning*, and the Action Generator (AG), with the ability to *act*.

To see how this architecture maps onto the observe-reason-act loop, we can trace the flow of information and computation through the system. To begin with, the observations and processing required to sample facts are done by the EG. These new facts are stored by the BM, which gives all the new facts a time-stamp. Operating with the rules located in the two knowledge bases, the IE uses these facts to infer new beliefs (which subsequently, are returned to the BM for maintenance). The IE then decides which actions to take, and informs the AG. The AG receives these requests for actions,

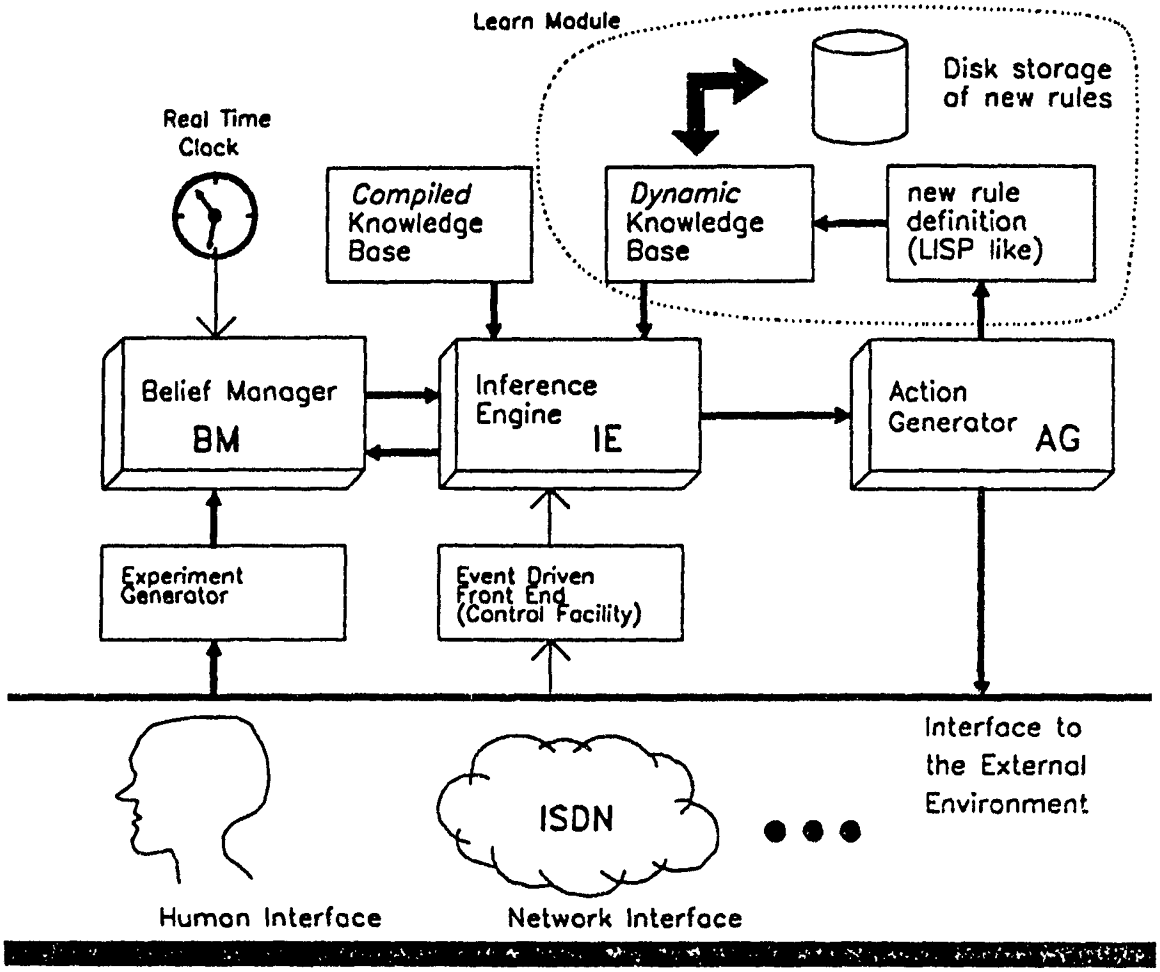


Figure 5 Architecture of an intelligent workstation based on an expert system design.

and performs the processing needed to complete them.

We will now take a closer look at the three main blocks that were introduced in the previous flow (BM, IE and AG), as well as the learn module.

2.2.2. Belief Manager

The belief manager is responsible for supporting the system's real-time data handling requirements. The belief manager detects invalid (*i.e.*, spoiled) beliefs, by keeping track of how old its supporting facts are. A method currently used in network expert systems, to do this, is to time-stamp dynamic data [5], which is also how our BM detects expired information.

The experiment generator is associated with BM and is used to obtain information. When a programmer writes a procedure, for the EG, to gather information, he or she decides how long the *fact* (data collected as a result of running the procedure) will remain accurate. This time estimate is always stored with the data-value of a fact in the BM.

The length of time the programmer estimates a fact will remain valid is referred to as its *preassigned expected shelf-life* [5]. From the point of view of the belief manager, this parameter is pre-assigned to it by the programmer. It is also the expectation (or the average) of the length of time it should remain valid. The use of the term shelf-life comes from the idea of a consumer products that can only be stored for a limited amount of time on the shelf of a store.

The belief manager monitors the aging of facts by comparing the arrival time-stamp with the preassigned expected shelf-life. When the inference engine asks for a fact that has spoiled, the belief manager forwards the request to the experiment generator, which then reacquires the fact.

A second important function of the belief manager is to maintain a consistent set of global beliefs. The use of global beliefs improves the efficiency of a knowledge-based system because beliefs can be shared among different problems. Duplicated effort in acquiring data for the beliefs is eliminated if they can be used by multiple applications.

To support global belief storage, a memory structure consisting of one global pool and many individual local pools is defined. Any inference may add beliefs to the global pool, providing the rules used to develop the belief are part of the global rule-set[1]. If. however, the belief is locally defined, then the belief is stored in that inference's private pool. This ensures consistency of beliefs within an environment of shared beliefs.

2.2.3. Inference Engine

The mechanism for inferring new beliefs (problem solving) is provided by the inference engine. The inference engine controls the processing of declarative knowledge that is usually coded as rules[2]. By working with probabilities and using heuristics, the inference engine can deal with and use both incomplete and potentially inaccurate (*i.e.*, fuzzy or noisy) facts to form conclusions (beliefs).

A requirement identified for an intelligent workstation is that it must support multiple concurrent inferences[3]. This means that it can work on more than one problem at a time. But since, in our architecture, there is only a single inference engine, this requirement is not immediately met. The solution we use involves giving the inference engine a problem schedular that is separate from its rule schedular.

The problem schedular maintains a queue of all active inferences (one for each problem). It uses round-robin arbitration to select which problem to work on next. The

---

[1] For a compete description of rule-sets the reader is directed to Section 4.1.3, which discusses the KOOLA *variable construct.*

[2] Production rules allow programmers to translate heuristic information into a tangible and precise computer format. The rules embody specific information corresponding to human problem solving skills.

[3] Inference is defined as the processing done by an inference engine for each application it runs.

problem schedular is a fair schedular. This ensures that low priority problems are not starved. Rather, they receive proportionally less processing time.

The facility for external control is the part of the problem schedular which receives the priorities from the operating system. The operating system can start a problem, change the priority of a problem, or terminate a problem through it.

The component of the operating system that would use the facility for external control would be an event-driven schedular. In Chapter 1 we situated this operating system components in a layer above the knowledge based system. Its task is to receive external events, and do the processing needed to start knowledge-based application. In the software architecture of the intelligent personal workstation, this layer is called **HALOS** (High Level Operating System Schedular) [18].

2.2.4. Action Generator

The requirement for controlling the workstation interfaces is met with the action generator. Once the inference engine decides to perform a certain action, the mechanism to translate the desire into an actual operation involves the action generator. The AG performs operations by issues commands to an appropriate operating system service routine. Next, we will show how the action generator can be viewed as an engine that processes object-oriented procedural knowledge.

We have made the action generator a separate entity from the inference engine, thus providing a means to express the knowledge of how to control external objects, different from the heuristic representation supported by the inference engine. Current research on graphical user interfaces and network modelling suggests that the set of paradigms available with object-oriented programming techniques is suitable to capture interfacing knowledge [19,14]. A major influence in the action generator's design came from our decision to express this knowledge in object form.

The attraction of using object-oriented design for external interfaces is that interfaces developed as class objects may be easier to use than procedural-based ones. A key to this is that users (final programs) do not access low-level code (such as an interface's initialization procedures) or private variables (used to maintain such concepts as states) [19]. Rather, users access public functions that are defined for the object's class which hide much of the object's details. This makes the interface between the inference engine and the action generator very simple.

2.2.5. Dynamic Modification

The three blocks in Figure 5, circumscribed with a dotted line, encompass the intelligent personal workstation's ability to be modified by the end-user. This is used if the user decides she or he would like to add or modify rules in the production system. The reason could be to customise a large application, or create a personal one. Either way, new rules are entered through a special user interface that checks them for global consistency and verifies that their consequences do not conflict with older rules in the production system. Once a rule is validated, it is placed into the **dynamic knowledge base.** Finally, for long term storage, the user can save a new rule-set as a data-file.

Our architecture derives the important characteristic of being "*easy to program and use,*" as defined in section 2.1.3, from this module. We say this because of how it provides user/programmers access to system capabilities, and its intelligent user interface.

Programs coded by rules in the dynamic knowledge-base run on the inference engine. The inference engine can access and use any information in the workstation This includes information on how to gather facts and start actions; also, the heuristic knowledge maintained in the *compiled knowledge base* is available. Since programs written by the end-user reside in the dynamic knowledge-base, these programs may access all the workstation's resources.

Writing rules is almost the same as issuing commands to the system. To illustrate this point, we will show an abstract rule, and an abstract interactive command.

A single-rule knowledge-based system to send a file at night could be written as:

IF it is later than 12 pm

THEN send *myfile* to *myoffice.*

An interactive command to send a file is almost the same as the consequence of the previous rule. For example, the abstract command to send a file would be:

Send *myfile* to *myoffice.*

These two actions are coded the same. Therefore, if the user knows how to send a file interactively, he or she can will know how to write a program that does it automatically

2.3. Conclusion

In the future, personal computers with ISDN capabilities shall be more responsive to user needs. This will be precipitated by software technologies similar to those used in expert systems. We believe that systems will appear with characteristics similar to those outlined in section 2.1. These new systems--intelligent personal workstations--shall do more for the people they serve by supporting autonomous control of their home environment.

Autonomous control will spread to the management of future communication resources such as ISDN. With knowledge processing primitives, intelligent workstations will permit people to exploit the information potential inherent in these geographically large networks.

This chapter serves to sketch out a conceptual personal workstation. We have developed an experimental system to test the feasibility of a personal workstation design based on the integration of a knowledge-based system with an ISDN connection. Chapter 5 provides details pertaining to an implementation of the knowledge-based system shell for the workstation.

CHAPTER 3

THE KOOLA PRODUCTION SYSTEM:

BASIC CONCEPTS

In this chapter we define the basic concepts behind a production system for programming knowledge-based applications on an intelligent personal workstation.

Previously (in Chapter 2), we developed a model for an intelligent personal workstation capable of controlling external devices like an ISDN connection. In this chapter and the next, we elaborate on this model by developing a language to program it. This development leads to a refinement of the model through the specification of a set of language constructs, that must be supported by a *run-time implementation of the model* (or *run-time kernel*).

3.1. Introduction to KOOLA

The name "KOOLA" is an acronym for **Knowledge-Based Object-Oriented Language.** We say that it is a knowledge-based language because it is a means for gathering and representing general declarative knowledge [1]. Because KOOLA only comprises a set of classes which are used to declare new objects, it is also an object-oriented language [19]. A full justification for using these two attributes to define KOOLA will be made in the next two chapters.

Like any language for developing knowledge-based systems, KOOLA supports the separation of knowledge which pertains to a specific problem domain, from the control information which specifies the use of this knowledge [15]. The *if-then* rule format is the knowledge representation method used in KOOLA to encode this domain-specific knowledge. For these reasons we can designate the KOOLA programming language as a *Production System* [1]. We commonly refer to the KOOLA language as the KOOLA production system to make the distinction between it and traditional procedural languages like C, Pascal or LISP.

The KOOLA production system is a program that facilitates the development of knowledge-based applications. *Templates* are used within the production system for all programming. All objects in KOOLA are defined by interactively filling in a template on the screen of a computer. The objects are automatically stored once they are entered, and may be revised at any time.

Another part of the production system is the KOOLA compiler. This program compiles the source code of knowledge-based application into a format that can be processed by a run-time kernel. At the same time, this process optimises the storage of the knowledge. Optimising involves stripping most of the symbolic references, and replacing them with numerical references. It also involves pre-sorting the references to improve searching. Therefore, the compiler reduces memory requirements and improves processing time of a final application.

The name we use for a knowledge-based application developed using the KOOLA production system; compiled with the KOOLA compiler; and, ready to run on an intelligent personal workstation is: a *knowledge-based system*. A knowledge-based system uses its compiled knowledge to solve specific problems for the end-user of the workstation. In this context, we say that the run-time kernel gets assigned *problems* to solve, much like an operating system schedular gets assigned *applications* to run.

An *inference* is the type of processing done by a run-time kernel while it solves an assigned problem. This processing involves using rules in a knowledge-based system and *facts* (*i.e.*, easily obtained data) to derive conclusions about the workstation's external environment. Conclusions are used by the run-time kernel to decide when to start new actions, according to the application.

3.2. Knowledge Representation in KOOLA

A production system must be built from an expressive knowledge representation scheme for its domain-specific knowledge. The KOOLA production system tries to satisfy this essential requirement with an infrastructure that incorporates two knowledge representation methods. These methods correspond to the KOOLA **rule** and the KOOLA **goal** class. They permit the gathering and storage of specific information on human problem solving skills. Since they are based on a symbolic processing model, we must understand this model in order to examine these constructs more deeply.

In symbolic processing, multiple schemes for representing knowledge are better understood in the context of a general *knowledge processing model* [1]. Within such a model, a distinction is made between *meta-level knowledge* and *processing knowledge*. *Processing knowledge* corresponds to the type of procedural knowledge embedded in software procedures like a KOOLA run-time kernel. In general, it is knowledge to the use of domain specific data. For the run-time kernel, it pertains to the ability to manipulate *facts*.

Analogous to the relationship between *processing knowledge* and its data, *meta-level knowledge* is knowledge pertaining to the use of *processing knowledge.* Since *meta-level knowledge* is knowledge about using other knowledge the *meta* prefix is used. In KOOLA, the **goal** and the **rule** are *meta-level knowledge* representation schemes. Their knowledge is used to manipulate the run-time kernel.

The next concept in the *knowledge processing model* we need to examine is that *meta-level knowledge* may be organised into a *hierarchy of meta-level knowledge.* This means that *meta-level knowledge* at one level directs the use of *meta-level knowledge* at a lower level, but both still direct the use of *processing knowledge.*

As the reader may have already surmised, the knowledge represented by the KOOLA **rule** and **goal** also form a *hierarchy of meta-level knowledge.* Knowledge represented by **goals** direct the use of **rules** by establishing objects to be solved with rules. These objects are called goal **beliefs**, which will be covered in depth in the next chapter.

The two knowledge representation schemes in KOOLA correspond to two different symbolic processing requirements. In the section on rules, we will see that **rules** are best suited for gathering *uncertain heuristic* knowledge. In section 4.3 we will see how **goals** are able to control an application's flow, and why we refer to them as *meta-rules.*

3.3. Uncertainty Modelling in a Knowledge-Based System

This section examines the effects of uncertainty in a knowledge-based system running on an intelligent personal workstation. It also includes the default uncertainty model defined for the KOOLA production system, and explains the algorithm used by the default model for inferring beliefs.

3.3.1. Symbolic Processing

A knowledge-based system is a program that is used to solve complicated problems. When the problem solving ability of a knowledge-based system approaches the level of a *human expert* on a particular subject, the program can be classified as an expert system [14]. To achieve this intelligence, a knowledge-based system must internally store all of the information needed to solve its specific problems. Consequently, the main activity of a knowledge-based system can be viewed as the processing of this information.

What is the nature of the information that a knowledge-based system must internally store? It can be considered a symbolic representation for the *real world idioms* that define a problem [14]. The term "*real world*" refers to something physical or conceptual that exists outside of the computer program. For example, a *real world* concept that a knowledge-based system must represent is **knowledge**, and the *idiom* used is the **rule**. In this sense, an *idiom* is the symbolic representation of a *real world* concept. Since knowledge-based systems employ these types of symbolic representations, they have been classified as symbolic processing applications [1].

Current research in symbolic processing indicates that the most important requirement for any symbolic processing application is the ability to perform computations with information that may be:

1) ***Uncertain***

2) ***Incomplete***

3) ***Conflicting*** [1].

How these *characteristics* of **information** can affect a knowledge-based system is examined next.

3.3.2. Symbolic Information

In the context of symbolic processing, the term *information* has a specific meaning. Generally, information is something that can be stored digitally in a computer. We divide information into two classes. The first class is knowledge which can be declarative (*e.g.*, rules) or procedural (*e.g.*, a software routine). The second class is data, that applications use and store in a computer. In fact, "Knowledge can be considered [as] data at a high level of abstraction...[4]", therefore they must share some common attributes.

Incompleteness, the second characteristic of symbolic information listed above, deals mostly with information in the context of knowledge processing. The problem of **incomplete knowledge** often occurs in AI applications. This is due to the nature of AI applications. In particular, developers of AI systems have incomplete knowledge of the problem to solve, and subsequently, how to solve it. If they did (*i.e.*, have a definite algorithm), then there would be no AI requirement [7]. For such problems, we employ heuristics to express our partial understanding of events. Of course, a knowledge-based system provides useful paradigms for processing incomplete heuristic knowledge.

Applications using symbolic processing deal with **conflicting information**. This is mostly a problem when rules are added to the system dynamically. Since the knowledge base for a KOOLA application is verified by the compiler, and there is no provision for default reasoning, this is not a problem for the KOOLA production system.

With respect to the workstation's knowledge-based system, the most important symbolic processing characteristic is the requirement to deal with **uncertain information**. Unlike the previous two requirements, this one includes both classes of information (facts and knowledge). This requirement deals with uncertain facts, as well as uncertainty in the knowledge. The consideration of these two symbolic processing requirements, dictates the default uncertainty model for KOOLA.

Information in the form of facts alone may not always be certain for a symbolic process. For a system working from a large distributed network (an ISDN for example) the uncertainty may depend on when it received the fact. With an OSI communication protocol based on error checking, a process can assume that the raw-data received is always correct, but the symbolic attribute of the information (meaning) may be noisy

Another way uncertainty gets introduced into the facts used by a system is if they are limited. That is if the system does not use all facts pertaining to the problem. This occurs if the domain of facts is too, or even infinitely large [4]. In such cases only a subset of the facts are available to the symbolic process. The information contained by such a subset of facts is incomplete and becomes uncertain because of this limitation.

This problem is also seen when dealing with human sources of information. The expert system MYCIN [20], for example, lets doctors assign certainty factors to observations. A doctor could say "I am 60% certain that ...". This is referred to as symbolic uncertainty, and means the same as "I am fairly certain...". Fuzzy logic is a symbolic primitive that can deal with this type of uncertainty in facts [22].

The second class of uncertain information in symbolic processing is uncertainty in an application's knowledge. This may stem from the problem of **incomplete information**. If one does not completely understand a problem, or can not list all possible logical relationships defining its solution (it may be too large), then the knowledge she or he expresses will be a limited subset of the full knowledge of the problem. As in the problem of limiting the set of facts, this introduces uncertainty into the knowledge.

Thus, for this situation there must be a means to express uncertain information in a symbolic application. For a production system using rules, there should be way for a programmer to express the certainty (or uncertainty) of his or her knowledge in the rule syntax. Also, a means of dealing with uncertain facts is needed. The approach we used to deal with this problem in KOOLA involves using ideas from probability theory and Baysian statistics.

3.4. An Uncertainty Model for KOOLA

Whatever method a knowledge-based system has to deal with uncertainty (*i.e.* its *uncertainty model*), the choice of model usually influences two important sub-systems in the overall system. In terms of the run-time kernel, or more precisely its inference engine, the uncertainty model dictates how uncertainty values (belief factors, probabilities, weights, etc...) are combined through rules. The uncertainty model also influences the syntax of the knowledge representation scheme used to codify heuristics. The objects (rules for example) must embody all of the necessary parameters required by the inference engine. At the same time, the objects must facilitate knowledge engineering. How an inference engine combines uncertainty values for a KOOLA knowledge-based application is examined next.

We have defined a default mechanism for solving probability inferences in KOOLA that uses both the **probability** and **weight** parameters stored with KOOLA rules. The structure of the default technique and the *inference algorithm* it employs; is framed by the environment the KOOLA knowledge-based system must operate in--*an event driven, real-time, communications and control environment.*

The default mechanism is based on a weighted average algorithm that assumes that rules form a medium to express an *a priori* probability that indicates how one belief affects the probability of another. We call the dynamic probability values calculated by the inference engine an *accumulative probability*, because it is accumulated from all supporting rules. **Rules** that affect a belief's probability are either fact based or belief based (generating primary or secondary beliefs), therefore, there are two variations of the algorithm.

3.5. Fact Based Algorithm for Inference

An examination of how a fact-based rule can contribute to a belief's *accumulative probability* provides a useful framework for obtaining the salient aspects of the fact-oriented algorithm.

3.5.1. Effective Rule Sets

Of course, the inference engine does not employ all the rules in a full-size knowledge base to solve a belief. Those that it does, however, are said to be members of the belief's *effective rule-set*. Membership requires that the fact/belief relationship defined by the rule meet the following two requirements:

1. *Static Requirements:* The belief must be specifically defined in the consequence of a rule. This is done by the knowledge engineer.
2. *Dynamic Requirements:* After solving the antecedent of the rule (by comparing some facts), the rule must still infer some information pertaining to the belief.

The first requirement is met ahead of time, and is flagged by a KOOLA compiler. The most important operation performed by a KOOLA compiler in creating a knowledge-based application is building an inference network. In this context, an inference network is a data structure that, with the help of the new rules, logically combines all beliefs in an application forming a *directed graph.* Imbedded in this type of data structure are all *rule-sets*, for all beliefs in the application, that an inference engine would work on. This information is imbedded in the data structure, because for all beliefs, there is a set of pointers (based on the rules) that identifies all of the belief's supporting elements (facts or beliefs). For any belief, this set of pointers, which are defined by the knowledge engineer's rules, are called the belief's *rule-set*, because they definitely meet the first membership criterium.

It may be the case that not all rules in a belief's *rule-set* will end up helping to solve the belief's *probability* when the system is finally running. In other words some rule/fact pairs meet the first criterium, but not the second. This results from the problem that there is no way of knowing the outcome of an experiment, or the value a fact may acquire, before a knowledge-based system is actually running an application. Consequently, not all promising rules (elements in the rule set) checked by an inference engine yield useful information concerning the belief that the inference engine is working on. A fact may not be available, or the outcome of the rule's antecedent may establish a condition in which the rule cannot infer anything about the belief (e.g. a false outcome with no ELSE clause). We are now ready to see how a set of successful rule/fact pairs infers a new belief.

3.5.2. Calculating a new belief from facts

An inference engine using the KOOLA *inference algorithm* to calculate a primary belief, may only employ the elements in the belief's *effective rule-set*. For these elements, two key parameters are read. The first is the **probability** associated with the belief, and the second is its **weight**.

According to the definition for the KOOLA uncertainty model, the **probability** parameter in a production **rule** is an indication of the knowledge engineer's certainty. He or she should use rules to express the certainty of the consequential belief occurring, given that the antecedent fact is true. When calculating a new value for the *accumulated probability* of a fact, the inference engine averages out the probabilities of the supporting rule/fact pairs.

The well-known method to find the *average* of a set of **n** numbers involves adding them together, and dividing their sum by **n**. We define a *weighted-average* in the same way, however, each number is multiplied by a *weight factor* before they are added, and then the sum is divided by a scaling factor.

A KOOLA inference engine calculates the *accumulated probability* of a belief, by employing this *weighted-average* scheme. This provides a knowledge engineer with a primitive for expressing the importance of the fact-belief relationship of one rule, with respect to other rules. At the same time, this system permits the inference engine to calculate a new belief in the presence of missing facts[1].

To see how this method works, we will examine the contribution of one successful rule/fact pair to the *accumulated probability* (Ac) of a belief. Basically, the contribution equals the product of the *a priori* probability (Pr) (defined in the rule) multiplied by a

---

[1] This idea will be elaborated on in the section on fault tolerence (Sec 3.4.4).

weight factor (Wf). Where the weight factor scales the contribution, based on the relative importance of the rule with respect to all the others used in the calculation.

The contribution of the j^{th} successful rule to the l^{th} belief's *accumulative-probability* (Ac) is:

$$Ac_{lj} = Wf_{lj}\ Pr_{lj} \tag{3.1}$$

The weight factor of one rule depends on the rest of the rules used to calculate the *accumulative probability* (all elements in the *effective rule-set*). The *weight factor* is an indication of the relative importance of the information gained from the rule, with respect to the rest of the rules. The *weight factor* for a rule is formed by the quotient of its weight with the sum of all other weights in the *effective rule-set*. Where the weight is initially defined in the rule.

If there are n total rules used to calculate the Ac, then the weight factor for the l^{th} belief used in the j^{th} assignment is:

$$Wf_{lj} = \frac{W_{lj}}{\sum_{i=1}^{n} W_{li}} \tag{3.2}$$

To find the final *accumulated probability* of the l^{th} belief, the inference engine would add the contributions of each rule in the *effective rule set*, in the following manner:

By substituting in equation 3.2, we get:

$$Ac_l = \sum_{j=1}^{n} Wf_{lj}\, Pr_{lj} \tag{3.3}$$

$$Ac_l = \frac{1}{\sum_{i=1}^{n} W_{li}} \sum_{j=1}^{n} W_{lj} Pr_{lj} \tag{3.4}$$

Equation 3.4 defines how the *accumulated probability* of the l^{th} *primary* belief is calculated from a set of rules, and facts. In the next section we will see how this scheme is extended to deal with secondary beliefs.

3.6. Belief-based Algorithm for Inference

The *belief-based algorithm* defines a way for an inference engine to infer information about a **secondary belief**. Secondary beliefs were defined as beliefs that are inferred from other beliefs (secondary or primary), but not from facts. Like all definitions for inferring information, the *belief-based algorithm* depends on the knowledge stored in KOOLA **rules**.

3.6.1. Effective Rule Set for Secodary Beliefs

In solving a secondary belief, an inference engine employs the information stored in the belief's *effective rule-set* (similar to that in section 3-3). As with *fact-based* rules, the *effective rule-set* is a subset of the belief's *rule-set*. Also, as with *fact-based* **rules**, a belief's *rule-set* is created by the KOOLA compiler, based on production rules. The main difference between the two, is that an element in a secondary belief's *rule-set* can only become a member of its ***effective rule-set*** if at least one of the supporting beliefs in the antecedent of the rule is ***well defined.***

For a belief to be *well defined,* its *effective rule-set* must be a non-empty set. This means that if it is a primary belief, at least one fact must be known that supports it. For secondary beliefs, at least one supporting belief must be *well defined.* For example, the system could base a secondary belief on a single belief that itself, was only based on a subset of supporting facts. Since this type of information is only known during run-time, an inference engine decides dynamically if a belief is *well defined.* Next, we will examine how to calculate a secondary belief's *accumulated probability,* using its *effective rule-set.*

3.6.2. Calculating a New Belief from Supporting Beliefs

The weighted-average scheme used to calculate the *accumulative probability* of a primary or fact-based belief, must be modified to deal with the calculation of belief-based (or secondary) beliefs. Equation 3.1 illustrated the amount a successful rule could contribute to the *accumulative probability* of a given belief. It was formed by the multiplication of a probability with a weight factor. A third term, however, was not shown in this equation--a Boolean variable to account for the logical outcome of the sampled facts and compared to the antecedent of the rule, to establish if the rule *fires*[1].

---

[1] A rule is said to fire if the boolean equation in its "if" part (antecedent) is calculated by the inference engine, and found true.

Since we only consider rules that were already found to be true, the addition of this term would be redundant.

Consideration of this third term cannot be omitted with the algorithm for calculating secondary beliefs. The production rules that express the probability of a new belief, given a supporting belief, are interpreted by the uncertainty model as saying: "Given that the supporting belief is certain, there exists a certain probability **Pr**, that the consequential belief is *true*."

When new secondary beliefs are formed, however, the supporting beliefs are not 100% *true*. The truth of any belief is defined by its *accumulative probability*. Thus, when a belief is used to infer another belief, the third term (previously the boolean variable) becomes the *accumulative probability*, and is multiplied by the *a priori* probability of the rule. We call this third term the supporting belief (**Sb**).

Another way to view the equation which defines the contribution of a belief-based rule to the *accumulative probability* of a new belief is in terms of classical probability theory for dealing with independent events, *i.e.*, the joint probability of two independent events equals the product of their individual probabilities. Following this theory, we say that a rule indicates that the probability of the *new belief* equals the product of the probability of the *supporting belief* and that of the rule.

Therefore, the factor contributed by the l^{th} successful belief-rule pair to the *accumulated probability* of the m^{th} belief (Ac) is:

$$Ac_{ml} = Wf_{ml} \left\{ Pr_{ml} Sp_{ml} \right\} \qquad (3.5)$$

The weight factor (Wf) is calculated the same as that in equation 2:

$$Wf_{ml} = \frac{W_{ml}}{\sum_{i=1}^{n} W_{mi}} \qquad (3.6)$$

Adding all n contributions of supporting belief/rules pairs yields the following:

$$Ac_m = \sum_{l=1}^{N} Wf_{ml} \cdot \left\{ Pr_{ml} Sp_{ml} \right\} \qquad (3.7)$$

Substituting equation 3-8 results is the following:

$$Ac_m = \frac{1}{\sum_{i=1}^{n} W_{mi}} \sum_{l=1}^{N} W_{ml} Pr_{ml} Sp_{ml} \qquad (3.8)$$

3.7. Fault Tolerance

Previously, we stated that the environment in which the ISDN workstation would operate in affects the design of the default KOOLA inference mechanism. The main environmental consideration is the ISDN network, and the workstation's reliance on it to deliver information. The network may not always be able to deliver the required *facts* due to problems in the network, or with third party information services (*e.g.*, data-bases).

With its inference mechanism, a KOOLA knowledge-based system can cope with a partially incomplete set of facts (information), while maintaining the integrity of its process. This comes from the distinction we made between an initial *rule set*, and its final *effective rule set*. As we stated, as long as there is sufficient information to satisfy at least one rule (*i.e.*, an non-empty effective rule set) an inference engine may infer a *belief* from the *facts*, and thus continue processing inspite of the missing information.

This ability is enhanced in KOOLA with the use of *rule weighting*. A knowledge-based system may contain a number of backup rules with low weighting that normally do not have much effect on the outcome. If all of the main rules fail, however, the inference engine falls back on the backup rules to continue processing. In effect, a form of fault tolerance through knowledge redundancy.

CHAPTER 4

THE KOOLA PRODUCTION SYSTEM:

LANGUAGE ELEMENTS

In this chapter we present the language elements of the KOOLA production system, and examine the development of a KOOLA knowledge-based application.

The basic concepts of the KOOLA production system were presented in Chapter 3. In that chapter we investigated a model to organise different knowledge representation schemes. From this we demonstrated that a hierarchy of meta-level knowledge could be used in KOOLA. With a review of uncertainty in symbolic processing, we were able to present how the KOOLA inference algorithm deals with information uncertainties. With this background, we will proceed with the language elements that make up the KOOLA production system.

We proceed as follows in this chapter. In Section 4.1, we present the KOOLA language elements that are used to represent procedural knowledge. In Section 4.2, we present the KOOLA **rule**. In Section 4.3 we present the KOOLA **goal**. In the last section, we detail the steps used to develop a KOOLA knowledge-based application.

4.1. KOOLA Support Primitives

Support primitives are the KOOLA language elements which are used to represent procedural knowledge and help in the definition of **rules** and **goals**. They are not used to represent declarative knowledge. Table 1 contains the list of KOOLA language elements, and includes the four support primitives. We will examine the characteristics shared by these primitives, especially the relationship that exists between the first three.

TABLE I. KOOLA LANGUAGE ELEMENTS.

PRIMITIVE	TYPE	DESCRIPTION
External Request	BELIEF	Defines how to get data from the ISDN network or from any other external device.
Internal Request	BELIEF	Defines how to get data from the end-user by asking her or him a question.
Primary & Secondary Belief	BELIEF	Specifies the names of the intermediate and the final conclusions that the inference engine will make when solving a problem.
External Action	ACTION	Defines how the system can do something with the ISDN network or another device.
Rule[1]	KNOWLEDGE	A heuristic equation that relates facts and beliefs to other secondary beliefs.
Goal[1]	KNOWLEDGE	Defines which beliefs the system will work on, also indicates when to undertake a specific External Actions.

4.1.1. General Belief Primitives

The first three elements are defined as being of type BELIEF. We make this distinction because they represent objects maintained by the *Belief Manager* of the proposed intelligent workstation architecture examined in Chapter 2. **Requests** primitives fall under this classification, because they define how to acquire a *fact*. In KOOLA, a

---

[1] The **rule** and the **goal** are not support primitives, but are included in this table for completeness.

fact is classified as a form of belief that is always certain. For example if the end-user states that there is someone at the door, then that fact (someone is at the door) is considered 100% certain, and the complimentary fact (someone is not at the door) is considered 0% certain. In general, any defined fact in KOOLA is either true (100% certain) or false (0% certain).

The **primary** and **secondary belief** primitives define a conclusion that may be reached by processing facts. In KOOLA, **rules** are used to define how these conclusions are *processed* (or *inferred*) from facts. Unlike facts, **primary and secondary beliefs** are not always certain. A **primary belief** is inferred from a fact, and a **secondary belief** is inferred from a **primary belief** or another **secondary belief**. We will show how their inter-relationship leads to the hierarchical organisation shown in Figure 6.

We say that when one type of information is inferred from another, then the former type of information is more abstract[1] than the latter. A fact is less abstract than its inferred belief because it is directly measurable. A **secondary belief** inferred from a **primary belief** is based less on physical facts--therefore more abstract. The organisation depicted in Figure 6 results from using this definition, and ordering the belief primitives by the degree of abstraction associated with the information they represent. The arrows in this drawing indicate the direction of inference. The ideas presented in this model help to explain the placement of these primitives in **rules** and **goals**.

---

[1] The level of a belief's abstraction is an indication of how far its *meaning* is from an easy measurable fact.

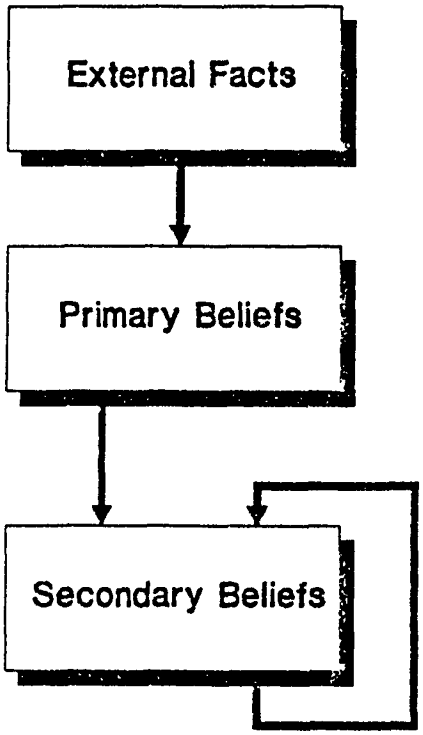


Figure 6. Hierarchical organisation of belief types in terms of their abstraction. The arrows indicate the direction of inference.

4.1.2. Support Primitive Definitions and the Working Set Domain

A KOOLA programmer *defines* support primitives prior to employing them in a rule. Like with the C language, *defining* a type creates an instance of the type and an association between the name given and the instance of the type [23]. Also, initialisation values are set during definition. Unless a primitive is defined as *variable* (which will be explained later) its instance is passed down to the knowledge-based system as a static primitive entity--created once and never destroyed.

Many of the KOOLA constructs are used in the definition of rules. For example, an enquiry is used in the antecedent of a production rule. In KOOLA, we stipulate that before a programmer can use a construct in a rule, it must be already defined. Also, we define that the set of all defined constructs, which may be used in a rule, as the *working set domain.* The reader should note that all elements in the working set domain are mutually exclusive.

The concept of a working set domain forces a developer to keep rules consistent. All operands of a new rule must be in the working set domain of its object. This construct also helps knowledge engineers use a bottom up approach to knowledge-based design, in association with the KOOLA production system. The developer can first state all beliefs, actions and information the system will use, then add the rules that use the information and effect the beliefs.

The concept of a *working set domain* for every support primitive object also simplifies the translation of KOOLA source code into the primitive knowledge format which a run-time kernel can use. The translation simplification is a result of the compiler not needing to extract this information from the rule in order to compile the rules.

4.1.3. The Variable Construct

The inclusion of the variable construct into the KOOLA language was to facilitate the development of more advanced applications. In particular, this permits applications to evaluate a (possibly unbounded) number of external objects using the same set of KOOLA primitives to classify each object. Typically, such applications choose a single "best" object, or all objects that meet a set threshold. An example will illustrate the usefulness of the construct.

Consider a knowledge-based system which evaluates stocks in a stock-exchange. Such a system would evaluate all candidate stocks, and then present the end-user with a list of the top 10. Since most stocks are evaluated in a similar manner, the system could use an identical set of rules to evaluate each candidate stock. If it did, however, the conclusions it reached (beliefs), for each candidate, would be stored in the same **belief** variables--resulting in the beliefs of one candidate over-writing the beliefs of another.

An unsatisfactory approach would be to write a new set of rules for each candidate stock evaluated. Each set of rules would store beliefs for its candidate stock in a separate set of belief variables. Once the system had evaluated all of the candidate, it would presents the final beliefs, for each candidate, to the end-user. This approach is impractical due to the excessive programming effort required to write a new set of rules for each candidate.

The approach KOOLA supports makes use of the *variable construct*. Using this, the knowledge-based system would dynamically allocate a new set of *variable beliefs* for each stock evaluated. Each set of variable beliefs would hold the conclusions for the candidate it evaluated. The common set of rules used to evaluate each candidate would belong to the application's variable rule set.

For an example of how this construct can be put into practice consider the following abstract rule[1]:

---

[1] An *Abstract Rule* is analogous to an *abstract data type*. It is a way of describing the meaning of a rules in English.

FOR a given stock

IF its value has been increasing in the last month,

AND its value has been increasing in the last three month,

AND its value has been increasing in the last six month,

AND its value has been increasing in the last year,

THEN conclude favourably about the given stock

The **variable type** in this rule is "given stock". A knowledge-based application with this rule would also have a number of **beliefs** with the same variable type. It would have beliefs like *given stock-buy* or *given stock-sell.* Every time a new candidate was identified, the Belief Manager would make a copy of all these beliefs, and associate them with the new candidate. For example, if the new candidate was IBM, then IBM would be the "given stock".

If the variable construct was not used in such an application, the rule base would require a similar rule for each possible stock that it *might* evaluate. Thus, in this type of application, the variable construct decreases the number of required rules, and allows it to solve an indefinite number of stocks.

4.1.4. Internal Enquiry

Internal enquiries provide a means by which a programmer can specify how a question should be asked of the end-user. By asking questions, a knowledge-based system can gather the *facts* it needs from the user. A knowledge-based system would gather facts because they are used to solve problems by inferring new beliefs.

Like all KOOLA fact-constructs, an **internal enquiry** has a shelf life. In Chapter 3 we saw how the *Belief Manager* uses this information to maintain all facts and beliefs in a run-time kernel up-to-date. A shelf life of 99999 indicates that the fact does not expire.

In the KOOLA production system there are two ways of defining an **internal enquiries**. A separate template is used for each definition. The way that an **internal enquiry** is defined depends on the nature of the question asked.

An **internal enquiry** may define a question that requires a **numerical** answer. If this is the case, the programmer selects the *numeric-based* template to define the enquiry. For example, if the knowledge-based system asks the end-user a questions such as: "*How old are you (in years)*", then the answer would be a number from 1 to 100. To help prevent the user from entering a totally invalid answer, this type of enquiry sets a range of valid replies. For age, the range could be from 1 to 100 years old.

The following example shows a *numeric-based* **internal enquiry** that asks the end-user to estimate the age of an unknown person. The **variable class** in this object is *unknown_person*.

FOR: unknown_person

INTERNAL ENQUIRY: age

ASK: Estimate the age of the person we are trying to identify

LOWER BOUND: 1

UPPER BOUND: 100

SHELF LIFE: 99999

In the section on rules (Sec 4.2), we will see that the antecedent ("if ..." part) of a KOOLA **rule** contains a Boolean expression that defines how a fact should be tested. For example, the Boolean expression *unknown_person:age = 1* is only true if the age of the person is one years old. If a range is more appropriate, a Boolean expression can be written as F<=2 *AND* F>=8, given that F is any fact. In this case as long as F is between two and eight, the result is true.

The second kind of **internal enquiry** is for text-based answers. With this type of question, the user is expected to select her or his answer from a fixed-set of allowable responses. As in the other enquiry, the template for defining a text-based **internal enquiry** contains the field for the questions. In addition, it also has eight blank fields for entering the allowable responses.

The following **internal enquiry** asks the user to define the size of a person:

FOR: unknown_person

INTERNAL ENQUIRY: size

ASK: What size is the person?

CHOICES: (tiny) (small) (medium) (large) (huge) (enormous) () ()

SHELF LIFE: 9999

KOOLA treats the allowable responses entered as an ordered set of symbolic atoms[1]. This means that the fact defined by a text-based **internal enquiry** takes on the characteristics of an integer when processed by a run-time kernel. For example, if the user selected the fourth response (large) when asked the previous question, then the fact associated with *unknown_person:size* would be set to four. This is an important feature, because it provides additional flexibility in defining a boolean equation; and, we as we shall see, overcomes an information uncertainty problem.

One of the problems discussed, in Section 3.3.2, about uncertainty was the presence of symbolic uncertainty in facts. We saw that this form of uncertainty shows up when English (or any other natural language) is used to qualitatively describe characteristics of something. This is usually the case when an end-user is asked a text-based question like the previous example. A person may be considered *huge* by one observer, but only *large* by another. The KOOLA approach to this problem makes use of the ordering of the symbolic atoms in an **internal enquiry**.

---

[1] We call each response a *symbolic atom* since it is a character string with a real-world symbolic meaning. Symbolic atoms can also be characterised by their ability to be ordered by their real-world meaning.

Let us now consider the following boolean equations, and contemplate the conditions required to make each one true:

IF *unknown_person:size* < 4,

IF *unknown_person:size* = 4,

IF *unknown_person:size* >= 2 .AND. *unknown_person:size* <= 4

The first expression is true if the person's size is identified as being less than large. The second equation is true only if the person is considered large. The last expression is true if the person is considered anything from small to large. If **internal enquiries** are used in this manner, *i.e.*, defining a range of acceptable answers, the problem associated with symbolic uncertainty, using a natural language, is reduced, since this manner allows for a greater range in correct answers.

4.1.5. External Enquiry

A knowledge-based system developed with KOOLA can use the **external enquiry** to gather *facts* that do not come from the end-user. A construct that implements this capability is not common in most production systems. The KOOLA production system however, must provide external fact gathering in order for it to support the development of autonomous applications. The reason that such applications need the **external enquiry** is that autonomous applications should gather data on their own, without a human operator present, .

For the experimental ISDN workstation, the main use of the **external enquiry** is to gather facts from the ISDN network. For example, the **external enquiry** can be used to obtain status information about the ISDN connection. It can also be used to make data base queries to third-party data base services through the network. A secondary use of the enquiry is to obtain control information from any other device connected to the workstation (*e.g.*, the centralised controller[1]).

Syntactically, the **external enquiry** is much like the previous numeric-based internal enquiry. The only difference is the **TOKEN** field. For the experimental ISDN workstation, this field is used to specify an operating system service. The type of service defined in the **TOKEN** field should correspond to the information required by an application.

One of the services supported by the operating system of the experimental ISDN workstation returns the status of an ISDN channel. The following **external enquiry** would defines how to get the status of a voice channel.

FOR: isdn_voice
EXTERNAL ENQUIRY: channel_state
TOKEN: 12323
SHELF LIFE: 1

---

[1] The Centralised Controller is a hardware device connected to the experimental ISDN workstation. It can measure and report temperature. It can also control electrical appliances plugged into it. Appendix 3 shows how it is connected to the workstation platform.

The fact obtained by this enquiry could be used in a rule to determine if a voice channel is free so that a call could be made.

4.1.6. Belief

The **belief** is the simplest object to define. It consists of a variable class and a name. There are two sub-classes of beliefs called the *primary* and *secondary* belief. In Figure 6, we saw that the difference between the two classes of beliefs was that a **primary belief** is inferred from fact(s); and that a **Secondary belief** is inferred from other belief(s).

The KOOLA language must distinguish between the two sub-classes of beliefs. The factor that decides the sub-class of a belief is how it is used in a rule. The convention used to do this is:

1- *If a belief is used by any rule that contains a fact-based antecedent, then that belief is a primary belief.*

2- *If a belief is not a primary belief, then it is a secondary belief.*

A **Secondary belief** (*i.e.*, a belief that is inferred from another belief) may be designated as a *goal belief*. This designation indicates that the **Secondary belief** is important, and is used to regulate the application it belongs to. We will examine how it does this in the next section on rules. The most important point the reader should note is that any **Secondary belief** can be designated as a *goal belief*.

4.1.7. External Action

The **external action** is the support construct that gives KOOLA applications the ability to control physical devices. For the ISDN workstation, the main use of this language element is to control the ISDN network interface. Through an **external action,** a knowledge-based system running on the workstation can make an ISDN data call.

The syntax of the **external action** follows the general format of all support constructs. When used for the ISDN workstation, the **TOKEN** field identifies an operating system service that can be used by a knowledge-based application. The following **external action** could be used to make an ISDN data connection:

FOR: isdn_data
EXTERNAL ACTION: make connection
TOKEN: 456

4.2. Rules

This section contains the language description for KOOLA **rules** and explains how they are used by the KOOLA production system to codify knowledge.

4.2.1. Production Rule Requirements

To develop a knowledge-based system, a human expert is required. This person possesses much knowledge about the problem that the knowledge-based system will work on. The relevant problem solving knowledge held by this person is frequently called *domain specific knowledge*, to distinguish it from the general programming knowledge needed to develop routines like an inference engine. In general, the creation of a knowledge-based application is the process of transferring the subject matter expert's, domain specific knowledge, into a program.

The task of the KOOLA production system is to facilitate this transfer of knowledge. The most important consideration is the method used to represent the domain specific knowledge in the computer (*i.e.*, the *knowledge representation method*). The method must be easy for the user to use, expressive enough that the user can embed detail knowledge about the problem domain, and it must be a format that can be manipulated by a computer.

What is the nature of the domain specific knowledge that must be stored in the *knowledge base* of a knowledge-based system? It is sometimes called *heuristic* (*a.k.a.*, rule of thumb) knowledge to emphasise its inexact nature [1]. From our discussion on uncertainty (Section 3.3), we saw that domain specific knowledge contains many types of uncertainty. Therefore, the *data* in a knowledge base can be characterised as heuristic knowledge about uncertain information. Any knowledge representation method that uses this type of knowledge must effectively take these characteristics into account.

4.2.2. Production Rule Format

The KOOLA knowledge representation method that responds to the needs of codifying domain specific knowledge is the **production rule** format.

The **production rule** comprises two components. The first component consists of an "IF" followed by a boolean equation that checks a fact. Since this part of the rule is processed first, it is called the *antecedent.* The second component consists of a "THEN" followed by a belief-based assignment. This assignment is an indication of how true a certain belief is, given that the antecedent is true. This part is called the rule's *consequence*, since it is processed as a consequence of evaluating the antecedent. The rule format we have just seen is referred to as the *if-then* format [1,14].

To summarise, a KOOLA rule expresses the probabilistic association between a sampled external event (fact) and the conclusion one could draw from these observations (beliefs). If the antecedent of a KOOLA rule contains a belief instead of a fact, the rule expresses a probabilistic association between one conclusion and a more abstract conclusion. For this reason, we define a KOOLA rule as: *A heuristic equation that uses uncertainty and operates on beliefs or facts to formulate more abstract beliefs.*

4.2.3. Backward Chaining

Rules are processed in a knowledge-based system so that the certainty of a belief can be established. In general, they are processed in an inference engine by matching their antecedents to facts; if they match, then the actions outlined in their *consequence* are performed [1,14]. In KOOLA, that action involves updating the certainty of a belief, using one of the algorithms given in Chapter 3. The action of checking an antecedent, and finding it to be true; is called *firing* a rule A rule is said to have *fired* if this action occurs.

The key consideration in the design of an inference engine is which rule to process next. Most knowledge-based system, and all expert systems, have huge knowledge bases [14]. The number of rules is usually so large that they cannot all be tested to solve a given problem. Also, different rules can make different conclusions about the same belief. This leads to the need for an effective scheme to decide the next rule to test--this is called the *scheduling algorithm* [7].

The KOOLA scheduling algorithm for rules makes use of a special set of beliefs we call *goal beliefs*. The algorithm works by designating one belief as the goal belief, and only testing the rules that affect the *goal belief*. If one of the rules that affects the *goal belief* also has a belief in its antecedent, then all the rules that affect the new belief are also tested. Eventually, the inference engine is only left with fact-based rules, which can be fired directly.

This algorithm keeps the inference engine working towards solving one belief, the goal belief. In other words, it is *goal oriented*. For this reason, the method used by KOOLA to schedule rules is called the *goal oriented method* [14]. Since the establishment of goal beliefs dictates which rules the inference engine will work on, a KOOLA programmer may use this mechanism to control the flow of an application. An explanation of how a programmer can exploit this control primitive is made in the next section on **goals**.

The mechanism we just described can be shown with a relational diagram. This diagram highlights the backward movement from a goal belief through supporting beliefs, down to facts. This movement is called *chaining* since one belief connects to another

From this point of view, the mechanism can be called *backward chaining* [14]. A diagram which shows an instance of chaining is called an *inference chain* [14].

Figure 7 presents a typical inference chain for KOOLA rules. In this diagram, one arrow shows the direction the inference engine searches for a fact-based rule to start with. The other arrow shows the direction beliefs are inferred from the facts. The inference chain also highlights the relationships between primary, secondary, and goal beliefs.

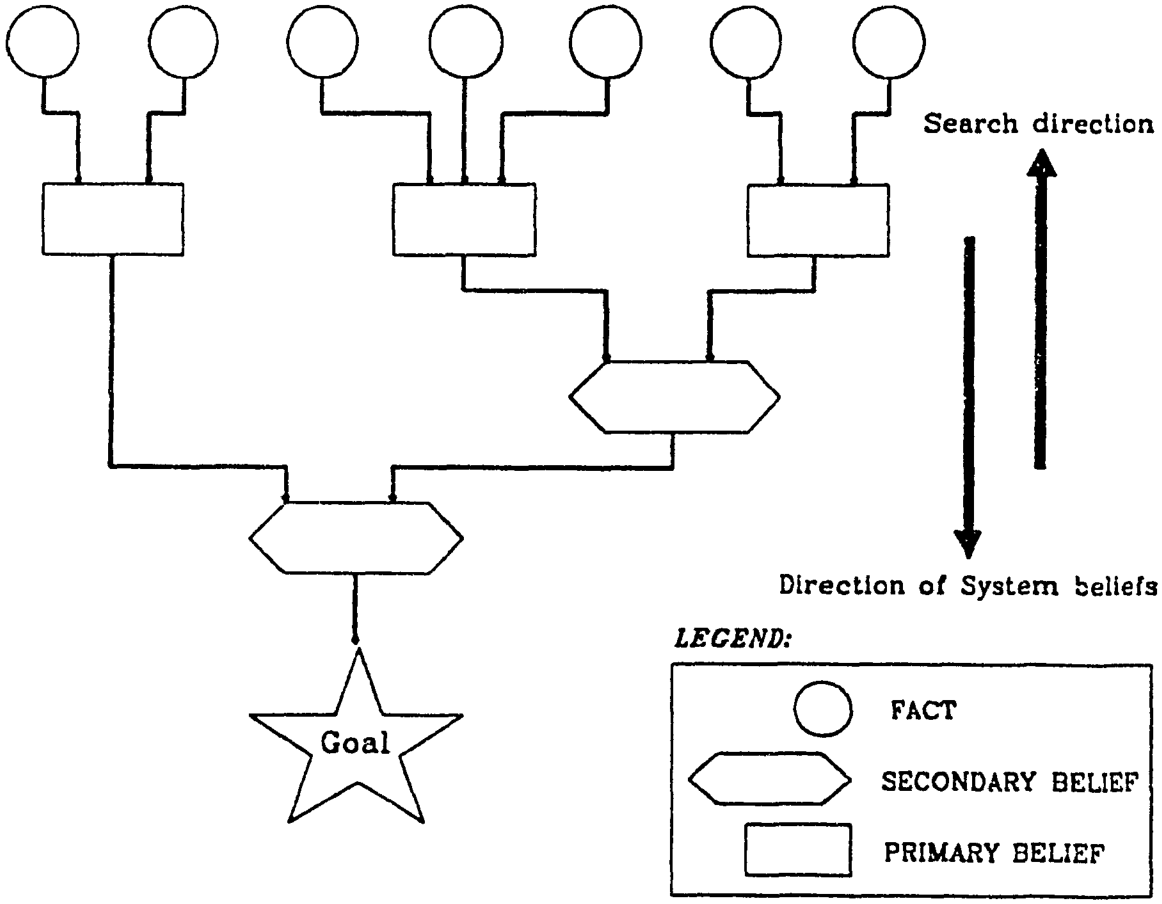


Figure 7. Generalised inference chain

4.2.4. Syntax

We use the *if-then* format for KOOLA rules. Following this format, a rule is divided into an antecedent that describes what the rule should test, and a consequence, that describes which belief should be affected by *firing* the rule (*i.e.* finding its antecedent true). We define two kinds of antecedents, which leads to two sub-classes of rules, fact based and belief-based. We will examine the syntax of both sub-classes of rules.

4.2.4.1. Fact-Based Rule Syntax

As discussed in section 4.1.4, a KOOLA *fact* is acquired and represented as numerical data that contains the value of something measured in the internal or external environment of a knowledge-based system. Since a fact involves a direct measurement, it is an exact value with no uncertainty. For example, a run-time kernel could establish that fact Q equals 23.

The antecedent of a fact-based rule comprises Boolean expressions with facts. The Boolean expression defines how to test a fact. It does this with simple comparative operators that relate facts to numbers. For example an antecedent with the equation fact $Q < 23$ would be false, since Q is not less than 23. An antecedent may group multiple Boolean expressions together by ANDing their outcomes.

The next two fact-based **rules** are part of a knowledge-based system that identifies people. They both belong to the variable class *unknown person*. In the system, they help to strengthen or weaken the belief that the unknown person is *Jim*, by evaluating his face

```
FOR unknown person:

        IF: eye colour == blue

        THEN: jim's face 100 weighted 95

        ELSE: jim's face 5  weighted 95

FOR unknown person:

        IF:  hair colour == brown

        AND: hair length > shoulder length

        THEN: jim's face 90 weighted 90

        ELSE: jim's face 10 weighted 40
```

Both rules define facts that can be tested to establish the primary belief: *jim's face*. The first rule is true if the eye colour of the unknown person is blue. The second rule is only true if the hair is brown **and** longer than shoulder-length.

The primary belief defined in the rules is *jim's face*. This belief is in both the "THEN" and "ELSE" parts of their consequence. Which part would be used depends on the facts. An inference engine would use the assignment in the "THEN" part if the rule is true, otherwise, use the "ELSE" part.

The consequence of a belief includes parameters that define a *probability* and a *weight*. For the first rule, the parameters used to calculate *jim's face* are 100%, weighted 95/100; if the rule is true, and 5% weighted 95/100, otherwise.

4.2.4.2. Belief-Based Rule Syntax

The second sub-class of KOOLA's if-then rule syntax is the belief-based rule syntax. A belief-based rule has beliefs in both its antecedent and its consequence. These rules express a probabilistic relationship between the two beliefs. The exact relationship is defined by the inference algorithm in section 3.4.

Consider the next two rules from a human identification knowledge-based system.

```
FOR: unknown person
        IF jim's face
        THEN jim 100 weighted 90
FOR: unknown person
        IF jim's body
        THEN jim 100 weighted 60
```

These rules infer the belief *jim*. Since this belief is inferred from another (more definite) belief, it is a secondary belief. In this example, both rules have primary beliefs in their antecedents, but they could also have had secondary beliefs. In general, for a belief-based rule, the antecedent may be any type of belief, but the consequence is always a secondary belief.

4.3. Goals

This section contains the language description for KOOLA goals and explains how they are used by the KOOLA production system to codify control knowledge.

In the previous section, we saw how a knowledge-based system could use KOOLA rules to calculate the certainty of a *goal belief* (expressed as a probability). We also introduced the notion that a KOOLA construct for establishing the order in which *goal beliefs* are calculated, would permit a programmer to dictate the flow of an application. We will now introduce that construct.

The **goal** permits knowledge engineers to control an application's flow. It provides two control oriented primitives. Of these, establishing *goal beliefs* is the most important primitive. The second primitive builds on the first. It permits programmers to express when his or her knowledge-based application should initiate an external action, based on the certainty of a *goal belief.* Together these two primitives provide the ability to codify control knowledge in a production system.

4.3.1. The format of a GOAL

The **goal** is a special class of production rule that uses the "if-then" format. We established that a production rule is an antecedent-consequence pair. The antecedent of a **goal** contains a *goal belief,* and the consequence contains an **external action** identifier. The consequence defines what actions a system will take after evaluating the antecedent.

A run-time kernel starts executing a KOOLA application by processing the application's initial **goal**. The first step in processing a **goal** involves checking its antecedent. The antecedent contains a *goal belief* in a Boolean expression. Before evaluating the Boolean expression, it must ascertain the certainty of its *goal belief.* KOOLA **rules** are used to establish this certainty. Once a *goal belief* is known, its value is substituted back into the Boolean expression; and the antecedent is either found true or false.

The second part of goal processing involves performing the appropriate action defined in the consequence. To facilitate this, the consequence is divided into a "THEN" consequence and an "ELSE" consequence. The "THEN" consequence is performed if the antecedent is found to be true, and the "ELSE" if found false. The action performed is specified by the **external action** identifier in the appropriate consequence.

The format of the goal leads to a useful interaction between beliefs and actions in a knowledge-based system. Beliefs are the result of employing the heuristics embodied by KOOLA rules, and are characterised by levels of certainty. Actions, on the other hand, represent procedural knowledge, that can either be done or not done (*i.e.*, no uncertainty). Thus, goals act as an interface between the probabilistic domain of heuristic knowledge, and the certain domain of procedural knowledge.

4.3.2. Goal Inference Strategy

The consequence of a goal identifies more than just what action the system should take. It also contains a field that identifies the next goal that will be processed in the application. This field controls the *chaining* of goals, since the next goal a run-time kernel will process depends on which consequence it selects. As we saw, this selection depends on the goal's antecedent.

The manner just prescribed for chaining goals is framed by the decision to let goals explicitly define the next goal in its inference chain. A KOOLA application always starts at its *initial goal*. After establishing the *goal belief*, the run-time kernel either uses the "THEN" or "ELSE" consequence to select the next goal to chain to. Likewise, which ever **goal** it chains to; the new goal is processed in the same way. The inference ends (and so does the application) once it encounters a consequence with a terminating clause.

This type of chaining involves a run-time kernel moving forward from one **goal** to the next. The kernel never needs to do the type of backward chaining required to solve KOOLA **rules**. The applications start at a well-defined *initial goal*, then moves forward through intermediate goals, until reaching a terminating goal. Since this type of inference is always moving forward, it is called the *forward chaining* method [1,14,7], and a KOOLA **goal** can be classified as a *forward chaining* rule [1,14,7].

A diagram showing all possible paths an inference engine could take from an initial rule is an *inference network* [14]. Figure 8 is a KOOLA inference network for chaining **goals**. In this inference network, we demarcate one path with a heavier line. Since this path shows only one instance of a path that a system could take, it is called an *inference chain* [14].

A common characteristic of all forward chaining inference methods is the unpredictability of their actual inference chains. Even though they start at a known location (the *initial goal* for example), their final location is determined dynamically while the system runs. It is the *data* used in the antecedents that dictates the path taken, and the end-point in the inference chain. For this reason, forward-chaining inference techniques are also called *data-driven techniques* [14].

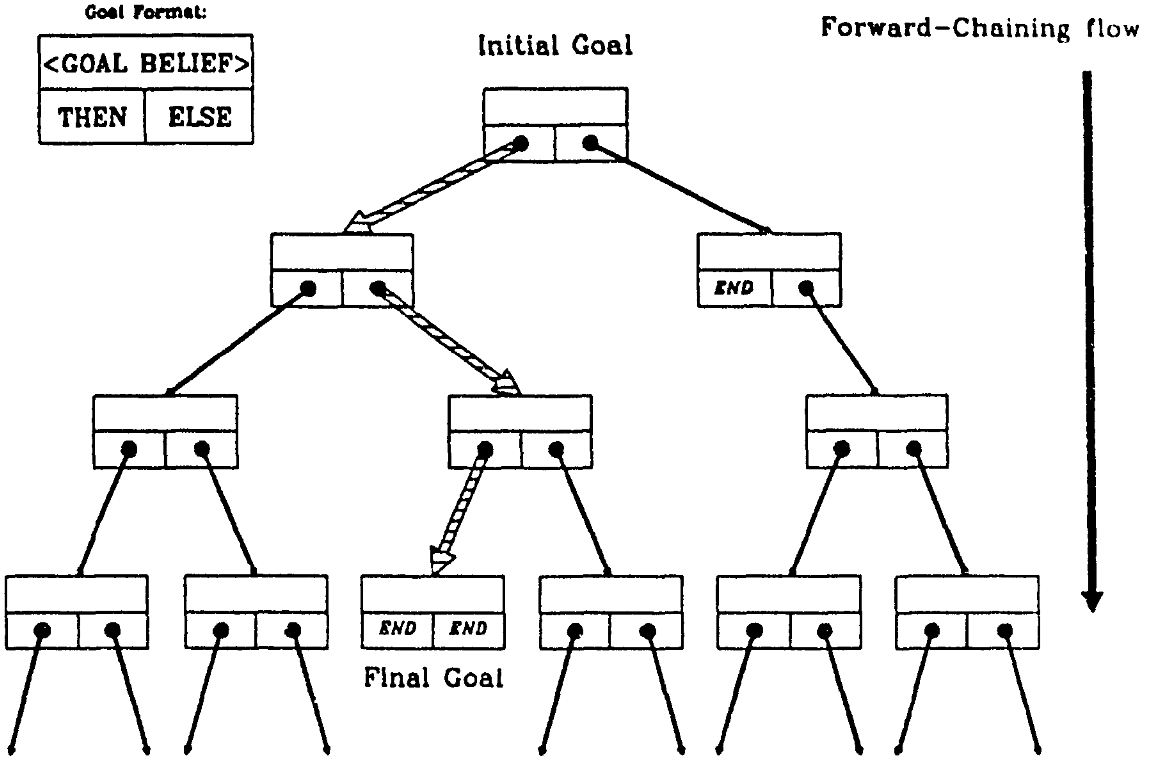


Figure 8. An inference network of goals showing the forward chaining control used to solve for one inference chain.

4.3.3. Real-time Control.

By letting programmers explicitly define the next goal in an inference, KOOLA gives programmers more control over the chaining of data-driven goals, than with backward-chaining rules. For similar reasons, other developers have found that the data driven technique is suitable to meet the needs of a real-time event-driven system [5].

Texas Instruments has developed an expert system shell for real-time process control called PICON [26]. One of the inference strategies that its inference engine supports, is similar to KOOLA's forward-chaining strategy for goals. In particular, it permits one rule to explicitly call another rule in its consequence.

By establishing *goal beliefs*, **goals** control the use of **rules** in a run-time kernel. In a real-time event driven control system, this ensures that *goal beliefs* are developed so that the importance of the most recent events is taken into consideration. Programmers can specify the order in which goal will be processed. Since this ordering affects when things are done in a run-time kernel, KOOLA can guarantee that the temporal order, specified for a real-time application, is followed.

4.3.4. Meta-Control

Previously we said that **goals** are a form of forward-chaining rules. We have also demonstrated that they control the use of heuristic-based rules. In effect, **goals** represent control knowledge on how to use heuristic knowledge. For this reason we can call the knowledge embodied in **goals**, *meta-knowledge* (a.k.a. knowledge about knowledge). We can also call **goals**, *meta-rules* [26].

This concept brings us back to the ideas expressed in Section 3.2. In that section, we introduced the theory of a knowledge processing model for symbolic processing applications that favours the organisation of knowledge into a hierarchy of meta-level knowledge.

The reader should see that the KOOLA **goal** and **rule** form such a hierarchy, with **goals** on top. Work on symbolic processing suggests that adding an extra level of meta-level knowledge to a system can improve system performance in lieu of more heuristics [11]. Thus, **goals** do not only improve the control a programmer has over an application, but also reduce the amount of required heuristic knowledge.

A meta-rule construct is applicable to expert systems in structuring the consultation phase of operations. The consultation phase involves the expert system extracting *facts* from the user, and inferring beliefs from the facts. Personal Consultant Plus, an expert system shell, uses meta-rules to ensure that the flow of questions asked of the user, during consultation, follows a logical progression [26] (*i.e.*, one subject at a time and using a logical progression from subject to subject).

A knowledge-based system designer, using KOOLA, can employ goals to influence the consultation phase (if user information is required) in an application. Setting a specific *goal belief*, forces the system to concentrate on **rules** pertaining to the belief, which keeps the question it asks focused on one subject. Because the programmer also controls the order in which *goal beliefs* are set, she or he can ensure that they generate a logical progression of question flow for the end-user.

4.3.5. Goal Syntax

The following is an example **goal**, showing how it would appear in the KOOLA production system:

```
GOAL: make call

FOR: ISDN voice channel

IF: free >= 95%

THEN DO: place call

THEN CHAIN: call in progress

ELSE DO:

ELSE CHAIN: TERMINATE
```

The name of the goal is *make call.* It works with the variable class *ISDN voice channel.* This goal asserts that if the belief *ISDN voice channel:free* has a probability greater then 95% the knowledge-based system should continue by placing the call. At the same time it asserts that if the belief is not true, then the system should do nothing and end the application.

4.4. A Koola Application

This section explains how to build a knowledge-based application using templates in the KOOLA programming environment. We illustrate the functionality of the programming environment by detailing the steps that were involved in developing a knowledge-based system in KOOLA.

The application we selected to demonstrate the KOOLA production system is called the Human Identification Knowledge-Based System (HIDS). HIDS is a very simple knowledge-based system (*i.e.* a problem in the toy domain), but it exercises all of

KOOLA's components. We decided on this simplified knowledge-based application to prevent the important details of an implementation form being eclipsed by the complexity of a large application. The source listing for HIDS can be found in appendix 1.

4.4.1. Problem Description

What should a young person do if someone is knocking on the front door of their house? It could be a family friend. It might be someone soliciting a product. It could also be a potential intruder or someone even worse. If the person is a friend, then they should be let in. Otherwise, the stranger must not be let in and the child's parents should be notified of the situation. But how can the child identify someone? HIDS is a proposed software solution to this problem.

HIDS will be a knowledge-based application that will run on the inference engine of the ISDN workstation. Its main task will be to help young people to identify any stranger that knocks on the door of his or her home. Its secondary task will be to sound an alarm if the stranger can not be identified. An alarm will involve sending a message through ISDN to a parent informing them of the situation. The program must tolerate error in its input data since the user may make a few errors when describing the unknown person, also it should be easy to use.

4.4.2. Starting With Goals

Since we prefer to use a top-down approach to software development, a logical starting point for programming HIDS would be to map out its high level flow control.

Fortunately KOOLA provides programmers with a meta-control primitive to manage the flow of processing in a target system. This meta-control primitive is the **goal** construct.

The mechanism by which **goals** are able to provide control is through the establishment of **goal beliefs**. By setting intermediate **goal beliefs**, that the inference engine will solve, **goals** indirectly controls the inference engine's processing. Also, by setting the threshold of certainty needed of a **goal belief** (*i.e.*, level of probability), before a system action is started; **goals** directly control all external actions. For example, one of the **goal** in HIDS decides when to send the help message on the ISDN.

The high-level flow we decided on for HIDS is mapped out in Figure 9. We were able to make this drawing as a direct consequence of the problem description. It shows that the first **goal belief** we wish to work on is whether the application even needs to be run. Obviously if no one is at the door, the ISDN workstation should not be asking the user about the unknown person there. This makes sense since this application could be triggered by the door bell (*i.e.*, if the door bell rings then there might be someone at the door). We decided on the rest of the blocks in Figure 9 using a similar general understanding of the problem domain, and some rules-of-thumb.

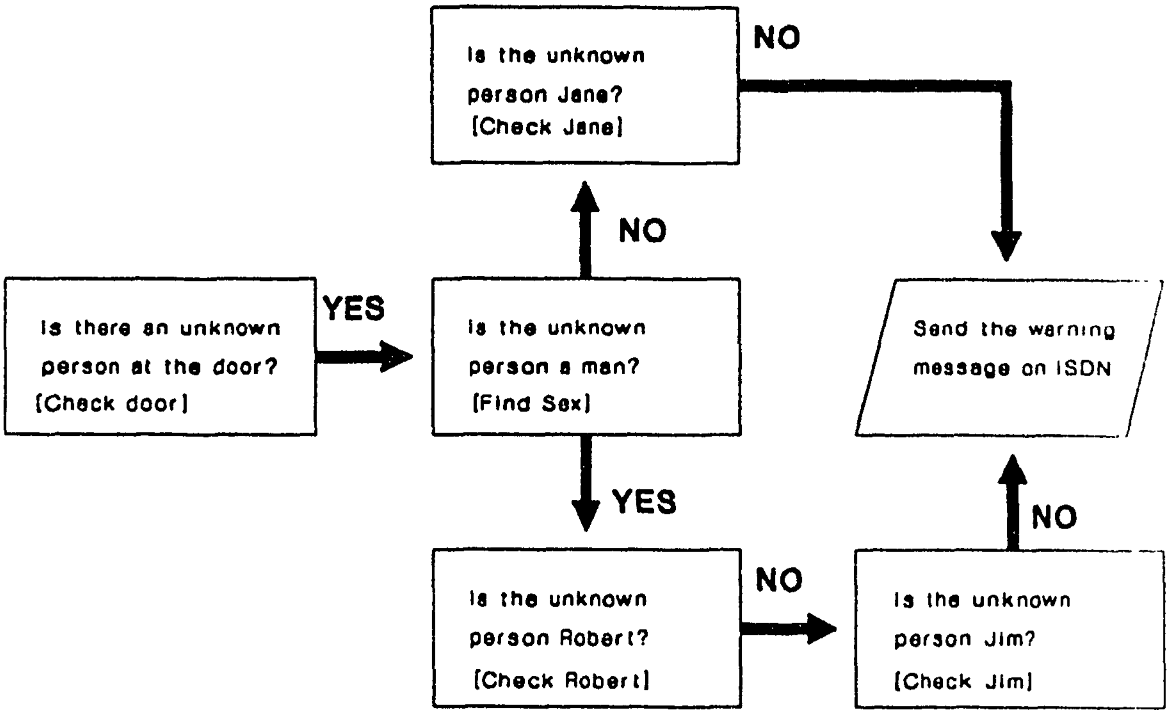
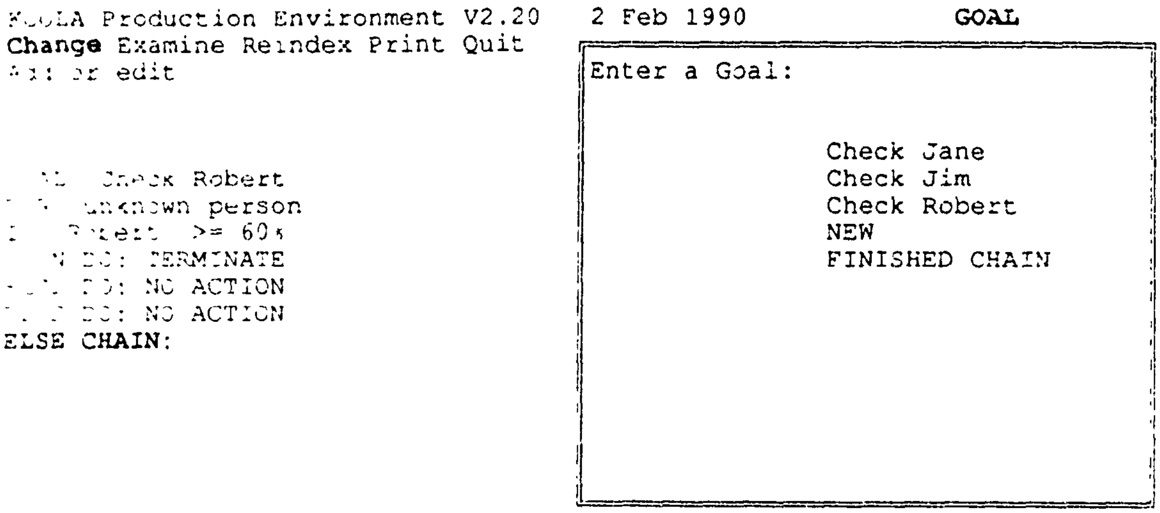


Figure 9. High-level flow for the HIDS knowledge-based system.

With the high-level flow of HIDS mapped out, we were ready to enter the goals for this application. We employed KOOLA goal templates in the programming shell to do this. An example of a goal template for this application is illustrated in Figure 10. The name of the goal is: *Check Robert*, and states that if the probability of the goal **belief:** *unknown pers--Robert* exceeds 60% then the application will terminate. This means that, a probability of over 60% indicates that the person is identified and the application has solve its problem

Figure 10 also shows the state of a goal template while the field "else chain" is being entered. This field is highlighted with an astrict. The programmer would be using the menu on the right side of the screen to select a goal identifier for this field. The



```
KOOLA Production Environment V2.20    2 Feb 1990              GOAL
Change Examine Reindex Print Quit
Add or edit                            Enter a Goal:

                                                 Check Jane
   GOAL  Check Robert                            Check Jim
   IF  unknown person                            Check Robert
   IS  Robert  >= 60%                            NEW
    THEN DO: TERMINATE                           FINISHED CHAIN
   [illegible] DO: NO ACTION
   [illegible] DO: NO ACTION
ELSE CHAIN:
```

Figure 10. Template for entering a goal.

identifier could be either the name of another goal or the terminating identifier.

4.4.3. Entering Enquiries

By entering a set of **goals** for the application, we created a set of **goal beliefs** which must be solved. For example, the previous goal template contained the belief *unknown pers--Robert* in its antecedent. It does not, however, indicate in any way how to ascertain this **belief**. KOOLA production rules are the primitives which lets a programmer express the heuristic relationship between *external facts* and **beliefs**. Before we can start entering these rules, however, the set of *facts* available to the application must be defined.

The two KOOLA primitives which we used to define what facts were available to the application was the **internal and external enquiries**. The set of possible **external**

enquiries is limited by the operating system because they defines a fact that a target system may ascertain from its operating system. For example, we used the external **enquiry** : *ISDN data -- is channel free*, before deciding to send a message in HIDS.

Internal enquiries define questions that may be asked of the end user, and are defined by the application programmer. The question asked will depend on the type of

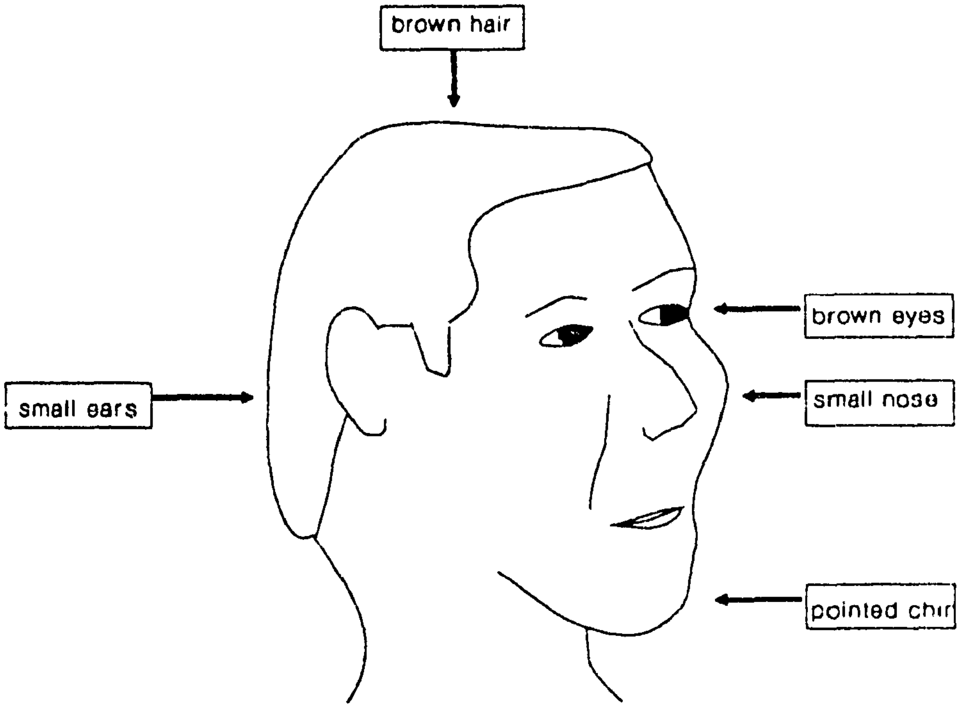
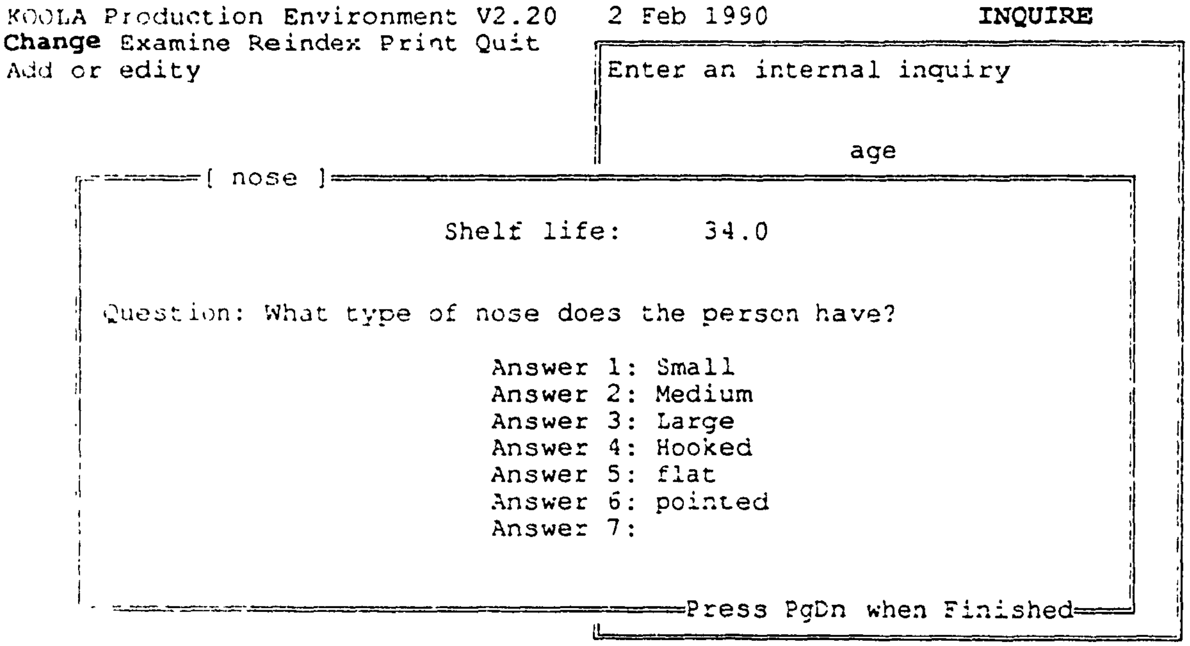


Figure 11. Possible point of observations of a person's face.

data needed. For HIDS, we examined a picture of a person as in Figure 11, to see what kind of information we could use to identify someone. This figure shows the type of human facial features which might help HIDS identify a person. Once we decided the type of facts we could draw from the end-user we started to enter the **internal enquiries** to define it.

A KOOLA **internal enquiry** template is shown in Figure 12. It corresponds to the question: *unknown person--nose.* It contains text for the question and lists all valid answers. The first three responses are ordered from small to *large.* This order is exploited in the rule that we will show next.



```
KOOLA Production Environment V2.20   2 Feb 1990          INQUIRE
Change Examine Reindex Print Quit
Add or edity
                                     Enter an internal inquiry

                                                    age
   [ nose ]

                        Shelf life:      34.0

   Question: What type of nose does the person have?

                          Answer 1: Small
                          Answer 2: Medium
                          Answer 3: Large
                          Answer 4: Hooked
                          Answer 5: flat
                          Answer 6: pointed
                          Answer 7:

                                         Press PgDn when Finished
```

Figure 12. A template for entering a question.

4.4.4. Entering Production Rules

Once we knew what **goal beliefs** we wanted to solve, and which facts would be available; we were ready to start the toughest part of developing HIDS--entering production rules.

In chapter 3 we described the KOOLA **rule** as: *A probabilistic equation that maps facts into beliefs.* This implies that in order for a programmer to define **rules**, she or he must use probabilities to convert heuristic knowledge of the problem domain into KOOLA **rules**. Since any KOOLA knowledge-based application must contain **rules**, a KOOLA programmer must be able to use the **rule** template correctly. The programming of HIDS was no exception.

By the very nature of heuristic knowledge its use is hard to describe. This type of knowledge is uncertain, not fully understood, subject to personal observations, and based on general rules-of-thumb. In Figure 13 we have organised some of the heuristics knowledge used in HIDS, to show how certain facts infer a given **goal belief**. This drawing illustrates that five facts are needed to define a person's face, and that based on face and body, a general belief can be inferred. Any one can argue against the heuristics we presented in Figure 13. The only way to decide if we are right or wrong is to run the knowledge-based system many times. If it operates with the degree of correctness required by the problem then its heuristics are good.

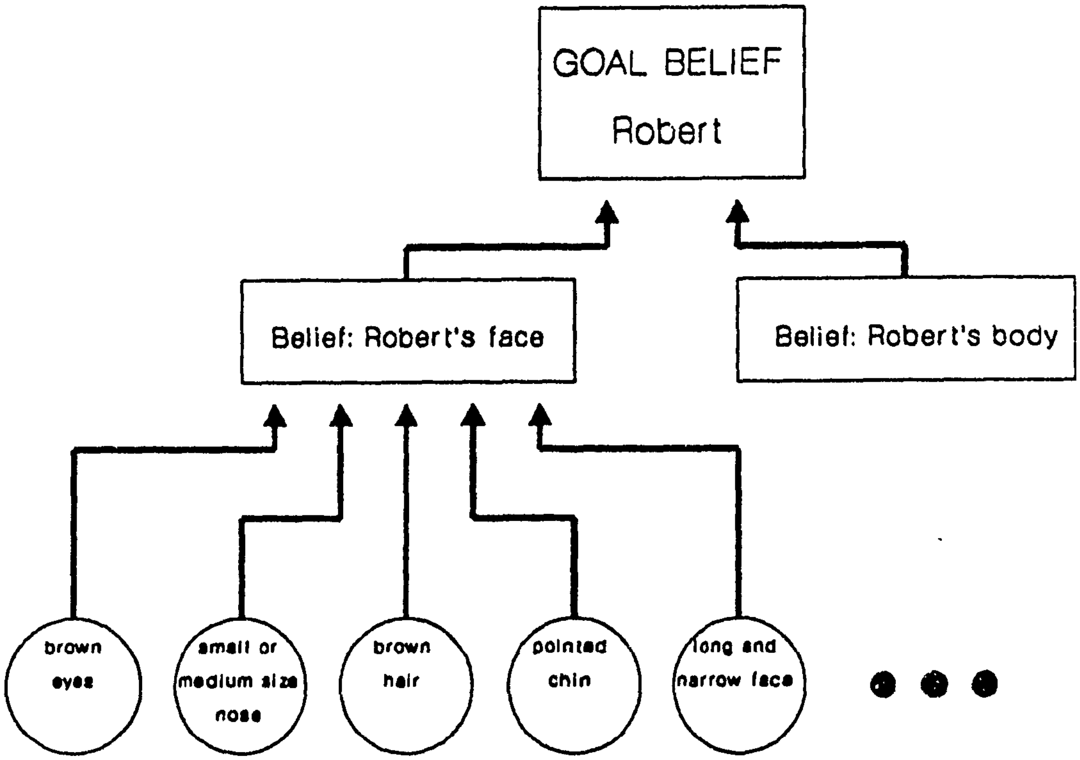
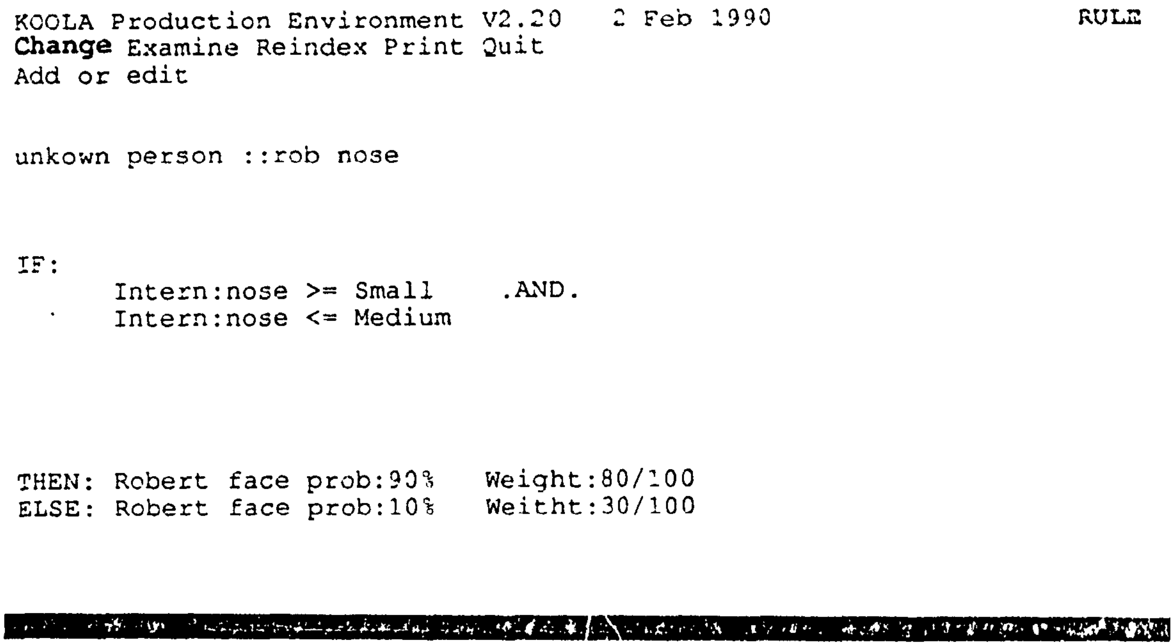


Figure 13. An inference network illustrating some of the heuristics employed in HIDS.

The KOOLA rule template that defines the HIDS rule *unknown person--Robert* is shown in Figure 14. We based the antecedent of this rule on the end-user's observation of the unknown person's nose size. As we alluded to in the previous section, this rule exploits the ordering of the allowable responses in the **enquiry**. The rule states that if the size of the nose is between small and medium, the system may conclude that the probability of the face being Robert's is 90%. It also indicates that this conclusion carries a *weight* of 80 over 100. The rule also states that if the antecedent is not met, the system



```
KOOLA Production Environment V2.20   2 Feb 1990                        RULE
Change Examine Reindex Print Quit
Add or edit

unkown person ::rob nose


IF:
      Intern:nose >= Small    .AND.
      Intern:nose <= Medium




THEN: Robert face prob:90%   Weight:80/100
ELSE: Robert face prob:10%   Weitht:30/100
```

Figure 14. Rule template.

is given licence to conclude that the probability of the face being Robert's is 10%, with a lesser weight of 30 over 100.

Choosing the *probabilities* for **rules** and **goals** is perhaps the most difficult part of KOOLA programming. We selected the thresholds for **goal beliefs** in HIDS to be 60% with the idea that 50% would imply being half certain of a person's identity. The previous rule would assign a probability of 90% to a favourable observation (that the nose meets Robert's criterium), and 10% for an unfavourable one (that it does not), i.e. probabilities are assigned around a central probability of 50%. Thus, this rule will either increase the certainty of a belief, or it will decrease it.

We assign a higher *weight* for a favourable outcome of the previous rule than an unfavourable outcome. We employ this seemingly unbalanced scheme for most of HIDS' rules. It reflects the idea that a *favourable observation strongly supports a conclusion*, while an *unfavourable observation weakly disproves a conclusion*. Again, the sceptic may wish to argue with this scheme, but this idea does work.

A complete examination of probabilistic reasoning in an intelligent system is far too complex a subject to cover here. It is, however, a very important concept in KOOLA programming. For this we can strongly recommend reference [20]. The other consideration in this type of reasoning is to understand the algorithms used to infer beliefs in KOOLA.

4.4.5. Summary

In this section we summarise the ideas presented in this chapter on creating knowledge-based systems with KOOLA.

From the sample application, we defined three sequential steps for building a knowledge-based application from KOOLA. They are summarised as follows:

1. Establish the application flow with **goals**
2. Decide what facts are available with **enquiries**
3. Define a set of production rules using heuristics from the problem domain

The issue of using KOOLA's support for uncertainty was also presented. From this, we saw that the most important and most complex part of knowledge-based programming is dealing with uncertainty. Also, that the primitives available in KOOLA for dealing with this problem are strong.

CHAPTER 5

IMPLEMENTATION OF A KOOLA RUN-TIME INFERENCE ENGINE FOR THE ISDN WORKSTATION

In this chapter we describe the implementation of the KOOLA knowledge-based system run-time shell. The source code for this shell was written for the RMX C-286 "C" compiler, to run under the RMX operating system. Appendix 4 contains a complete listing of this source code. The KOOLA run-time shell supports knowledge-based applications developed in the KOOLA programming environment.

5.1. Software Approach

We follow an approach to software development referred to as the object- oriented approach [19,24,25]. The main concept in this approach is the division of software into self contained modules. Instead of using globally accessible data-structures, all major data-structures are place into modules, which are called *objects*. Any routine that needs to manipulate a data-structure in a foreign object does so by calling a special access function in the foreign object. Consequently, routines never directly access data-structures that are not in their object.

In the object-oriented vernacular, software engineers consider an access function a *method* that belongs to an object [19]. The data-structures and methods are therefore called *members* of their object [19]. The process of calling a method becomes *sending a message* to an object [19]. In an abstract sense, a software engineer views the execution of an object oriented program as a series of messages being sent, received, and acted on.

Another important consideration in object-oriented programming is the criterium by which the members of an object are organised. The correct approach to follow is to group things together that have common characteristics or use the same data [25]. This permits each object to be developed and tested individually. Another beneficial consequence of this approach is the improvement in source code re-usability [24].

We did not use an object-oriented language for the implementation[1]. This meant that the compiler did not enforce the object-oriented constraints, like not directly accessing data across an object boundary. But, by imposing the set of constraints as programming conventions, it became possible to adopt the object- oriented approach to conventional "C" programming. The caveat was that the set of constraints has to be self-imposed rather than compiler enforced, which meant that there was no error checking for violations

Employing object oriented constructs, we organised the KOOLA run-time kernel into six objects. Figure 15 depicts the software structure resulting from this organisation Each object in KOOLA is represented as a block in Figure 15. Every object except the bottom one can only use the *method* [24] of the object below it.

---

[1] Initially we attempted to port MS-DOS Guidelines C++ compiler to the RMX operating system. When this proved to be unsuccessful, we settled on using the RMX "C" stand-alone.

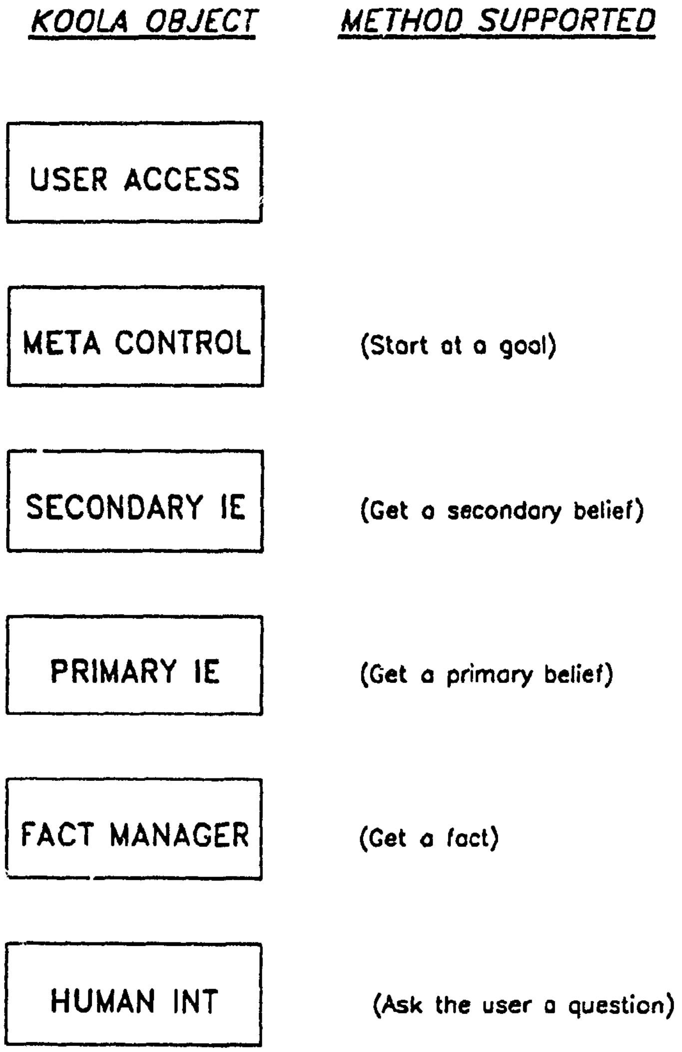


Figure 15. Object-oriented organisation of the KOOLA shell

bottom one can only use the *method* [24] of the object below it.

The most important consequence of this organisation is the support it gives for data abstraction. With these constraints in place, *member functions* at level *n* can only use the method of the next object down. Further, the functions at level *n* must go through level *n-1* to access information stored in any object at a lower level than *n-1*. If such a system is designed correctly, these constraints do not impose a severe performance reduction. They do, on the other hand, go a long way in isolating the complexity of the complete system into more manageable sub-modules.

5.2. Architecture

The actual implementation of the KOOLA shell follows from the architecture we proposed in Chapter 2. It satisfies the symbolic processing requirements of a KOOLA production system, as discussed in Chapters 3 and 4.

5.2.1. Comparison Between the General and Actual Architectures

Figure 16 illustrates the experimental architecture of the KOOLA run-time shell. Since this implementation follows from the intelligent personal workstation architecture (Figure 6), we next show the relationship between the general and experimental architectures.

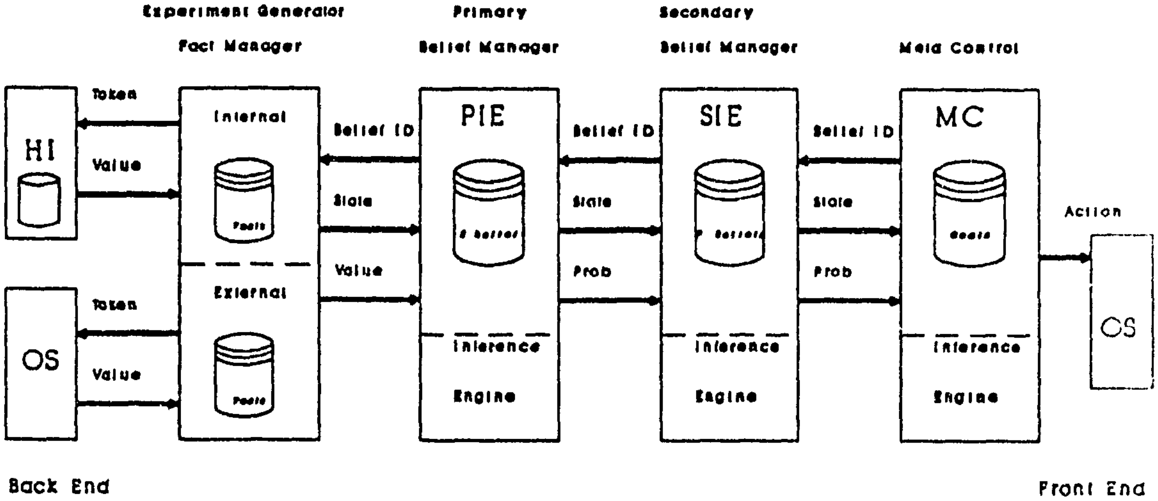


Figure 16. KOOLA Architecture

In the experimental architecture, we have combined the functions of the BM and some of the functions of the IE. The resulting software has been divided into a primary and secondary belief manager/inference engine, which is depicted in Figure 16. The rest of the functions associated with an IE are implemented as the meta-control object.

In the experimental architecture, the EG comprises the experiment generator object, the human interface, and the operating system primitives used by the experiment generator. Functions of the AG are partly handled by the meta-control object, but mostly by the operating system.

The learning capabilities illustrated in Figure 6 correspond to the KOOLA language. This is not a full implementation of its dynamic learning capabilities we envisioned, but is sufficient to satisfy the requirements of the experimental ISDN workstation.

5.2.2. Detailed Implementation

The KOOLA run-time shell is a hybrid inference engine that processes forward chaining *goal rules*, backward chaining *production rules*, and the information stored as *requests* and *actions*.

The **front end** of the architecture, which is on the right side of Figure 16, comprises the *meta control centre*, and it receives commands from the operating system. The **back end** comprises the *fact acquisition centre* which uses facilities in the operating system to gather facts from the ISDN network or the user. To illustrate the operation of this system we examine the processing done to solve a set of goals.

The **Meta Control** (MC) contains a knowledge base of goal rules and its own inference engine which uses them. The knowledge base is separate from the inference engine. Consequently, a knowledge engineer may change the knowledge base without modifying the source code of the inference engine. But because we want this system to run optimally, the knowledge base is linked with the inference engine when the system is *built* in RMX. This quasi separation between declarative and procedural knowledge is present throughout this architecture.

In general, KOOLA processing starts only after the MC receives a valid **Goal Identifier** (GID). Thus when the operating system sends the MC a GID it is informing it of two things: that it should start processing; and, that it should start at the goal indicated. The GID is a system token which uniquely identifies one of the goal rules in the knowledge base. Information stored by the identified goal rule is used to continue the processing.

The first piece of information that the MC uses from the goal rule stored in its knowledge base is the identifier for the secondary belief (or goal belief) the antecedent of the rule is based on. The MC does not store the probabilistic value of any beliefs. Consequently, to ascertain the value of the goal belief it passes a request to the next object to its left in Figure 16.

The **Secondary Inference Engine** (SIE) receives requests to determine the probability value of goal beliefs from the MC. Like the MC, the SIE has an inference engine and a knowledge base. One type of information that the SIE's knowledge base contains defines a static inference network that corresponds to the rules programmed by the knowledge engineer[1]. The other information is dynamic information and this includes the state of all beliefs and their probability-values (if defined).

The first thing done by SIE when it receives a request identifying a goal belief is to determine its state in the knowledge base. If it is the case that the state of the goal belief is *defined* then no further processing is required from SIE, and it returns the value of the belief to the MC. On the other hand if the state is *undefined* more processing is needed to solve the goal belief.

To solve a belief, SIE uses the inference network defined in its knowledge base. For any given belief the inference network defines the beliefs that support this belief. Thus to solve the goal belief SIE must solve all of its supporting beliefs.

According to the KOOLA definition the supporting beliefs of a secondary belief may be either primary beliefs, or secondary beliefs. If they are secondary beliefs then they are also solved the same way. Obviously this is a recursive relationship. A FIFO queue manages the recursion, that will be defined by the main system algorithm in the next section.

At some point, processing on an inference network always ends with a set of primary beliefs. Since SIE does not solve primary beliefs, like the MC, it passes them as requests to the next module on its left (the PIE).

The **Primary Inference Engine** (PIE) receives requests from SIE for the probability-values of primary beliefs. Like the MC and SIE, PIE has an inference engine

---

[1] This refers to the type of diagram shown in Figure 4.2 (a KOOLA inference network)

and a knowledge base. The knowledge base contains a one-level inference network that relates beliefs directly to facts. It also contains dynamic information that is similar to the dynamic information in the SIE knowledge base.

Like the SIE, PIE first checks if the state of a requested primary belief is *defined*. If it is, then the value of that belief is returned immediately. If it is not, then PIE looks in its knowledge base to find all of the facts needed to support the belief. Since its inference network is only one deep, there is no recursion in PIE.

To ascertain the value of the facts need to calculate a primary belief, PIE sends requests to the next object on its left.

The final object in this sequence is the **Fact Manager** (FM). We say that it consists of procedures and a data base since it does not do any heuristic processing. Like the previous two objects it keeps track of states and values. The values it keeps, though, correspond to actual facts (for example the user's answer to a question). Values are store as 32-bit floating point numbers.

If the FM does not already have the value of a requested fact it either requests it from the operating system or it requests it from the human interface.

Going back to the first object we examined, the **Meta Control** (MC) object we can complete the discussion. The MC gets back the value of its goal belief and can *fire* its rule. Based on the outcome, and the current goal rule it is working on, the MC may send the OS an *action request* and/or it may chain to another goal rule, thus propagating the processing.

5.2.3. Algorithm For Solving a Goal Belief

This section contains the main KOOLA algorithm for solving an inference.

1. Place the given goal belief on a LIFO queue.

2. Repeat the next steps until the goal belief is solved, or found to be unavailable (or block).

3. Examine (do not remove) the next belief at the head of the queue.

4. If the belief is a fact:
   4.a and the state of the fact is *not available* or *defined*, then remove the fact (belief).
   4.b else leave the fact on the queue, request that it be solved, wait until it gets solved (block).

5. Else, if the state of the belief is *defined*:
   5.a and the belief is the goal belief, then stop and announce success.
   5.b else remove the belief from the queue.

6. Else, if the state of the belief is *not available*:
   6.a and the belief is the goal belief, then stop and announce failure.
   6.b else remove the belief form the queue.

7. Otherwise, the state of the belief is *undefined*. Expand the belief into all its supporting beliefs, called its children. For each of the children, do the following steps

   7.a If the state of the child is *defined*, save the probability of the child (If it is a fact, save its value).
   7.b Else, if the state of the child is *not available*, do nothing with it.
   7.c Else, the state of the child is undefined. Place the child on the queue.

8. If all states of all the children were *not available*, then define the belief the same way.

9. Else, if all of the children were defined, solve the probability of the belief, store it, and set the state of the belief as defined.

10. Otherwise, some of the children were not defined, so leave the belief on the queue.

CHAPTER 6

SUMMARY AND CONCLUSIONS

In this chapter we summarise the main points of this thesis, present the major conclusions, and provide some suggestions for future work.

6.1. Summary

In chapter 1, we introduced the idea of an intelligent ISDN personal workstation. The purpose of our research was to design the knowledge based component of such a system. We determined that the design should be bounded by the desire to have a target system with these two key abilities:

1. The ability to help people exploit ISDN information services
2. The ability to support autonomous control activities.

In chapter 2, we presented four characteristics of an ISDN based intelligent system with these abilities. Those characteristics were:

1. **Control Workstation Resources:** have a software mechanism that permits application programs to control all hardware connected to the workstation.
2. **Handle Real-Time Information:** the ability to recognise spoiled information caused by the elapse of time, and take the necessary steps if it occurs.
3. **Easy to Use and Program:** A user of the system should find it easy to modify (program) the workstation so that it can work on her or his own particular problems.

4. **Multiple Problem Solving:** Deal with an environment in which more than one problem at a time may occur.

In the second half of chapter 2 we used these characteristics as the underlying requirements for our approach to the design. In particular, our approach involved developing a knowledge-based system shell. The first part of our solution was to propose a run-time architecture for an ISDN-oriented knowledge-based system. The important contributions in the first part of our approach include the following developments:

- **An inference engine kernel capably of working on many problems at a time.**
- **Separate inference engines for dealing with different symbolic processing needs**
- **Strongly connected to the external environment**
- **Extremely controllable by higher-levels of the workstation's operating system.**

The second part of our approach involved developing a special programming system for our run-time architecture. We defined the KOOLA rule-based production system as the primary method to program our knowledge-based system shell. The following five attributes of KOOLA distinguishes it from other production systems (*i.e.*, our contributions):

1. **Meta-Control rules:** A class of rules used to guide the symbolic processing.
2. **Network Oriented Fault Tolerance:** A unique use of probabilities and uncertainty that permits the system to continue processing in spite of the inability to ascertain some information.

3. **Variable Construct:** A production system paradigm which permits the system to solve multiple occurrences of the same problem while using a common set of rules.
4. **Expected Shelf Life:** Permits a programmer to specify how long a fact can remain valid.
5. **ISDN Actions and Requests Objects:** An object-oriented programming method of gathering information from the network, which is fully integrated to the symbolic processing of the inference engine.

6.2. Conclusions

Since the ISDN workstation project is still on-going, we cannot draw too many conclusions from testing our implementation of the KOOLA knowledge-based system. We can, however, draw conclusions from the research we performed on artificial intelligence and provide suggestions for continuing the project.

The major conclusions we draw from this research are the following ones:

A network-connected, knowledge-based system must deal with the loss of information.

Information, or data, in such a system may sometimes be unavailable due to a network problem, or the inability to access an external service (for example a third-party data-base service). To overcome this potential problem, the system's inference engine should be designed with the ability to continue processing despite the fact that it could be missing relevant data. We implemented this idea in the design of the KOOLA inference engine.

Rules are more expressive if they contain primitives for quantifying importance.

When referring to importance we make a distinction between using probabilities or certainty factors in the consequence of a rule, and expressing how important the rule is. For example, when more then one rule infers different levels of a given belief's probability, *importance* defines which rule should be given greater *weighting* to the outcome. In KOOLA we implemented this idea with the **weight** paradigm in the consequence of rules.

A knowledge-based system can run under any primitive operating system environment.

Normally a system must provide symbolic-processing primitive before it can support a knowledge-based system. If, however, the run-time part of a knowledge-based system is written in a general-purpose, well support language (*e.g.* C), it can be ported to any platform that has a corresponding compiler. Our run-time implementation of KOOLA was on such a platform (the RMX OS). Our knowledge entry, which requires symbolic processing primitives, was developed on a separate platform. In the KOOLA system, knowledge is pre-compiled in a form acceptable to the RMX C compiler. When compiled, and linked with the binary shell they form a target knowledge-based system.

6.3. Suggestions for Future Work

The main suggestion we propose is to complete the project *as is*. This would involve using KOOLA to build a *full-featured* knowledge-based system or even an expert system. From this, a final evaluation of this knowledge-based system shell would be possible.

The second suggestion we propose involves taking the ISDN workstation out of the home environment. We built the workstation on top of the RMX real-time operating system kernel. Currently we do not make use of this potential for real-time processing. With a real-time operating system, however, the platform might be able to support the low-level, high-speed, switching and routing functions required of a hybrid private branch exchange, like the one shown in Figure 17.

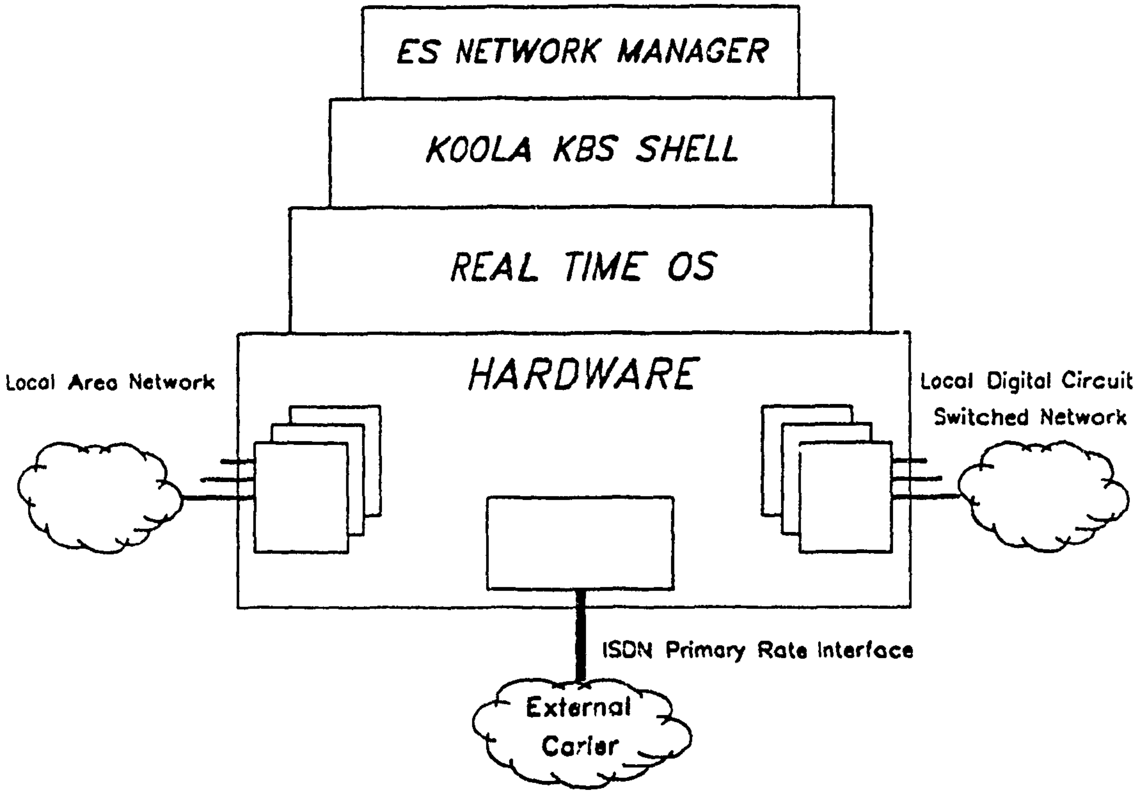


Figure 17. Block diagram of a possible configuration for a hybrid Private Branch Exchange (PBX).

A network management expert system built on this platform, would provide intelligent control of local network resources. The real-time OS kernel would guarantee that the hardware would get the real-time response it needed, while the expert system would make intelligent routing and resource assignment decisions based on cost and utilisation criteria. Apart from requiring a few software modifications, the system would need a primary-rate ISDN connection and LAN access hardware.

REFERENCES

[1] **W. W. Benjamin** *et al*, "Computers for symbolic Processing", *Proceedings of the IEEE* (invited paper), vol.77, no. 4, April 1889, pp. 509-539.

[2] ---, *Integrated Services Digital Network (ISDN)*, VIIIth Plenary Assembly CCITT Recommendations of the Series I Red Book, Vol.3., Fascicle III.5, 1985.

[3] **G. Hanover**, "Networking the Intelligent Home", *IEEE Spectrum*, Vol. 26, No. 10., Oct 1989, pp. 48-49.

[4] **C. V. Ramamoorthy, W. W. Benjamin**, "Knowledge and Data Engineering" *IEEE Transactions of Knowledge and data engineering*, Vol. 1 No. 1., Mar 1989, pp. 9-15.

[5] **M. Sutter & P. Zeldin**, "Designing Expert Systems for Real-Time Diagnosis of Self-Correcting Networks," *IEEE Network*, vol. 2, no. 5, September 1988, pp. 43-51.

[6] **Robert L. Brown**, *et al*. *Levels of Abstraction in Operating System.* NASA Technical Report Contract NAS2-11530, July 1984.

[7] **P. Winston** *Artificial Intelligence (Second edition)*, Addison-Wesley, Mass., 1984.

[8] **A. S. Tanenbaum**, *Computer Networks*, Second Edition, Prentice Hall, Englewood Cliffs N.J., 1988.

[9] **W. Stallings**, *ISDN an introduction*, Macmillan, New York, N.Y., 1989.

[10] ---, *ISP188/ISDN Basic Access Product Binary Manual*, DGM&S, New Jersey, 1989.

[11] ---, *Microelectronics Data Book*, Mitel, Ottawa, Ontario, 1989.

[12] ---, *ISDN Development Kit 29C53 User's Manual*, Intel, California, 1988.

[13] **J. Chatterley**, B. Newman and R. Wellard, "The ISDN PC: A Flexible Voice Data Workstation," *IEEE Globecom '86*, 1986, pp. 1504-1508.

[14] **P. Waterman**, *A Guide to Expert Systems*, Addison Wesley, California, 1986.

Computer Systems," *Ph.D. Dissertation*, University of California, Berkeley, May 1988.

[16] **B. Gates**, "The 25th Birthday of BASIC," *BYTE*, McGraw-Hill, New York, N.Y., Vol.14, No.10, October 1989, pp.269-276.

[17] **J. Pasquale**, "Using Expert Systems to Manage Distributed Computer Systems," *IEEE Network*, vol. 2, no. 5, September 1988, pp. 22-27.

[18] **Z. Slodki**, *A Knowledge-Based, Event-Driven, Real-Time Operating System for an ISDN Personal Workstation*, M. Eng. Thesis, Concordia University, Montreal, June 1990.

[19] **B. Stroustrup**, *The C++ Programming Language* Addison-Wesley New Jersey 1987.

[20] **J. Pearl** *Probabilistic Reasoning in Intelligent Systems* Morgan Kaufmann, California, 1988.

[21] **L. A. Zadeh**, "Knowledge Representation in Fuzzy Logic", *IEEE Transactions of Knowledge and data engineering*, Vol. 1 No. 1., March 1989, pp.89-100.

[22] **M Stefik** *et al*, "The Architecture of Expert Systems", *Building expert system*, Addison Wesly, Reading, Mass., 1983.

[23] **B. W. Kernighan, D. M. Ritchie**, *The C Programming Language Second Edition*, Prentice Hall, New Jersey, 1988.

[24] **K. Gorlen**, "An Object-Oriented Class for C++ Programs," *Software Practice and Experience*, vol. 17, John Wiley & Sons Ltd, Dec 1987, pp.899-922.

[25] **D, Hy**, *C/C++ for Expert Systems*, MIS PRess, California, 1989.

[26] **J. Martin, S. Oxman**, *Building Expert Systems* Prentice Hall, New Jersey, 1988.

[27] ---, *System 120 Hardware Installation and User's Guide*, Intel, California, 1988.

[28] ---, *IBM Technical Reference Personal Computer AT*, IBM, Florida, 1984.

APPENDIX I

KOOLA SOURCE CODE FOR HIDS APPLICATION

```
KOOLA Production System     Rule Listing  Date: 06/07/1990

   FOR: isdn

   Primary Rule ( 1): prim file free
      IF   External: file conct stat ==     0.00
      THEN: file free p  90 Weight: 100
      ELSE: file free p  10 Weight: 100

   Secondary Rule ( 2): r1
      IF: use voic
      THEN: voice 100 Weight: 100

   Primary Rule ( 3): r2p
      IF: phone == yes
      AND  External: voic conct stat ==     0.00
      THEN: use voic  90 Weight: 100
      ELSE: use voic  10 Weight: 100

   Secondary Rule ( 4): sect file free
      IF: file free p
      THEN: file free s 100 Weight: 100

   Primary Rule ( 5): user data p
      IF: phone == no
      AND  External: file conct stat ==     0.00
      THEN: user data p  90 Weight: 100
      ELSE: user data p  10 Weight: 100

   Secondary Rule ( 6): user data s
      IF: user data p
      THEN: user data s 100 Weight: 100
```

```
KOOLA   Rule Listing  Date: 06/07/1990                    Page:  2

   Primary Rule ( 7): voice free p
      IF   External: voic conct stat ==      0.00
      THEN: voice free p  90 Weight: 100
      ELSE: voice free p  10 Weight: 100

   Secondary Rule ( 8): voice free s
      IF: voice free p
      THEN: voice free s 100 Weight: 100
   FOR: operator

   Primary Rule ( 1): want to ID p
      IF: want to ID pers == YES
      THEN: wnt I.D. prim 100 Weight: 100
      ELSE: wnt I.D. prim   0 Weight: 100

   Secondary Rule ( 2): want to ID s
      IF: wnt I.D. prim
      THEN: wnt I.D. person 100 Weight: 100


   FOR: unknown person

   Primary Rule ( 1): Jane body gen
      IF   Question: body weight >     40.00
      AND  Question: body weight <     50.00
      AND: body breasts >= Small
      AND: body breasts <= Full figured
      THEN: Jane body  80 Weight:  90
      ELSE: Jane body  20 Weight:  90

   Primary Rule ( 2): Jane face
      IF: face shape == Medium features
      AND: nose <  Medium
      AND  Question: age >     18.00
      AND  Question: age <     22.00
      THEN: Jane face  90 Weight:  90
      ELSE: Jane face  10 Weight:  40
```

```
KOOLA  Rule Listing  Date: 06/07/1990                    Page:   3

   Primary Rule ( 3): Jane lenght
      IF   Question: body length >     110.00
      AND  Question: body length <     130.00
      THEN: Jane body  90 Weight:  90
      ELSE: Jane body  10 Weight:  90
   Secondary Rule ( 4): Jane s
      IF: Jane body
      AND: Jane face
      THEN: Jane 100 Weight: 100
   Primary Rule ( 5): rob eyes
      IF: eye colour == Brown
      THEN: Robert face  95 Weight: 100
      ELSE: Robert face   5 Weight: 100

   Primary Rule ( 6): rob face 1
      IF: face shape == Long
      AND: face beard == Medium beard but clean shaven
      THEN: Robert face  90 Weight:  80
      ELSE: Robert face  10 Weight:  80

   Primary Rule ( 7): rob length
      IF   Question: body length >     120.00
      AND  Question: body length <     136.00
      THEN: Robert body  80 Weight:  80
      ELSE: Robert body  20 Weight:  80

   Primary Rule ( 8): rob nose
      IF: nose >= Small
      AND: nose <= Medium
      THEN: Robert face  90 Weight:  80
      ELSE: Robert face  10 Weight:  30

   Primary Rule ( 9): rob weight
      IF   Question: body weight >      58.00
      AND  Question: body weight <      78.00
      THEN: Robert body  90 Weight:  80
      ELSE: Robert body  10 Weight:  80

   Secondary Rule (10): robert s
      IF: Robert body
      AND: Robert face
      THEN: Robert 100 Weight: 100
```

APPENDIX II

HARDWARE CONFIGURATION AND SCHEMATIC DIAGRAMS

Figure 18 illustrates the hardware architecture of our ISDN work station. Since we still consider it a development platform, it is constructed from mostly *off the self* components—resulting in a physically large and distributed system. The current configuration consists of the following hardware:

1. Intel system 120 host, running Intel's iRMX II.3 real-time operating system [40].
2. An IBM AT with an Intel **PC53** ISDN basic rate access card [41] connected to its' expansion bus. DGM&S **ISP-188** version 3.0 basic rate ISDN software, configured as a TE, runs on the card [42].
3. A second IBM AT with the same configuration as the previous, only the DGM&S software is set-up to run in NT mode.
4. Custom built *centralised controller.*
5. Three custom built 16/24-bit parallel I/O cards, that are compatible with the IBM AT expansion bus specifications [43].

The Intel system 120 is a 386 based computer configured with 2 M-bytes of RAM and a 387 floating point coprocessor. This configuration is suitable for

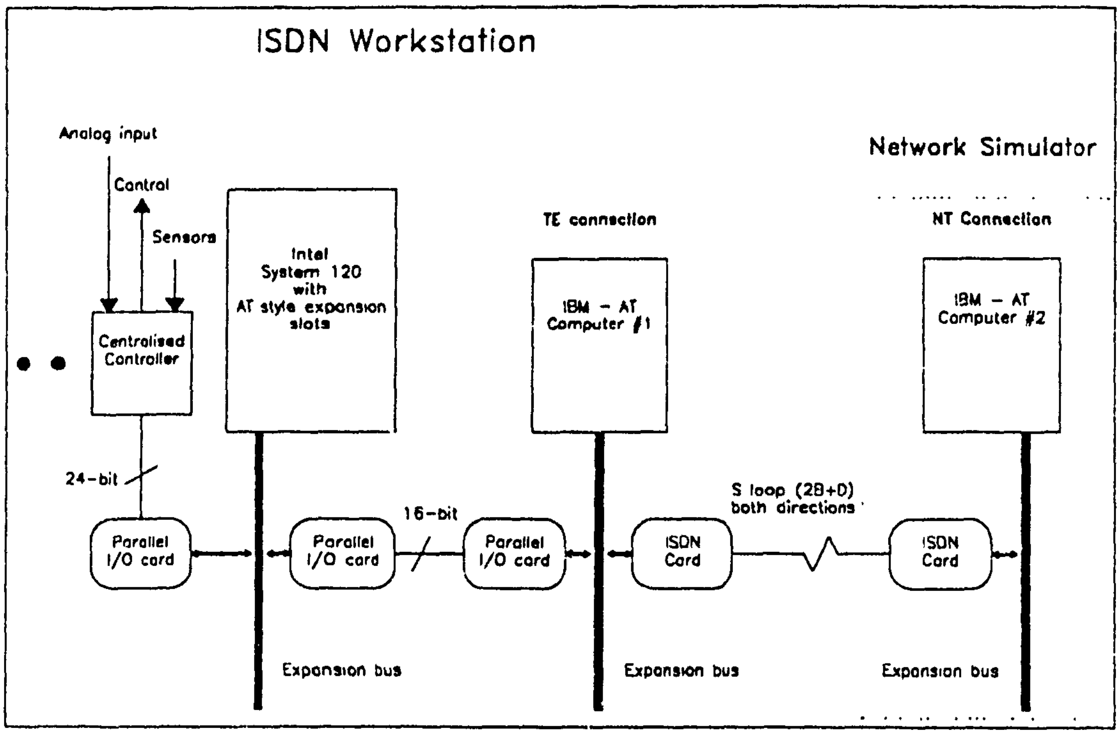


Figure 7
Experimental ISDN personal workstation, shown with the ISDN network simulator.

executing the workstation's operating system with its knowledge-based system. We chose the system 120 as the main host for the ISDN workstation because it can run the Intel real-time kernel; and because its' expansion bus follows the IBM AT standard, increasing the availability of third party expansion cards, and facilitating the development of custom built cards. All programming for the System 120 is done in C, using the Intel **C-286** compiler.

The centralisation of control devices is achieved in hardware by the *centralised controller*. Currently the controller does no processing on its own, but does have an 8-bit analog to digital converter. Other operations performed by the controller includes sampling six inputs and controlling six outputs, all using standard 12VAC signalling. It communicates with the host via one of the 24-bit parallel cards. Additional controllers can be added to the system by placing another parallel card in the System 120.

All ISDN communications go through DGM&S ISDN cards. These cards are compatible with an IBM-AT expansion bus and are all mounted in AT's An 80188 microprocessor provides these card with sufficient processing ability to execute the DGM&S ISDN network layer software. An important limitation with the DGM&S software is that the drivers and the loader are only available for MS-DOS, with the result that the ISDN cards must operate in an IBM-AT, running that operating system.

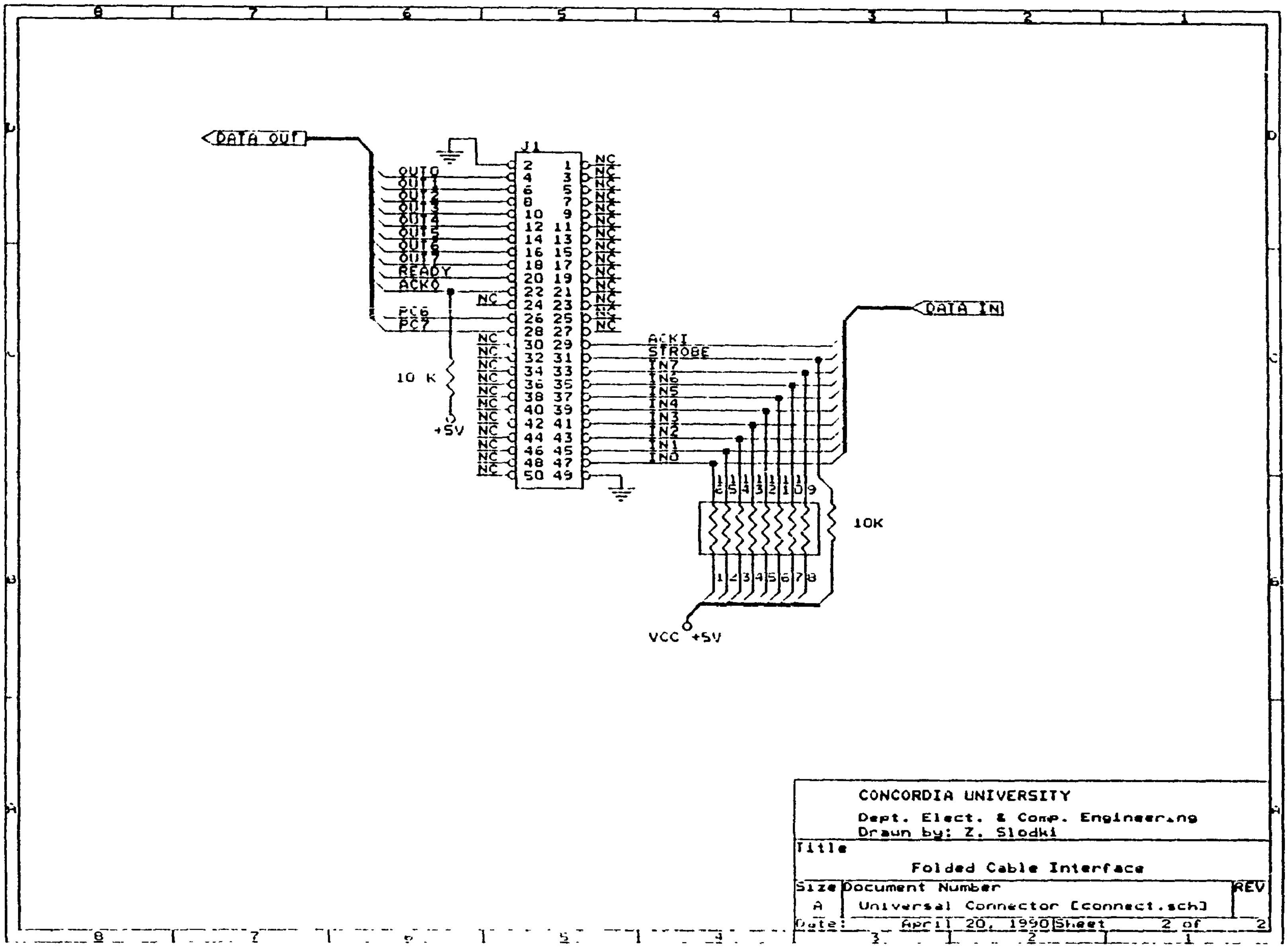
The first IBM-AT provides the ISDN TE[1] access for the workstation's operating system. Owing to the limitations outline above, this card can not operate directly in the iRMX environment, which is why it is connected through a 16-bit parallel card. In the future we plan to develop an iRMX driver for the card, and thus eliminating the need for this sub-system.

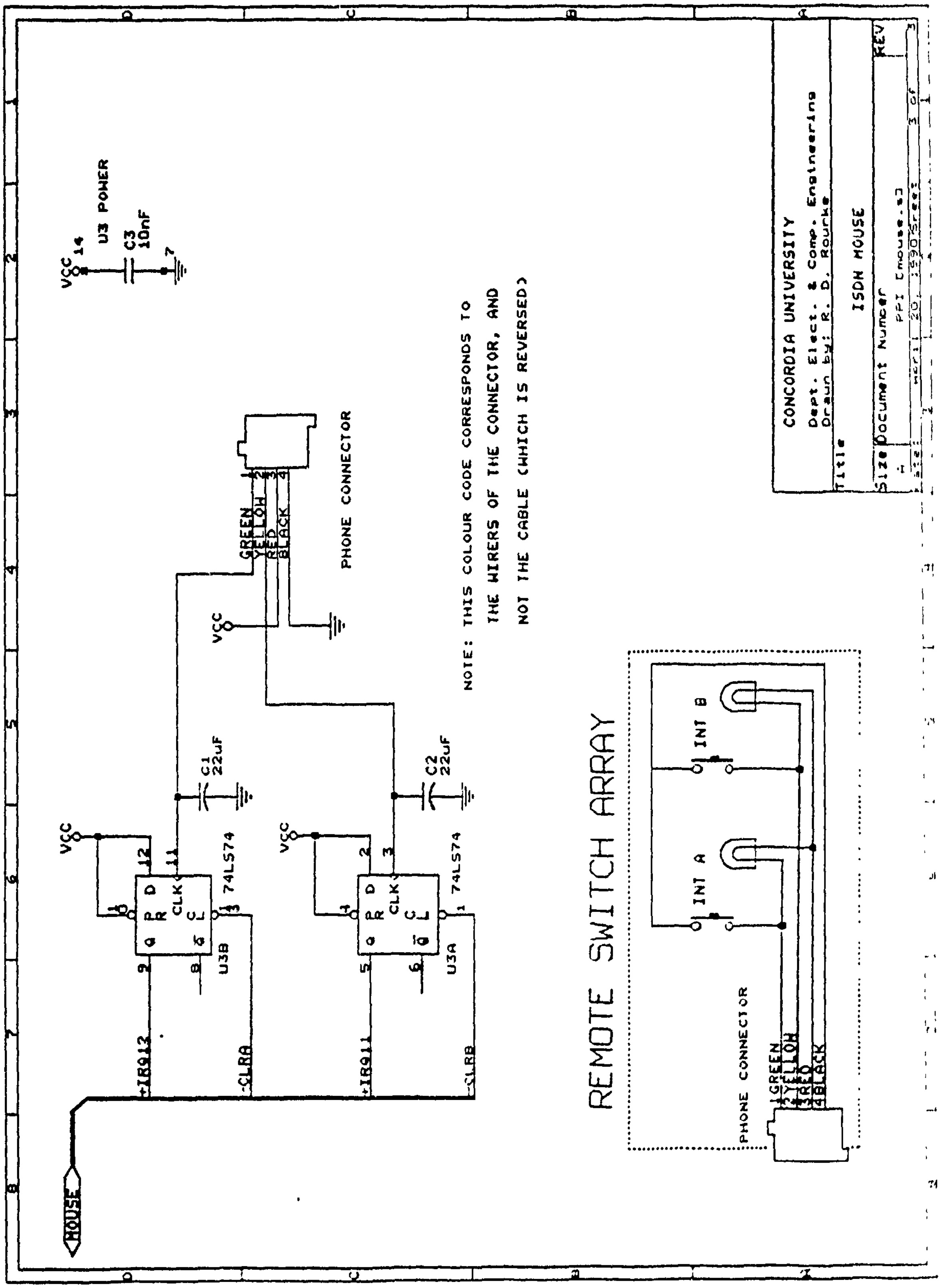
The second AT provides a network simulator to test the ISDN workstation. As such, the ISDN software running on it is configured in ISDN NT[2] mode.

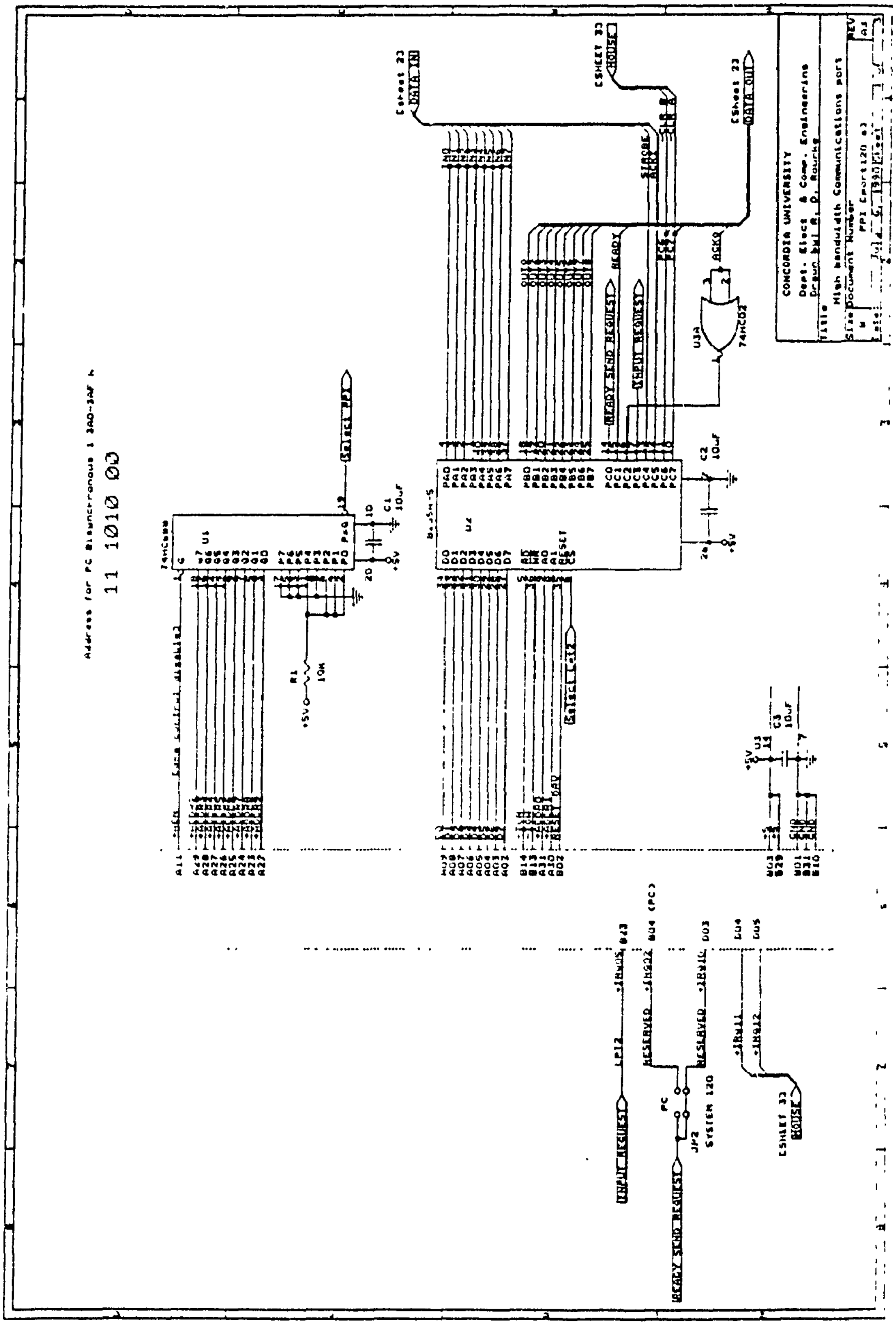
---

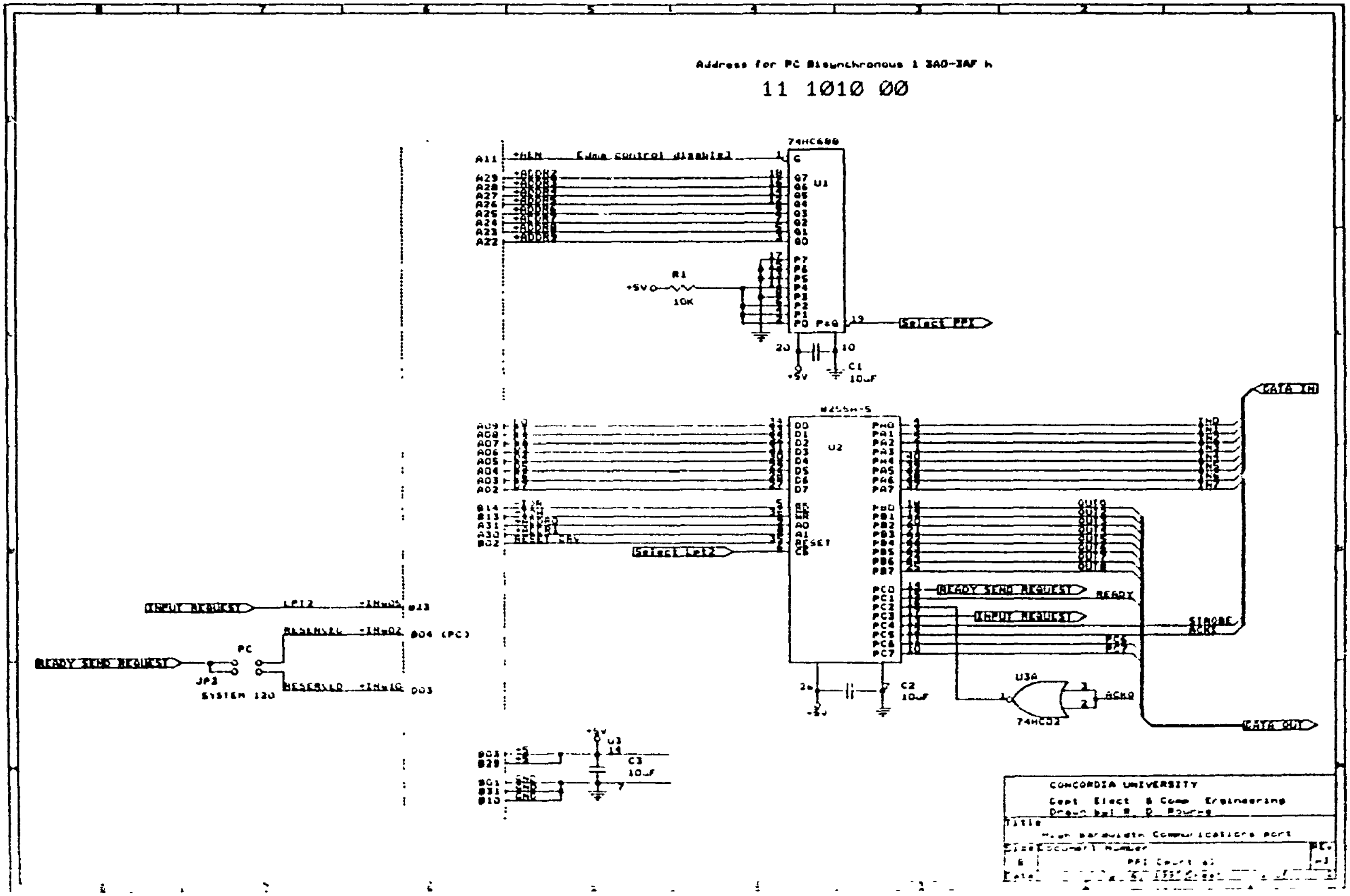
[1] The CCITT defines the reference configuration TE: terminal equipment functions. This functional grouping represents equipment such as voice/data terminals with protocol handling and interface capabilities[26].

[2] The CCITT defined NT: network termination equipment functions as broadly equivalent to layer 1 of the OSI reference model [24]. The equipment also provides timing and power to the TE. It also multiplexes the inboard and outboard chains [26].


DATA OUT
DATA IN
J1
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
READY
ACKO
PC6
PC7
NC
10 K
+5V
ACKI
STROBE
IN7
IN6
IN5
IN4
IN3
IN2
IN1
IN0
10K
VCC +5V
CONCORDIA UNIVERSITY
Dept. Elect. & Comp. Engineering
Drawn by: Z. Slodki
Title
Folded Cable Interface
Size
A
Document Number
Universal Connector [connect.sch]
REV
Date:
April 20, 1990
Sheet 2 of 2


U3 POWER
VCC
14
C3
10nF
7
PHONE CONNECTOR
GREEN
YELLOW
RED
BLACK
1
2
3
4
VCC
NOTE: THIS COLOUR CODE CORRESPONDS TO
THE WIRERS OF THE CONNECTOR, AND
NOT THE CABLE (WHICH IS REVERSED)
C1
22uF
C2
22uF
VCC
12
11
D
CLK
PR
CL
Q
Q̄
74LS74
U3B
10
13
9
8
VCC
2
3
4
1
5
6
74LS74
U3A
+IR912
-CLRA
+IR911
-CLRB
MOUSE
REMOTE SWITCH ARRAY
INT A
INT B
PHONE CONNECTOR
1GREEN
2YELLOW
3RED
4BLACK
CONCORDIA UNIVERSITY
Dept. Elect. & Comp. Engineering
Drawn by: R. D. Rourke
Title
ISDN MOUSE
Size
A
Document Number
PPI [mouse.s]
REV
3
April 20, 1990
Sheet 3 of
A
B
C
D
1
2
3
4
5
6
7
8


Address for PC Bisynchronous : 3A0-3AF h
11 1010 00
74HC688
U1
Select PPI
C1 10uF
R1 10K
+5V
U2
C2 10uF
C3 10uF
U3A
74HC02
ACKQ
READY
READY SEND REQUEST
INPUT REQUEST
STROBE
ACK
SHEET 23 DATA IN
SHEET 23 DATA OUT
SHEET 23 HOUSE
RESERVED
PC
JP2
SYSTEM 120
+IRQ11
+IRQ12
CONCORDIA UNIVERSITY
Dept. Elect. & Comp. Engineering
Drawn by: R. D. Rourke
Title
High bandwidth Communications port
Size
Document Number
PPI Cport120 e)
REV
Date: July 6, 1990 Sheet


Address for PC Bisynchronous 1 3A0-3AF h
11 1010 00
74HC688
U1
G
+MEM
(dma control disabled)
A11
A29
A28
A27
A26
A25
A24
A23
A22
R1
10K
+5V
P=Q
Select PPI
C1
10uF
8255A-5
U2
D0
D1
D2
D3
D4
D5
D6
D7
RD
WR
A0
A1
RESET
CS
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
A09
A08
A07
A06
A05
A04
A03
A02
B14
B13
A31
A30
B02
Select LPT2
DATA IN
DATA OUT
READY SEND REQUEST
INPUT REQUEST
READY
STROBE
ACKI
U3A
74HC32
ACKO
C2
10uF
C3
10uF
U3
INPUT REQUEST
LPT2
RESERVED
B04 (PC)
D03
READY SEND REQUEST
JP3
PC
SYSTEM 120
CONCORDIA UNIVERSITY
Dept Elect & Comp Engineering
Title
High bandwidth Communications port
Size
Document Number
B
Date

APPENDIX III

SOURCE CODE LISTING

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project      :   KOOLA shell                               =
   =    Sub-Project  :   Human interface                           =
   =    File         :   fm.c                                      =
   =    Author       :   Robert D. Rourke                          =
   =    Start Date   :   01 Mar 1990                               =
   =    Update       :   10 Apr 1990                               =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
/*#define         TESTON*/
/*#define   RMXON*/
/*#define         USECOSI*/
/*#define         LINKZEN*/

#include    <stdio.h>
#include    <ctype.h>
#ifdef      RMXON
#include    <udi.h>
#include    <rmx.h>
#endif
#include    "extreq.h"

#include    "huminter.h"

#include    "hi.hx"
#include    "fm.h"

#define           CLEARBUF    getchar();

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   fact_value                                   -
 -    Input       :   highlevel fact ID (exess 500)                -
 -    Output      :   value of the fact                            -
 -    Action      :                                                -
 -    Date        :   22 Feb 90                                    -
 -    UpDate      :                                                -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
float fact_value(hl_factid)
unsigned hl_factid;
{
      if (hl_factid < EXCESS) return(intern_getvalue(hl_factid));
      else return(extern_getvalue(hl_factid-EXCESS));
}

/*
 *          EXTERNAL OBJECTS
 */

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   extern_constructor                           -
 -    Input       : B  none                                         -
 -    Output      :   none                                         -
 -    Action      :   reads questions sets up list of questions    -
 -    Date        :   22 Feb 90                                    -
 -    UpDate      :   05 Mar 90                                    -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

```
/*
 *    Principle storage of the external facts
 */
fact  Extern_fact[NUMEXRQ];

void  extern_constructor()
{
      extern      fact  Extern_fact[NUMEXRQ];
      char  fact_date[DATESIZE];
      fact* this_extern;
      char  dummy[80];
      float shelflife;
      FILE* datafile;
      int         i;

      printf("\n\nLoading the fact manager info...\n");
      /* open the input data file terminate if error */
      if ((datafile = fopen(DATAFILE, "r"))==NULL)
           terror("Missing the main fact file");

      /* remove the header store date stamp */
      for (i=1;i<DATELOC;i++)
           fscanf(datafile, "%s",dummy);
      fscanf(datafile, "%s",fact_date);
      printf ("\nThe enquiry token date-stamp: %s\n",fact_date);

      /* load all external tokens */
      for(this_extern=Extern_fact;;this_extern++) {
            if  (fscanf(datafile,  "%d",&(this_extern->cositoken))==EOF)
break;
            printf(".");
/*          printf("\ntoken: %d",(this_extern->cositoken));*/
            fscanf(datafile, "%f",&shelflife);
/*          printf("   The shelf life: %f",shelflife);*/
            /* initialise the structure to undefined */
            this_extern->state = UNDEFINED;
            this_extern->value = INITVAL;
      }/*for ever*/
      fclose(datafile);
      printf("\nAll facts loaded\n");
} /*end extern_constructor()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :   extern_getstate  extern_getvalu ..gettoken    -
 -    Input        :   extern fact ID  (starting at 1)               -
 -    Output       :   state of the fact                             -
 -    Action       :   looks up the fact and returns it state        -
 -    Date         :   22 Feb 90                                     -
 -    UpDate       :                                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
unsigned    extern_getstate(ext_index)
unsigned    ext_index;
{
      extern      fact  Extern_fact[NUMEXRQ];
      if (--ext_index > NUMEXRQ) terror("Fact ID out of index");
      return(Extern_fact[ext_index].state);
}

unsigned    extern_gettoken(ext_index)
unsigned    ext_index;
{
```

```
    extern      fact  Extern_fact[NUMEXRQ];
    if (--ext_index > NUMEXRQ) terror("Fact ID out of index");
    return(Extern_fact[ext_index].cositoken);
}

float extern_getvalue(ext_index)
unsigned    ext_index;
{
    fact* this_fact;
    extern      fact  Extern_fact[NUMEXRQ];

    if (--ext_index > NUMEXRQ) terror("Fact ID out of index");
    if ((this_fact=&Extern_fact[ext_index])->state == UNDEFINED) {
        this_fact->value = rqst_ext(ext_index);
        this_fact->state = KNOWN;
    }
    return(this_fact->value);
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -  Function    :  rqst_ext                                                  -
 -  Input       :  extern fact index (starting at 0)                         -
 -  Output      :  value of the fact requested                               -
 -  Action      :  Displays the token of the fact, ask for numer             -
 -  Date        :  05 Mar 90                                                 -
 -  UpDate      :  10 Apr 90                                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
float newval;
float       rqst_ext(fact_id)
unsigned    fact_id;
{
    extern      int   WCOSI$RESPONSE$DMB;
    extern      int   WCOSI$COMMAND$DMB;
    int         except;
    extern      fact  Extern_fact[NUMEXRQ];
    /*
     *  Later on this will pass a message to the OS to get the fact
     */
    printf("\n Simulating mailbox call to get info from COSI:");
    printf("\n The token is: %u ", Extern_fact[fact_id].cositoken );
    wait();
#ifdef USECOSI
    rq$send$data (WCOSI$COMMAND$DMB, &Extern_fact[fact_id].cositoken, 2,
     &except);
    rq$receive$data (WCOSI$RESPONSE$DMB, &newval, FOREVER, &except);
    printf ("\n Return from COSI with %f: ",newval);
    wait();
#else
    printf(" Enter a value: ");
    scanf("%f",&newval);CLEARBUF
#endif
    return(newval);
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -  Function    :  intern_constructor                                        -
 -  Input       :  none                                                      -
 -  Output      :  none                                                      -
 -  Action      :  reads questions sets up list of questions                 -
 -  Date        :  22 Feb 90                                                 -
 -  UpDate      :                                                            -
```

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/*
      Principle storage of facts
*/
fact  Intern_fact[NUMQUEST];

void  intern_constructor()
{
      extern      fact  Intern_fact[NUMQUEST];
      fact* this_intern;
      int   i;

      printf("\nInitialise the inernal FM...");
      /* initialise the structure to undefined */
      for(i=0;i<NUMQUEST;i++) {
            (this_intern=&Intern_fact[i])->state = UNDEFINED;
            this_intern->value = INITVAL;
      }/*for ever*/
} /*end intern_constructor()*/

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   intern_getstate  intern_getvalu ..gettoken    -
 -    Input       :   intern fact ID  starting at  1                -
 -    Cutput      :   state of the fact                             -
 -    Action      :   looks up the fact and returns it state        -
 -    Date        :   22 Feb 90                                     -
 -    UpDate      :                                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
unsigned    intern_getstate(ext_index)
unsigned    ext_index;
{
      extern      fact  Intern_fact[NUMQUEST];
      if (--ext_index >= NUMQUEST) terror("Fact ID out of index state");
      return(Intern_fact[ext_index].state);
}

unsigned    intern_gettoken(ext_index)
unsigned    ext_index;
{
      return(ext_index);
}

float intern_getvalue(ext_index)
unsigned    ext_index;
{
      extern      fact  Intern_fact[NUMQUEST];
      fact* this_fact;
      if (ext_index > NUMQUEST) terror("Fact ID out of index int-val ");
      if ((this_fact=&Intern_fact[ext_index-1])->state == UNDEFINED) {
            this_fact->value = ask_quest(ext_index);
            this_fact->state = KNOWN;
      }
      return(this_fact->value);
}

#ifdef            LINKZEN
#else
void  wait()
{
      printf("\nPress Return to Continue...");
      getchar();
}
```

```
#endif

/*
end file fm.c */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell                               =
   =    Sub-Project   :   Human interface                           =
   =    File          :   hi.c                                      =
   =    Author        :   Robert D. Rourke                          =
   =    Start Date    :   22 Feb 1990                               =
   =    Update        :   28 Feb 1990                               =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
/*#define   TESTON*/
/*#define   RMXON*/

#include    <stdio.h>
/*#include  <conio.h> not suported in rmx*/
#include    <math.h>
#include    <ctype.h>
#include    "huminter.h"

#include    "hi.h"


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   main                                       -
 -    Input       :   none                                       -
 -    Output      :   none                                       -
 -    Action      :   Test the reading the printing of file      -
 -    Date        :   22 Feb 90                                  -
 -    UpDate      :                                              -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
main()
{
      test_hiread();
}
*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   test_hiread                                -
 -    Input       :   none                                       -
 -    Output      :   none                                       -
 -    Action      :   Test the human intreface                   -
 -    Date        :   22 Feb 90                                  -
 -    UpDate      :                                              -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void  test_hiread()

{
      int i;
      float response;
      char dummy;

      hi_constructor();
      /* test verification for the numbers */
/*    for (i=0;i<NUMNUMR;i++){
            response = ask_numr(&Numr_list[i]);
            printf ("\n--->The answer to %u is %f ",i,response);
```

```
      }
*/

 /*   printf("\ntesting the text");
      for (i=0;i<NUMTEXT;i++){
            response = ask_text(&Text_list[i]);
            printf ("\n--->The answer to %u is %f",i,response);
      }
 */
      for (;;){
            printf ("\n\nEnter a question ID: ");
            scanf("%d%c",&i,&dummy);
            response = ask_quest(i);
            printf ("\n--->The answer to %u is %f ",i,response);
      }

      exit(0);
} /*end test_hiread*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :    hi_constructor                                -
 -    Input       :    none                                          -
 -    Output      :    none                                          -
 -    Action      :    reads questions sets up list of questions     -
 -    Date        :    22 Feb 90                                     -
 -    UpDate      :                                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
      Principle storage of symbolic question information
*/
/*    the list of all questions, ids the sub list */
questref    Quest_list[NUMQUEST];
/*    The list of all numerical-based questions */
numrquest   Numr_list[NUMNUMR];
/*    The list of all text-based questions */
textquest   Text_list[NUMTEXT];
/*    Date stamp of the input text file */
char  Date_stamp[DATESIZE];

void  hi_constructor()
{
      extern      questref    Quest_list[NUMQUEST];
      extern      numrquest   Numr_list[NUMNUMR];
      extern      textquest   Text_list[NUMTEXT];
      extern      char        Date_stamp[DATESIZE];

      numrquest*  this_numr;
      textquest*  this_text;
      questref*         this_quest;
      char              quest_type[TYPESIZE];
      char              dummy[Q_SIZE+1];
      unsigned          num_ans, i, curt_quest, curt_ans;
      unsigned          curt_text, curt_numr;
      float       timestamp;
      FILE*       datafile;

      /* open the input data file terminate if error */
      if ((datafile = fopen(DATAFILE, "r"))==NULL)
           terror("Missing the main data file");

      /* remove the header store date stamp */
```

```
      printf("\nLoading the questions...");
      for (i=1;i<DATELOC;i++) fscanf(datafile, "%s",dummy);
      fscanf(datafile, "%s",Date_stamp);
      printf ("\nThe date stamp: %s\n",Date_stamp);

      /* load the questions */
      curt_text = curt_numr = 0;
      for(this_quest=Quest_list;;this_quest++) {
          printf(".");
            if (fscanf(datafile, "%s",quest_type)==EOF) break;
/*          printf("\n type: %s",quest_type);*/
            fscanf(datafile, "%f",&timestamp);
/*          printf("the float %f %f",0.34556,timestamp); */
            if (quest_type[0]==TEXTID) {
                  /* then it is a text-based question */
                  this_text = &Text_list[curt_text++];
                  /* clear 1 leading space */
                  fgets(dummy,CLSPACE,datafile);
                  fgets((this_text->question),Q_SIZE,datafile);
/*                printf("\n Question:%s",(this_text->question));*/
                  fscanf(datafile, "%d",&num_ans);
/*                printf(" %u ",num_ans); */
                  for (curt_ans=0;curt_ans<num_ans;curt_ans++) {
                        /* clear 1 leading space */
                        fgets(dummy,CLSPACE,datafile);

fgets((this_text->answer[curt_ans]),A_SIZE,datafile);
/*                      printf("\n sns:%s",(this_text->answer[curt_ans]));
*/
                  }
                  this_text->num_ans = num_ans;
                  this_quest->quest_type=TEXTID;
                  this_quest->index = (char*)this_text;
            }
            else {
                  /* then it is numeric */
                  this_numr = &Numr_list[curt_numr++];
                  fgets(dummy,CLSPACE,datafile);
                  fgets((this_numr->question),Q_SIZE,datafile);
/*                printf("\n Question:%s\n",(this_numr->question));*/
                  fscanf(datafile, "%f",(&(this_numr->upper)));
/*                printf("upper  %.2f  ",this_numr->upper); */
                  fscanf(datafile, "%f",(&(this_numr->lower)));
/*                printf("lower  %f.2",this_numr->lower); */
                  this_quest->quest_type=NUMRID;
                  this_quest->index = (char*)this_numr;
            }
/*    printf("\n"); */
      }/*for ever*/
fclose(datafile);
      printf("\nData loaded");
} /*end hi_constructor()*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   ask_quest                                     -
 -    Input       :   question ID index                             -
 -    Output      :   response                                      -
 -    Action      :   ask the user a num or text question           -
 -    Date        :   01 Mar 90                                     -
 -    UpDate      :      Mar 90                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
float ask_quest(quest_id)
```

```
unsigned    quest_id;
{
      extern      questref    Quest_list[NUMQUEST];
      questref* this_quest;
      float respons;

      /* A small error check */
      if (quest_id > NUMQUEST) terror("Question ID out of index");
      if ((this_quest=&Quest_list[--quest_id])->quest_type==TEXTID)
           return(ask_text( (textquest*)(this_quest->index) ) );
      else
           return(ask_numr((numrquest*)(this_quest->index)));
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   ask_numr                                  -
 -    Input       :   pointer to a numr question data           -
 -    Output      :   response                                  -
 -    Action      :   ask the user on question                  -
 -    Date        :   26 Feb 90                                 -
 -    UpDate      :      Feb 90                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
float ask_numr(this_numr)
numrquest*  this_numr;

{
      float respons;

      /* do a little error check on the question */
      if (strlen(this_numr->question)<1) terror ("ask_numt");

      /* set up the dispay */
      printf("\n");
      for (;;){
            printf ("\nQuestion:");
            printf("\%s",(this_numr->question));
            printf("\nThe range is %.2f to %.2f :",
             this_numr->lower, this_numr->upper);
            scanf("%f",&respons);
            /* remove the CR character. */
            getchar();
            if ((respons<=this_numr->upper) &&
(respons>=this_numr->lower))
                  break;
            printf("\a Wrong!!!\n");
      }
      return(respons);

}/*end funcion ask_numr */


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   ask_text                                  -
 -    Input       :   pointer to a text question data           -
 -    Output      :   response                                  -
 -    Action      :   ask the user on question                  -
 -    Date        :   28 Feb 90                                 -
 -    UpDate      :      Feb 90                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
float ask_text(this_text)
textquest*  this_text;
```

```
{
       int i;
       char resp;
       int respi;

       /* do a little error check on the question */
       if (strlen(this_text->question)<1) terror ("ask_numt");

       /* set up the dispay */
       printf("\n");
       printf ("\nQuestion: ");
       printf("\%s",(this_text->question));
       for (i=0;i<this_text->num_ans;i++)
             printf ("\n%c- %s",i+'A',this_text->answer[i]);
       printf ("\nSelect an answer :");
       for (;;) {
/*           resp = getch(); */
             resp = getchar();
             getchar();
/*           fread (&resp, sizeof(resp),1,stdin);*/
             if (resp<='Z') respi = resp-'A';
             else  respi = resp-'a';
             if ( respi >= 0 && respi < this_text->num_ans) {
/*                  putch(toupper(resp));*/
                    putchar(toupper(resp));
                    break;
             }
             printf("\a");
    }
       return((float)respi+1);
}/*end funcion ask_text */


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -     Function    :   terror                                      -
 -     Input       :   none                                        -
 -     Output      :   none                                        -
 -     Action      :   Test the reading the printing of file       -
 -     Date        :   22 Feb 90                                   -
 -     UpDate      :                                               -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void  terror(Message)
char* Message;
{
      fprintf (stderr,"\aCritical terminating error: ");
      fprintf (stderr,Message);
      fprintf (stderr,"\n");
      exit(1);
}


/*
end file hi.c */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project      :   KOOLA shell                               =
   =    Sub-Project  :   Human interface                           =
   =    File         :   test_fm.c                                 =
   =    Author       :   Robert D. Rourke                          =
   =    Start Date   :   01 Mar 1990                               =
```

```
  =    Update       :   28 Mar 1990                              =
  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
#define           TESTON
/*#define    RMXON*/

#include    <stdio.h>
#include    <ctype.h>
#include    "extreq.h"
#include    "huminter.h"
#include    "hi.hx"
#include    "fm.hx"

void  test_fm(void);


#define           CLEARBUF    getchar();


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :   main                                       -
 -    Input        :   none                                       -
 -    Output       :   none                                       -
 -    Action       :   Test the reading the printing of file      -
 -    Date         :   22 Feb 90                                  -
 -    UpDate       :                                              -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
main()
{
      test_fm();
      exit(0);
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :   test_fm                                    -
 -    Input        :   none                                       -
 -    Output       :   none                                       -
 -    Action       :   Test the human intreface                   -
 -    Date         :   22 Feb 90                                  -
 -    UpDate       :                                              -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void  test_fm()

{
      int i,k;
      float response;
      hi_constructor();

      extern_constructor();
      intern_constructor();
      wait();
      for (i=1;i<=NUMQUEST;i++) {
            printf("\n %d The state %d ",i,intern_getstate(i));
      }
      wait();
      for (i=1;i<=NUMEXRQ;i++)
            printf("\nEX state %d ",extern_getstate(i));
      wait();

      for (k=0;k<5;k++){
            printf ("\n\nEnter a high level fact ID: ");
            scanf("%d",&i);CLEARBUF
```

```
        response = fact_value(i);
        printf ("\n--->The answer to %u is %f ",i,response);
        wait();
    }

    for (i=1;i<=NUMQUEST;i++)
        printf("\n state %d ",intern_getstate(i));
    wait();

    for (i=1;i<=NUMEXRQ;i++)
        printf("\nEX state %d ",extern_getstate(i));

    printf("\n Test Complete\n");
} /*end test_fm*/


 /*= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 =    Project         :   KOOLA programing language                  =
 =    Sub-project     :   Include file for external requests         =
 =    Language        :   C-286 for intel RMX operating System       =
 =    File            :   extreq.h                                   =
 =    Date            :   02/05/1990                                 =
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

    Warning:   do not make any changes to this file because it can be
               automaticaly update by the KOOLA compiler
*/
#ifndef     _EXTRQINTER
#define     _EXTRQINTER       1
/*
    Some maximum values used to compile the interface to COSI:
*/
#define     NUMEXRQ                    3     /* total number of questions
defined */

#endif

/* end file huminter.h */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell                                =
   =    Sub-Project   :   Human interface                            =
   =    File          :   hi.h                                       =
   =    Author        :   Robert D. Rourke                           =
   =    Start Date    :   22 Feb 1990                                =
   =    Update        :   28 Mar 1990                                =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#define     DATAFILE    "huminter.dat"
#define     Q_SIZE            61
#define     A_SIZE            31
#define     CLSPACE           2
#define     DATESIZE    15
#define     TYPESIZE    4

#define     DATELOC           9
#define     TEXTID            'T'
#define     NUMRID            'N'


/*    list of all questions */
```

```
typedef struct
{
      unsigned quest_type;
      char* index;
} questref;

/*    a numeric-based question node */
typedef struct
{
      char question[Q_SIZE];
      float  upper, lower;
} numrquest;

/*    a text-based question node */
typedef struct
{
      char question[Q_SIZE];
      char num_ans;
      char answer[MAXANSWERS][A_SIZE];
} textquest;


#ifdef RMXON
void        test_hiread();
void        terror();
void        hi_constructor();
float       ask_numr();
float       ask_text();
float       ask_quest();

#else
void        test_hiread(void);
void        terror(char Error_message[]);
void        hi_constructor(void);
float       ask_numr(numrquest* one_numeric_struc);
float       ask_text(textquest* one_text_struc);
float       ask_quest(unsigned quest_id);
#endif

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell
   =    Sub-Project   :   Fact Manager of the Belief Manager
   =    File          :   fm.hx
   =    Author        :   Robert D. Rourke
   =    Start Date    :   06 Mar 1990
   =    Update        :   28 Mar 1990
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
Description:
      This file defines the "external" members of the fact manager

*/
#ifdef RMXON
void        extern_constructor();
void        intern_constructor();
float       fact_value();

#else
void        extern_constructor(void);
void        intern_constructor(void);
float       fact_value(unsigned highlevel_factid);
#endif
/*
```

```
end file fm.hx */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell                                 =
   =    Sub-Project   :   Human interface                             =
   =    File          :   hi.hx                                       =
   =    Author        :   Robert D. Rourke                            =
   =    Start Date    :   22 Feb 1990                                 =
   =    Update        :   28 Feb 1990                                 =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
        Access procedure for other tasks to use the hi
*/

#ifdef RMXON
float       ask_quest();
void        hi_constructor();
#else
float       ask_quest(unsigned quest_id);
void        hi_constructor(void);
#endif


/* = = = = = = = = = = = = = = = = = = = = = = =   = = = = = = = = = =
   =    Project       :   KOOLA shell                                 =
   =    Sub-Project   :   Human interface                             =
   =    File          :   fm.h                                        =
   =    Author        :   Robert D. Rourke                            =
   =    Start Date    :   02 Mar 1990                                 =
   =    Update        :   05 Mar 1990                                 =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#define     FOREVER          0xffff
#define     DATAFILE    "extreq.dat"
#define     CLSPACE          2
#define     DATESIZE    40
#define     TYPESIZE    4

#define     UNDEFINED   -1
#define     UNAVAILAB   1
#define     KNOWN       2
#define     INITVAL          -33.0
#define     EXCESS      500

#define     DATELOC          9


/*    list of any fact */
typedef struct
{
      unsigned cositoken;
      unsigned state;
      float value;
} fact;


#ifdef RMXON

unsigned    intern_getstate();
unsigned    intern_gettoken();
float       intern_getvalue();
```

```
void        test_fm();
void        terror();
void        wait();

unsigned    extern_getstate();
unsigned    extern_gettoken();
float       extern_getvalue();
float       rqst_ext();

#else

unsigned    intern_getstate(unsigned int_index);
unsigned    intern_gettoken(unsigned int_index);
float       intern_getvalue(unsigned int_index);

void        test_fm(void);
void        wait(void);
void        terror(char* Messagestr);

unsigned    extern_getstate(unsigned ext_index);
unsigned    extern_gettoken(unsigned ext_index);
float       extern_getvalue(unsigned ext_index);
float       rqst_ext(unsigned fact_id);

#endif

/*
end file fm.h */

 /*= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -
 =    Project          :   KOOLA programing language
 =    Sub-project      :   Include file for external requests
 =    Language         :   C-286 for intel RMX operating System       -
 =    File             :   extreq.h                                   -
 =    Date             :   02/05/1990                                 =
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

     Warning:    do not make any changes to this file because it can be
                 automaticaly update by the KOOLA compiler
*/
#ifndef     _EXTRQINTER
#define     _EXTRQINTER      1
/*
     Some maximum values used to compile the interface to COSI:
*/
#define     NUMEXRQ                   3     /* total number of questions
defined */

#endif

/* end file huminter.h */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = : -
   =    Project       :   KOOLA shell                                  -
   =    Sub-Project   :   Fact Manager of the Belief Manager
   =    File          :   fm.hx                                        -
   =    Author        :   Robert D. Rourke                             -
   =    Start Date    :   06 Mar 1990                                  =
   =    Update        :   28 Mar 1990                                  -
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -
Description:
     This file defines the "external" members of the fact manager.
```

```
*/
#ifdef RMXON
void          extern_constructor();
void          intern_constructor();
float         fact_value();

#else
void          extern_constructor(void);
void          intern_constructor(void);
float         fact_value(unsigned highlevel_factid);
#endif
/*
end file fm.hx */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =     Project        :   KOOLA shell                                =
   =     Sub-Project    :   Human interface                            =
   =     File           :   pie.h                                      =
   =     Author         :   Robert D. Rourke                           =
   =     Start Date     :   22 Feb 1990                                =
   =     Update         :   23 Mar 1990                                =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#define       DATAFILE    "prknbase.dat"
#define       MAXFACT           20
#define       CLSPACE           2
#define       DATESIZE    15
#define       TYPESIZE    4

#define       DATELOC           9

#define       EXCESS            500

#define       THENRULE    1     /* possible rule senarios */
#define       ELSERULE    2
#define       CONTINUED   3
#define       ENDRULE           0x10
#define       MASKEND           3

#define       DEFINED           1     /* states */
#define       UNDEFINED         2
#define       UNAVAILABLE 3

#define INVALIDPR       -333

/*
 *    One component of the primary knowledge base
 */

typedef struct
{
      char        contex;     /* then or else clause */
      int         fact;
      char        prob;
      char        weight;
      char        opratr;
      float oprand;
} pr; /*primary rule*/

typedef struct
{
```

```
     unsigned      num_fact;
     pr            rule[MAXFACT];
     float         dyprob;              /*    dynamical    determined
p=dprob/10,000 */
     char          state;
} prmblf;


#ifdef RMXON
int          solv_brule();
int          chk_logic ();
#else
int          solv_brule(pr* startof_rule);
int          chk_logic (float* operand1, int operator, float* operand2);
#endif


 /*= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -
 =    Project          :   KOOLA programing language
 =    Sub-project      :   Include file for primary inference eng
 =    Language         :   C-286 for intel RMX operating System     -
 =    File             :   knowbase.h
 =    Date             :   02/05/1990                               -
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = . -

     Warning:    do not make any changes to this file because it can be
                 automaticaly update by the KOOLA compiler
*/
#ifndef      _PRIMARYINF
#define      _PRIMARYINF
/*
     Some maximum values used by the RMX C compiler for the inference
engines:
*/
#define      NUMPRMBLF                12     /*  total   number   of  primary
beliefs */
#define      NUMSECBLF                9      /* total   number   of  secondary
beliefs */
#define      NUMGOAL                  10     /* total number of goals */

#endif

/* end file knowbase.h */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -
   =    Project       :   KOOLA shell                                -
   =    Sub-Project   :   Human interface                            -
   =    File          :   pie.c                                      -
   =    Author        :   Robert D. Rourke
   =    Start Date    :   06 Mar 1990
   =    Update        :   23 Mar 1990
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -
*/
/*#define    TESTON*/
#define      RMXON

#include     <stdio.h>
#include     <ctype.h>
#include     "knowbase.h"
#include     "fm.hx"
#include     "pie.h"
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :    find_pbelief                                  -
 -    Input       :    none                                          -
 -    Output      :    none                                          -
 -    Action      :    Test the reading the printing of file         -
 -    Date        :    22 Feb 90                                     -
 -    UpDate      :                                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
float        find_pbelief(pr_belief)
int          pr_belief;
{
        extern          prmblf      Prm_btree[NUMPRMBLF];
        prmblf*      this_blf;
        int          cur_fact;
        pr*          this_fact;
        int          end_fact;
        int          solved_flg;
        float ac_prob, ac_weight;


        /* convert from a primary belief ID into a pointer to that belief */
        if (pr_belief>NUMPRMBLF||pr_belief<0)
                terror ("Primary belief ID out of rang");
        this_blf = &Prm_btree[pr_belief-1];


        solved_flg = 0;
        ac_prob = 0.0;
        ac_weight = 0.0;
        end_fact = this_blf->num_fact;

        /* find out if the belief is known */
        if (this_blf->state==UNAVAILABLE)  return (INVALIDPR);
        if (this_blf->state==DEFINED)      return (this_blf->dyprob);

        /* the belief must be UNDEFINED, this routin will try to define it
*/
        for (cur_fact=0;cur_fact<end_fact;cur_fact++) {
/*              printf("\n\nChecking the next rule:");*/
                if (solv_brule(this_fact=&(this_blf->rule[cur_fact]))) {
/*                      printf("\n\nthe outcome is true assign p %i w %i ",
                         this_fact->prob,this_fact->weight);
 */
                        ac_prob +=
(float)(this_fact->prob)*(float)(this_fact->weight)/100.0;
                        ac_weight += this_fact->weight;
/*                      printf  ("\nThe current value of prob and wight %f
%f",ac_prob,ac_weight);*/
                        solved_flg++;
                }
/*              else printf ("\nthe outcome false nothing assigned");*/
/*              printf  ("\nThe  current  value  of  prob  and  wight  %f
%f",ac_prob,ac_weight);*/
                /* make sure we are at the end */
                while (this_blf->rule[cur_fact].contex<ENDRULE) cur_fact++;
        }
/*      printf   ("\nThe   current   value   of   prob   and   wight   %f
%f",ac_prob,ac_weight);*/
        if (solved_flg) {
/*              printf("\nBelief solved:");*/
                this_blf->dyprob=100*ac_prob/ac_weight;
/*              printf("\nthe output probability is: %f ",this_blf->dyprob);*/
```

```
        this_blf->state = DEFINED;
        return (this_blf->dyprob);
    }
    else {
        this_blf->state = UNAVAILABLE;
/*      printf("\nthe belief is not available");*/
        this_blf->dyprob=INVALIDPR;
        return (INVALIDPR);
    }

} /*find_pbelief*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -  Function   :  solv_brule                                              -
 -  Input      :  pointer to the start of rule                            -
 -  Output     :  outcome of the the antecedent                           -
 -  Action     :  decides if the facts suport the anteedent               -
 -  Date       :  07 Mar 90                                               -
 -  UpDate     :                                                          -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*  set the current fact to the first
    do until the current fact-cluase is false or the rule ends
        compaire the current fact to see is true
        if found false terminate with false
        set the current fact to the next
        if the next fact is not part of this rule terminat true
 */
int     solv_brule(this_fact)
pr*     this_fact;

{
    int         fact_id;
    float fact_val;
    char  operator;
    float number;
    int         logics;

    logics = (this_fact->contex & MASKEND)==THENRULE;
/*  printf("\nthe logic is based on 0=else 1=then %i", logics);*/
    for (;;this_fact++) {
        fact_id = this_fact->fact;
        operator = this_fact->opratr;
        number = this_fact->oprand;
        fact_val = fact_value(fact_id);
#ifdef TESTON
        printf("\nThe fact we are looking for is:%i ",fact_id);
        printf("\nthe operator is %i", operator);
        printf("\nthe number  is %f", number);
        printf("\nThe fact value is:%f ",fact_val);
#endif
        /* test for one false fact that can terminate the rule */
        if (!chk_logic(&fact_val,operator,&number)) return(!logics);
        /* if all facts or possitive, then the rule is possitive */
        if (this_fact->contex > ENDRULE) return (logics);
    }
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -  Function   :  chk_logic                                               -
 -  Input      :  operand1 ?  operatnd22 ==                               -
```

```
 -    Output      :   logical outcome                                -
 -    Action      :   decides if the facts suport the anteedent      -
 -    Date        :   07 Mar 90                                      -
 -    UpDate      :                                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
int         chk_logic (float operand1, int operator, float operand2)
*/
int         chk_logic (operand1, operator, operand2)
float*      operand1;
int         operator;
float*      operand2;
{
/*    printf  (" The  operand  is  (at  the  chk_logic:  %i,  %f,
%f",operator,*operand1,*operand2);*/
      switch (operator) {
            case 1:
                  if (*operand1==*operand2) return (1);
                  break;
            case 2:
                  if (*operand1<*operand2) return (1);
                  break;
            case 3:
                  if (*operand1>*operand2) return (1);
                  break;
            case 4:
                  if (*operand1<=*operand2) return (1);
                  break;
            case 5:
                  if (*operand1>=*operand2) return (1);
                  break;
            case 6:
                  if (*operand1!=*operand2) return (1);
                  break;
            default:
                  terror("unknown compar operator");
      }
/*    printf ("fell through");*/
      return (0);
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   pie_dump                                 -
 -    Input       :   none                                           -
 -    Output      :   none                                           -
 -    Action      :   reads the primary knowledge base form file     -
 -    Date        :   06 Mar 90                                      -
 -    UpDate      :                                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void  pie_dump()

{
      extern            prmblf       Prm_btree[NUMPRMBLF];
      unsigned    numfacts;
      unsigned    curtfact;
      int               j;


      for(j=0;j<NUMPRMBLF;j++) {
            numfacts=Prm_btree[j].num_fact;
            for (curtfact=0;curtfact<numfacts;curtfact++)
```

```
            printf("  R: %i", Prm_btree[j].rule[curtfact].contex);

        wait();

        /* enter the facts directly */
        for (curtfact=0;curtfact<numfacts;curtfact++)
            printf("  F: %i", Prm_btree[j].rule[curtfact].fact);

        wait();

        /* enter the probability in it is % form */
        for (curtfact=0;curtfact<numfacts;curtfact++)
            printf("  P: %i", Prm_btree[j].rule[curtfact].prob);
        wait();

        /* enter the weight, it is in % form */

        for (curtfact=0;curtfact<numfacts;curtfact++)
            printf("  W: %i", Prm_btree[j].rule[curtfact].weight);
        wait();



        /* enter the operator  */
        for (curtfact=0;curtfact<numfacts;curtfact++)
            printf(" Od: %i", Prm_btree[j].rule[curtfact].oprator);
        wait();

        /* enter the number (float) it is compaired to */
        for (curtfact=0;curtfact<numfacts;curtfact++)
            printf("  O: %f", Prm_btree[j].rule[curtfact].oprand);
        wait();

    } /*until EOF */
} /*end pie_dump()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -   Function   :  pie_constructor                                   -
 -   Input      :  none                                              -
 -   Output     :  none                                              -
 -   Action     :  reads the primary knowledge base form file        -
 -   Date       :  06 Mar 90                                         -
 -   UpDate     :  08 Mar 90 format of rule contex                   -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
    Principle storage of symbolic question information
*/
/*  the list of all bliefs forming a tree */
prmblf      Prm_btree[NUMPRMBLF];

void  pie_constructor()

{
    extern          prmblf      Prm_btree[NUMPRMBLF];
    char        date_stamp[DATESIZE+56];
    char            dummy[80];
    unsigned    numfacts;
    unsigned    curtfact;
    int             i;
    int             input;
```

```
    int             prvrule, newrule;
    prmblf*         this_blf;
    FILE*           datafile;


    hi_constructor();
    extern_constructor();
    intern_constructor();
    /* open the input data file terminate if error */
    if ((datafile = fopen(DATAFILE, "r"))==NULL)
       terror("Missing the primary KB file");

    /* remove the header store date stamp */
/*  printf("\nLoading the primary knowledge base...");*/
    for (i=1;i<DATELOC;i++) fscanf(datafile, "%s",dummy);
    fscanf(datafile, "      %s ",date_stamp);
    printf ("\nPrimary KB date stamp: %s\n",date_stamp);

    /* load the questions */
    for(this_blf=Prm_btree;;this_blf++) {
       printf("...");

        /* read the number of suporting facts and save this value */
        if (fscanf(datafile, "%d",&numfacts)==EOF) break;
/*      printf("Number of facts: %i",numfacts);*/
        this_blf->num_fact = numfacts;
        this_blf->state  = UNDEFINED;
        this_blf->dyprob = (float) INVALIDPR;


        /* enter the first rule ids */
        fscanf(datafile, "%d",&newrule);
/*      printf("\nFirst rule id: %i",newrule);*/
        if ((prvrule=newrule)>EXCESS)
this_blf->rule[0].contex=ELSERULE;
        else this_blf->rule[0].contex=THENRULE;

        /* enter the rest using the prv rule to chain */
        for (curtfact=1;curtfact<numfacts;curtfact++) {
/*          fscanf(datafile, "%i",&newrule);*/
            fscanf(datafile, "%d",&newrule);
/*          printf("\nRule id: %i",newrule); */
            if (newrule>EXCESS)
this_blf->rule[curtfact].contex=ELSERULE;
            else this_blf->rule[curtfact].contex=THENRULE;

            if (prvrule!=newrule) {
                /*indicate that the previous fact was the last of
the rule */
                this_blf->rule[(curtfact-1)].contex += ENDRULE;
            }
            prvrule=newrule;
        }
        this_blf->rule[(curtfact-1)].contex += ENDRULE;

/*      wait();*/

        /* enter the facts directly */
        for (curtfact=0;curtfact<numfacts;curtfact++) {
            fscanf(datafile, "%d",&(this_blf->rule[curtfact].fact));
/*          printf("\nFact id: %i",this_blf->rule[curtfact].fact);*/
        }
```

```
/*          wait();*/

            /* enter the probability in it is % form */
            for (curtfact=0;curtfact<numfacts;curtfact++) {
                 fscanf(datafile, "%d",&input);
                 this_blf->rule[curtfact].prob=(char)input;
/*               printf("\nProb id: %d",this_blf->rule[curtfact].prob);*/
            }

/*          wait();*/

            /* enter the weight, it is in % form */
            for (curtfact=0;curtfact<numfacts;curtfact++) {
                 fscanf(datafile, "%d",&input);
                 this_blf->rule[curtfact].weight=(char)input;
/*               printf("\nWeight id:
%d",this_blf->rule[curtfact].weight);*/
            }
/*          wait();*/

            /* enter the operator  */
            for (curtfact=0;curtfact<numfacts;curtfact++) {
                 fscanf(datafile, "%d",&input);
                 this_blf->rule[curtfact].opratr=(char)input;
/*               printf("\nOopratr id:
%i",this_blf->rule[curtfact].opratr);*/
            }
/*          wait();*/

            /* enter the number (float) it is compaired to */
            for (curtfact=0;curtfact<numfacts;curtfact++) {
                 fscanf(datafile,
"%f",&(this_blf->rule[curtfact].oprand));
/*               printf("\nOopratr id:
%f",this_blf->rule[curtfact].oprand);*/
            }
/*          wait();*/

      } /*until EOF */
      fclose(datafile);
      printf("\nPrim Knowledge loaded");
} /*end pie_constructor()*/



/*
end file pie.c */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell                               =
   =    Sub-Project   :   primry infernce                           =
   =    File          :   test_pie.c                                =
   =    Author        :   Robert D. Rourke                          =
   =    Start Date    :   23 Mar 1990                               =
   =    Update        :   23 Mar 1990                               =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
/*#define        TESTON*/
#define     RMXON

#include    <stdio.h>
```

```
#include    <ctype.h>
#include    "pie.hx"


void  pie_test();
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -     Function    :   main                                          -
 -     Input       :   none                                          -
 -     Output      :   none                                          -
 -     Action      :   Test the reading the printing of file         -
 -     Date        :   23 Mar 90                                     -
 -     UpDate      :   23 Mar                                        -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
main()
{
      pie_constructor();
      wait();
      pie_test();
      pie_dump();
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -     Function    :   pie_test                                      -
 -     Input       :   none                                          -
 -     Output      :   none                                          -
 -     Action      :   Test the reading the printing of file         -
 -     Date        :   22 Feb 90                                     -
 -     UpDate      :   23 Mar 90                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void  pie_test()
{
      int         belief;
      float       value;

      for (;;) {
            printf("\n\nEnter a primary belief, (0=exit): ");
            scanf ("%d",&belief);
            getchar();
            if (belief==0) break;
            value = find_pbelief(belief);
            printf ("\n\n Value calculated is : %f",value);
      }
      printf ("\nTest ended.");
}


 /*= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
 =     Project          :   KOOLA programing language                =
 =     Sub-project      :   Include file for external requests       =
 =     Language         :   C-286 for intel RMX operating System     =
 =     File             :   extreq.h                                 =
 =     Date             :   17/04/1990                               =
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

      Warning:    do not make any changes to this file because it can be
                  automaticaly update by the KOOLA compiler
*/
#ifndef     _EXTRQINTER
#define     _EXTRQINTER       1
/*
      Some maximum values used to compile the interface to COSI:
*/
```

```
#define     NUMEXRQ                   3      /* total number of questions
defined */

#endif

/* end file huminter.h */
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =     Project       :   KOOLA shell                                =
   =     Sub-Project   :   Human interface                            =
   =     File          :   pie.hx                                     =
   =     Author        :   Robert D. Rourke                           =
   =     Start Date    :   23 Mar 1990                                =
   =     Update        :   23 Mar 1990                                =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#ifdef RMXON
void        pie_constructor();
void        pie_dump();
float find_pbelief();
#else
void        pie_constructor(void);
void        pie_dump(void);
float find_pbelief(int belief_id);
#endif


/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =     Project       :   KOOLA shell                                =
   =     Sub-Project   :   Secondary Inference Engine                 =
   =     File          :   queue.hx                                   =
   =     Author        :   Robert D. Rourke                           =
   =     Start Date    :   25 Mar 1990                                =
   =     Update        :   25 Mar 1990                                =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#ifdef      RMXON
int         qupop();
int         quexam();
void        qupush();
void        qu_constructor();
int         quempt();
#else
int         qupop(void);
int         quexam(void);
void        qupush(int);
void        qu_constructor(void);
int         quempt(void);
#endif
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =     Project       :   KOOLA shell                                =
   =     Sub-Project   :   Secondary Inference Engine                 =
   =     File          :   sie.c                                      =
   =     Author        :   Robert D. Rourke                           =
   =     Start Date    :   23 Mar 1990                                =
   =     Update        :   30 Mar 1990                                =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
/*#define   TESTON*/
/*#define   RMXON*/
```

```
#include    <stdio.h>
#include    <ctype.h>
#include    "knowbase.h"
#include    "pie.hx"
#include    "queue.hx"
#include    "sie.h"

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   find_sbelief                                  -
 -    Input       :   none                                          -
 -    Output      :   none                                          -
 -    Action      :   reads the primary knowledge base form file    -
 -    Date        :   26 Mar 90                                     -
 -    UpDate      :                                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
float find_sbelief(goal)
int   goal;
{
      extern      secblf      Sec_btree[NUMSECBLF];
      int         curt_blf;

      if (goal > NUMSECBLF)
            terror ("Goal belief out of range");

      /* creat the stack and put the goal on it */
      qu_constructor();
      qupush(goal--);
      /* main loop till goal is solved */
      for (;;) {
#ifdef TESTON
            printf ("\n\nTop of sbelief: ");
#endif
            /* examine the next belief if it is goal and defined success
*/
            curt_blf = quexam()-1;
#ifdef TESTON
            printf ("\n Now pulled %d",curt_blf+1);
            printf (" it has a state of: %d",Sec_btree[curt_blf].state);
#endif
            if (curt_blf==goal)     {
                  if (Sec_btree[curt_blf].state==DEFINED)    {
                        /* success */
#ifdef TESTON
                        printf ("success");
#endif
                        return (Sec_btree[curt_blf].dyprob);
                  }
                  if (Sec_btree[curt_blf].state==UNAVAILABLE)      {
                        /* failure */
#ifdef TESTON
                        printf ("failure");
#endif
                        return (INVALIDPR);
                  }
            } /*end if goal*/

            switch (Sec_btree[curt_blf].state) {
                  case DEFINED:
                  case UNAVAILABLE:
                        qupop();
                        break;
                  case UNDEFINED:
```

```
                        sove_suport(curt_blf);
                        break;
                default:
                        terror ("Unkwon state");
                        break;
            }

      }/* repeat forever */
}/*end function find_sbelief*/



/*
float find_pbelief();
 */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :   sove_suport                                   -
 -    Input        :   target belief
 -    Output       :   none                                          -
 -    Action       :   tryes to solve the target belief
 -    Date         :   27 Mar 90                                     -
 -    UpDate       :                                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
binf        Suports[MAXBELF];
/*#define   TESTSOLV*/

void  sove_suport(targ_blf)
int targ_blf;
{
      extern      binf   Suports[MAXBELF];
      extern      secblf       Sec_btree[NUMSECBLF];
      int         curt_supt;
      int         solvable,available,enough;
      sr*         this_sup;
      binf* this_solv;
      float ac_prob, ac_weight;
      float prob, weight;
      float minprob;

      solvable = 1;           /* assume there will be enough info t[illegible]
*/
      /*
       *    step 1 load all suporting beliefs into the suport array
       */
#ifdef TESTON
      printf    ("\nStart    of    sove,    Number    of    support
%d",Sec_btree[targ_blf].num_belf);
#endif
      for (curt_supt=0;curt_supt<Sec_btree[targ_blf].num_belf;curt_[illegible]
{
            this_sup   = &(Sec_btree[targ_blf].rule[curt_supt]);
            this_solv = &Suports[curt_supt];
#ifdef      TESTSOLV
            printf  ("\nAt  the  top  of  the  loop  in  solve  [illegible]
blef=%d",this_sup->belf);
#endif
            if (this_sup->belf>EXCESS) {
#ifdef TESTON
                  printf ("\nThe bel_ef was found primary");
#endif
                  this_solv->dyprob = find_pbelief(this_sup->belf-EXCESS);
            }
```

```
        else {
#ifdef TESTON
            printf ("\nThe belief was second now testing its state:
%d",find_state(this_sup->belf));
#endif
            switch (find_state(this_sup->belf)) {
                case UNDEFINED:
                    qupush(this_sup->belf);
                    this_solv->dyprob = -6666.0;/*only for error
trapping */
                    solvable = 0;
                    break;
                case DEFINED:
                    this_solv->dyprob =
find_val(this_sup->belf);
                    break;
                case UNAVAILABLE:
                    this_solv->dyprob = (float) NOTAVALABLE;
                    break;
                default:
#ifdef TESTON
                    printf ("\n %d
",find_state(this_sup->belf));
#endif
                    terror("Unkown state of belief");
                    break;
            }
        }
    }/*end for all suporting beliefs step 1 */

    if (!solvable) {
#ifdef TESTON
        printf ("\nCould not solve the target this time");
#endif
        return;
    }

    /*
     *    Step 2 try to derive the value of the current belief
     *    Uses the infor stored in the solve array
     */
    ac_prob   = 0.0;
    ac_weight = 0.0;
    available = 0;

    /* for all suporting beliefs */
    for (curt_supt=0;curt_supt<Sec_btree[targ_blf].num_belf;curt_supt++)
{
        this_sup  = &(Sec_btree[targ_blf].rule[curt_supt]);
        prob   = (float)this_sup->prob;          /*    from    the
knowledge base */
        weight = (float)this_sup->weight;   /* from the knowledge base
*/

        enough = 0;             /*  if  the  following  sub-set  can
conclude */
        minprob = 100.0;  /* find the worst case beleif */

        for (;;curt_supt++) {
            this_sup  = &(Sec_btree[targ_blf].rule[curt_supt]);
            this_solv = &Suports[curt_supt];
#ifdef      TESTSOLV
```

```
                  printf   ("\nThe   prob   of   the   next   belief:   %f
",this_solv->dyprob);
#endif
                  if (this_solv->dyprob > 0) { /* by defin. a valid prob
*/
                        enough++;
                        if (this_solv->dyprob < minprob)
                              minprob = this_solv->dyprob;
                  }
                  /* UNTIL last belief in rule */
                  if (this_sup->contex >= ENDRULE) break;
            }
#ifdef      TESTSOLV
            printf   ("\nOne   complete   rule   set   examined   min
:%f",minprob);
#endif
            if (enough) {
                  available++;
                  ac_prob += minprob*prob*weight/100.0;
                  ac_weight += weight;
#ifdef      TESTSOLV
                  printf ("\nThe current value of prob and wight %f %f",
                  ac_prob,ac_weight);
#endif
            }
#ifdef TESTON
            else printf ("\nNot enough info ");
#endif

      }/*end for all suportin beleifs*/
      if (available) {
#ifdef TESTON
            printf ("\nThe current value solved of prob and wight %f %f",
             ac_prob,ac_weight);
            printf ("\n the final prob is: %f", ac_prob/ac_weight);
#endif
            /* change the state */
            Sec_btree[targ_blf].dyprob = ac_prob/ac_weight;
            Sec_btree[targ_blf].state = DEFINED;
      }
      else {
#ifdef TESTON
            printf ("\nthe current belief was not sloved");
#endif
            Sec_btree[targ_blf].dyprob = (float) NOTAVALABLE;
            Sec_btree[targ_blf].state = UNAVAILABLE;
      }
#ifdef      TESTSOLV
      wait();
#endif
      return;
} /*end function sove_suport */

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   find_state find_val
 -    Input       :   secondary blelief in highlevel ID              -
 -    Output      :
 -    Action      :   tryes to solve the target belief               -
 -    Date        :   27 Mar 90                                      -
 -    UpDate      :   27 Mar 90                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int   find_state(belief)
```

```
int belief;
{
      extern      secblf      Sec_btree[NUMSECBLF];

      if (belief>NUMSECBLF)  {
            printf ("\n\n belief ID in error: %d\n",belief);
            terror ("Belief ID out of range at find_state");
      }
      return (Sec_btree[--belief].state);
}

float find_val(belief)
int         belief;
{
      extern      secblf      Sec_btree[NUMSECBLF];

      if (belief>NUMSECBLF) {
            printf ("\n belief ID in error: %d\n",belief);
            terror ("Belief ID out of range at find_val.");
      }
      if (find_state(belief) != DEFINED)
            terror ("Trying to retreave an undefined belief");
      return (Sec_btree[--belief].dyprob);
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   sie_dump                                          -
 -    Input       :   none                                              -
 -    Output      :   none                                              -
 -    Action      :   reads the primary knowledge base form file        -
 -    Date        :   06 Mar 90                                         -
 -    UpDate      :                                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void  sie_dump()

{
      extern            secblf      Sec_btree[NUMSECBLF];
      unsigned    numbelfs;
      unsigned    curtbelf;
      int               j;

      printf ("\n\n Sec Knowledge base dump:");

      for(j=0;j<NUMSECBLF;j++)  {
            numbelfs=Sec_btree[j].num_belf;
            if (numbelfs) {
                  printf  ("\nThe number of suport of: %d is: %d ",j+1,
numbelfs);
                  printf   ("\n  Its   state   and   dynamic   prob   s:%d,
pr:%f\n",Sec_btree[j].state,Sec_btree[j].dyprob);
                  for (curtbelf=0;curtbelf<numbelfs;curtbelf++)
                        printf ("          R:     %d ",
Sec_btree[j].rule[curtbelf].contex);
                  printf("\n");


                  /* enter the belfs directly */
                  for (curtbelf=0;curtbelf<numbelfs;curtbelf++)
                        printf ("          B:     %d ",
Sec_btree[j].rule[curtbelf].belf);
                  printf("\n");
```

```
                  /* enter the probability in it is % form */
                  for (curtbelf=0;curtbelf<numbelfs;curtbelf++)
                        printf("     P:  %d",
Sec_btree[j].rule[curtbelf].prob);
                  printf("\n");

                  /* enter the weight, it is in % form */

                  for (curtbelf=0;curtbelf<numbelfs;curtbelf++)
                        printf("     W:  %d",
Sec_btree[j].rule[curtbelf].weight);

                  wait();
            }

      } /*until EOF */
} /*end sie_dump()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :    sie_constructor                                  -
 -    Input       :    none                                             -
 -    Output      :    none                                             -
 -    Action      :    reads the secondary knowledge base from file     -
 -    Date        :    23 Mar 90                                        -
 -    UpDate      :    23 Mar 90                                        -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
      Principle storage of secondary knowledge base
*/
secblf      Sec_btree[NUMSECBLF];

void  sie_constructor()
/*#define   TESTCONS*/
{
      extern            secblf      Sec_btree[NUMSECBLF];
      char        date_stamp[DATESIZE+56];
      char              dummy[80];
      int               numbelfs;
      int               curtbelf;
      int               i;
      int               input;
      int               prvrule, newrule;
      secblf*           this_blf;
      FILE*             datafile;

      pie_constructor();
      /* open the input data file terminate if error */
      if ((datafile = fopen(DATAFILE, "r"))==NULL)
           terror("Missing the primary KB file");

      /* remove the header store date stamp */
      for (i=1;i<DATELOC;i++) fscanf(datafile, "%s",dummy);
      fscanf(datafile, "         %s ",date_stamp);
      printf ("\nSecondary KB date stamp: %s\n",date_stamp);
      /* load the rules */
      for(this_blf=Sec_btree;;this_blf++) {
          printf(".");

            /* read the number of suporting Suports and save this value */
            if (fscanf(datafile, "%d",&numbelfs)==EOF) break;
#ifdef TESTCONS
```

```
        printf("Number of belfs: %d",numbelfs);
#endif
        this_blf->num_belf = numbelfs;
        this_blf->state = UNDEFINED;
        this_blf->dyprob = (float)INVALIDPR;

        /* enter the first rule ids */
        if (numbelfs > 0) {
            fscanf(datafile, "%d",&newrule);
#ifdef TESTCONS
            printf("\nFirst rule id: %d",newrule);
#endif
            if ((prvrule=newrule)>EXCESS)
this_blf->rule[0].contex=ELSERULE;
            else this_blf->rule[0].contex=THENRULE;
        }
        /* enter the rest using the prv rule to chain */
        for (curtbelf=1;curtbelf<numbelfs;curtbelf++) {
            fscanf(datafile, "%d",&newrule);
#ifdef TESTCONS
            printf("\nRule id: %d",newrule);
#endif
            if (newrule>EXCESS)
this_blf->rule[curtbelf].contex=ELSERULE;
            else this_blf->rule[curtbelf].contex=THENRULE;

            if (prvrule!=newrule) {
                /*indicate that the previous belf was the last of
the rule */
                this_blf->rule[(curtbelf-1)].contex += ENDRULE;
            }
            prvrule=newrule;
        }
        /* the last is aways an endrule */
        this_blf->rule[(curtbelf-1)].contex += ENDRULE;

#ifdef TESTCONS
        wait();
#endif

        /* enter the belfs directly */
        for (curtbelf=0;curtbelf<numbelfs;curtbelf++) {
            fscanf(datafile, "%d",&(this_blf->rule[curtbelf].belf));
#ifdef TESTCONS
            printf("\nbelf id: %d",this_blf->rule[curtbelf].belf);
#endif
        }
#ifdef TESTCONS
        wait();
#endif

        /* enter the probability in it is % form */
        for (curtbelf=0;curtbelf<numbelfs;curtbelf++) {
            fscanf(datafile, "%d",&input);
            this_blf->rule[curtbelf].prob=(char)input;
#ifdef TESTCONS
            printf("\nProb id: %d",this_blf->rule[curtbelf].prob);
#endif
        }

#ifdef TESTCONS
        wait();
```

```
#endif

            /* enter the weight, it is in % form */
            for (curtbelf=0;curtbelf<numbelfs;curtbelf++) {
                  fscanf(datafile, "%d",&input);
                  this_blf->rule[curtbelf].weight=(char)input;
#ifdef TESTCONS
                  printf("\nWeight is
%d",this_blf->rule[curtbelf].weight);
#endif
            }
#ifdef TESTCONS
            wait();
#endif

      } /*until EOF */
      fclose(datafile);
      printf("\nSecondary Knowledge loaded \n");
} /*end sie_constructor()*/

/*
end file sie.c */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell
   =    Sub-Project   :   Secondary Inference Engine
   =    File          :   queue.c
   =    Author        :   Robert D. Rourke
   =    Start Date    :   25 Mar 1990
   =    Update        :   27 Mar 1990
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
/*#define         TESTON*/
/*#define   RMXON*/

#include    <stdio.h>
#include    <ctype.h>
#include    "queue.hx"


#define     MAXQUEUE          100
#define TLEMPTY               0


/* A stack like queue LIFO */
typedef struct
{
      int         tail;
      int         max;
      int         info[MAXQUEUE];
} lifoqu;
/*
main()
{
      int   i;

      qu_constructor();
      for (i=0;i<MAXQUEUE;i++)   {
            printf("\n IN %i",i);
            qupush(i);
      }
      for (;!quempt();) printf ("\n OUT %i",quexam());
```

```
}
*/

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   qupop                                         -
 .    Input       :   none                                          -
 -    Output      :   none                                          -
 -    Action      :   queue-                                        -
 -    Date        :   26 Mar 90                                     -
 -    UpDate      :   26 Mar 90                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
int         qupop()
{
      extern            lifoqu      Secqueue;
#ifdef            TESTON
      printf ("<<poping a belief>>");
#endif
      if (quempt())
            terror ("Queue underflow");

      return (Secqueue.info[--Secqueue.tail]);
}
/*end funciton qupop*/


/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   quexam                                        -
 -    Input       :   none                                          -
 -    Output      :   none                                          -
 -    Action      :   queue-                                        -
 -    Date        :   26 Mar 90                                     -
 -    UpDate      :   26 Mar 90                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
int         quexam()
{
      extern            lifoqu      Secqueue;
#ifdef            TESTON
      printf ("<<examing a belief>>");
#endif

      if (quempt())
            terror ("Queue underflow");

      return (Secqueue.info[(Secqueue.tail-1)]);
}
/*end funciton quexam*/



/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   quempt                                        -
 -    Input       :   none                                          -
 -    Output      :   0 if not emplty, 1 if empty                   -
 -    Action      :   queue-                                        -
 -    Date        :   26 Mar 90                                     -
 -    UpDate      :   26 Mar 90                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
int         quempt()
{
      extern            lifoqu      Secqueue;
      return (Secqueue.tail == TLEMPTY);
```

```
}
/*end funciton quempt*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   qupush                                       -
 -    Input       :   none                                         -
 -    Output      :   none                                         -
 -    Action      :   queue-                                       -
 -    Date        :   26 Mar 90                                    -
 -    UpDate      :   26 Mar 90                                    -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void  qupush(newinfo)
int         newinfo;
{
      extern            lifoqu      Secqueue;
#ifdef            TESTON
      printf ("<<pushing a belief>>");
#endif

      if (Secqueue.tail >= MAXQUEUE)
            terror ("Queue overflow");
      Secqueue.info[Secqueue.tail++] = newinfo;
}
/*end funciton qupush*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   qu_constructor                               -
 -    Input       :   none                                         -
 -    Output      :   none                                         -
 -    Action      :   queue-                                       -
 -    Date        :   26 Mar 90                                    -
 -    UpDate      :   26 Mar 90                                    -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*    Convention:
      if tail = 0 then queue is empty
      the tail points to an open slot
      to enter, tail slot is filled then incremented
 */

lifoqu      Secqueue;

void  qu_constructor()
{
      extern            lifoqu      Secqueue;
      register    i;

      Secqueue.tail = TLEMPTY;
      Secqueue.max = MAXQUEUE;
      for (i=0;i<MAXQUEUE;i++)
            Secqueue.info[i]=-22;
}
/*end funciton qu_constructor*/


/* end file queue.c
 */
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KCOLA shell                              =
   =    Sub-Project   :   Human interface                          =
```

```
 =    File         :   sie.h                                  =
 =    Author       :   Robert D. Rourke                       =
 =    Start Date   :   23 Mar 1990                            =
 =    Update       :   23 Mar 1990                            =
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#define     DATAFILE    "sdknbase.dat"
#define     MAXBELF            20                        /*   max   num   of
supportin beliefs */
#define     CLSPACE            2
#define     DATESIZE    15
#define     TYPESIZE    4

#define     DATELOC            9

#define     EXCESS             500                       /* diferenc between
primar & second */

#define     THENRULE    1                          /* possible rule senarios
*/
#define     ELSERULE    2
#define     CONTINUED   3
#define     ENDRULE            0x10                /* added to last rule */
#define     MASKEND            3

#define     DEFINED            1                         /* blief states */
#define     UNDEFINED   2
#define     UNAVAILABLE 3

#define INVALIDPR       -444
#define NOTAVALABLE            -666

/*
 *    One component of the secondary knowledge base
 */

typedef struct
{
      char         contex;      /* then or else clause */
      int          belf;
      char         prob;
      char         weight;
} sr; /*secondary rule*/

typedet struct
{
      unsigned     num_belf;
      sr           rule[MAXBELF];
      float        dyprob;
      char         state;
} secblf;

typedef struct
{
      float        dyprob;
      int                state;
} binf;


#ifdef RMXON
int          chk_logic ();
```

```
void  sie_dump();
void  sie_constructor();
void  sove_suport();
int         find_state();
float find_val();
float find_sbelief();
#else

void  sie_constructor(void);
void  sie_dump(void);
int         chk_logic (float* operand1, int operator, float* operand2);
void  sove_suport(int targ_blf);
int         find_state(int highlevel_beliefID);
float find_val(int highlevel_beliefID);
float find_sbelief(int goal_blelief);

#endif

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell                                 =
   =    Sub-Project   :   Goal Manager                                =
   =    File          :   es.c                                        =
   =    Author        :   Robert D. Rourke                            =
   =    Start Date    :   30 Mar 1990                                 =
   =    Update        :   30 Mar 1990                                 =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
#define     TESTON
/*#define         RMXON*/
/*#define         USEHALOS*/

#include    <stdio.h>
#include    <ctype.h>

#ifdef RMXON
#include    <udi.h>
#include    <rmx.h>
#endif

#include    "gm.hx"
#include    "es.h"

int   error_state;
#ifdef      RMXON
main() {
      extern void WES$TASK();


      printf ("\nAbout to creat");
      wait();
      rq$create$task (PRIORITY,(char*)WES$TASK,(int)0,(char*)0L,STCKSIZE,
        1,&error_state);
      printf ("\n created ");

      WES$TASK();
}

#else
void  WES$TASK();

main() {
```

```
      WES$TASK();
}

#endif
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :   WES$TASK                                    -
 -    Input        :   none                                        -
 -    Output       :   none                                        -
 -    Action       :                                               -
 -    Date         :   30 Mar 90                                   -
 -    UpDate       :                                               -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int         Aut_goal;

void  WES$TASK()
{

      extern       int          WES$COMMAND$DMB;
      extern       int          Aut_goal;
      int goal;
      int          error_state;
      int          numreturn;

#ifdef       RMXON
      rq$sleep (1000,&error_state);
#endif

      printf ("\n THe WESTASK is running.");
      wait();
      for (;;) {
            goal = 1;
            for(;goal;) {
                  gm_constructor();
                  if (goal=List_goal()) Start_goal(goal);
            }
            printf    ("\n\n    Auotomatic    monitoring    procedures
initiated...\n");

#ifdef       USEHALOS
            numreturn = rq$receive$data (WES$COMMAND$DMB,
              &Aut_goal,WAIT4EVER,&error_state);
            if (numreturn != sizeof(int)) {
                  printf ("\nWrong size of token %d",numreturn);
                  terror ("expert system task");
            }
            Start_goal(Aut_goal);
#else
            wait();
#endif
      }
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :   es$constructor()                            -
 -    Input        :   none                                        -
 -    Output       :   none                                        -
 -    Action       :   Creates the mail boxes and the ES task      -
 -    Date         :   30 Mar 90                                   -
 -    UpDate       :                                               -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

```
#ifdef      RMXON
 sdsadsadsad
int         WES$COMMAND$DMB;
extern      void  WES$TASK();
extern      int         WES$COMMAND$DMB;

void  es$constructor()
{

      /* the mail box used to recieve commands */
      WES$COMMAND$DMB = rq$create$mailbox (DATAMB, &error_state);

      /* the main task in the ES     */
      printf ("\nAbout to creat");
      wait();
      rq$create$task (PRIORITY,(char*)WES$TASK,0,(char*)01,STCKSIZE,
        FLOATS,&error_state);
      printf ("\n created ");
      wait();
      return;

}

#endif
/*
end file es.c */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell                               =
   =    Sub-Project   :   Goal Manager                              =
   =    File          :   es.c                                      =
   =    Author        :   Robert D. Rourke                          =
   =    Start Date    :   30 Mar 1990                               =
   =    Update        :   30 Mar 1990                               =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
#define     TESTON
/*#define         RMXON*/
/*#define         USEHALOS*/

#include    <stdio.h>
#include    <ctype.h>

#ifdef RMXON
#include    <udi.h>
#include    <rmx.h>
#endif

#include    "gm.hx"
#include    "es.h"

int   error_state;
#ifdef      RMXON
main() {
      extern void WES$TASK();


      printf ("\nAbout to creat");
      wait();
      rq$create$task (PRIORITY,(char*)WES$TASK,(int)0,(char*)0L,STCKSIZE,
        1,&error_state);
      printf ("\n created ");
```

```
      WES$TASK();
}

#else
void  WES$TASK();

main() {

      WES$TASK();
}

#endif
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function   :   WES$TASK                                      -
 -    Input      :   none                                          -
 -    Output     :   none                                          -
 -    Action     :                                                 -
 -    Date       :   30 Mar 90                                     -
 -    UpDate     :                                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int         Aut_goal;

void  WES$TASK()
{

      extern      int         WES$COMMAND$DMB;
      extern      int         Aut_goal;
      int goal;
      int         error_state;
      int         numreturn;

#ifdef      RMXON
      rq$sleep (1000,&error_state);
#endif

      printf ("\n THe WESTASK is running.");
      wait();
      for (;;) {
            goal = 1;
            for(;goal;) {
                  gm_constructor();
                  if (goal=List_goal()) Start_goal(goal);
            }
            printf    ("\n\n   Auotomatic     monitoring    procedures
initiated...\n");

#ifdef      USEHALOS
            numreturn = rq$receive$data (WES$COMMAND$DMB,
              &Aut_goal,WAIT4EVER,&error_state);
            if (numreturn != sizeof(int)) {
                  printf ("\nWrong size of token %d",numreturn);
                  terror ("expert system task");
            }
            Start_goal(Aut_goal);
#else
            wait();
#endif
      }
}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
 -    Function   :   es$constructor()                              -
 -    Input      :   none                                          -
 -    Output     :   none                                          -
 -    Action     :   Creates the mail boxes and the ES task        -
 -    Date       :   30 Mar 90                                     -
 -    UpDate     :                                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

#ifdef      RMXON
 sdsadsadsad
int         WES$COMMAND$DMB;
extern      void  WES$TASK();
extern      int         WES$COMMAND$DMB;

void  es$constructor()
{

      /* the mail box used to recieve commands */
      WES$COMMAND$DMB = rq$create$mailbox (DATAMB, &error_state);

      /* the main task in the ES     */
      printf ("\nAbout to creat");
      wait();
      rq$create$task (PRIORITY,(char*)WES$TASK,0,(char*)01,STCKSIZE,
        FLOATS,&error_state);
      printf ("\n created ");
      wait();
      return;

}

#endif
/*
end file es.c */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :  KOOLA shell                                 =
   =    Sub-Project   :  Goal Manager                                =
   =    File          :  es.h                                        =
   =    Author        :  Robert D. Rourke                            =
   =    Start Date    :  30 Mar 1990                                 =
   =    Update        :  02 Apr 1990                                 =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

/*    RMX constants */
/* task constants */
#define     PRIORITY    150         /* for any task started             */
#define NODATASEG 0           /* disk uses private DS             */
#define     NOSTCKPNT   0L          /* rmx creats the stack             */
#define     FLOATS            1           /* task may use floating pnt  */
                                          /* task flags
            */
#define     STCKSIZE    16000 /* size of stack given to new */
                                          /* task
            */

#define     DATAMB            0x0020      /* data     FIFO mailbox
            */
#define     TOKONMB           0           /* token FIFO mailbox
      */
#define WAIT4EVER 0xffff      /* used for waiting at M/B          */
```

```
/*
 end file es.h */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell                                   =
   =    Sub-Project   :   Goal Manager                                  =
   =    File          :   gm.c                                          =
   =    Author        :   Robert D. Rourke                              =
   =    Start Date    :   28 Mar 1990                                   =
   =    Update        :   02 Apr 1990                                   =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
#define      TESTON
/*#define    RMXON*/
/*#define          USECOSI*/

#include     <stdio.h>
#include     <ctype.h>

#ifdef RMXON
#include     <udi.h>
#include     <rmx.h>
#endif


#include     'knowbase.h"
#include     "gm.h"
#include     "pie.hx"
#include     "sie.hx"


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   List_goal                                         -
 -    Input       :   none                                              -
 -    Output      :   highlevel goal id selected                        -
 -    Action      :   Displays all startable goal frames                -
 -    Date        :   29 Mar 90                                         -
 -    UpDate      :                                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int          List_goal()

{
      extern      metarl      Goal_tree[NUMGOAL];
      int         goal_disp[MAXGOALDISPLAY+1];
      int         num_goals, cur_goal;
      int         gl;
      int         rep;

      /* load all of the valid goals */
      num_goals=0;
      for (gl=0;gl<NUMGOAL&&num_goals<MAXGOALDISPLAY;gl++)
            if (Goal_tree[(gl)].start) goal_disp[num_goals++]=gl;
      for (;;)     {
            /* display all of the goals */
            printf("\n\nSelect an knowledge-base application to run in
KOOLA:");
            for (gl=0;gl<num_goals;gl++)
                  printf ("\n
%c - %s",'A'+gl,Goal_tree[goal_disp[gl]].name);
            printf ("\n\n                                                   %c
- %s",'A'+gl,"AUTONOMOUSE CONTROL");
            printf("\nPlease enter a letter: ");
```

```
            rep = getchar();
            getchar();
            if (rep>'Z') rep=rep-'a';
            else rep=rep-'A';
            if   (rep<0||rep>num_goals)   printf   ("\aWrong!!!         Try
again...\n");
            else break;
        }
        if (rep==num_goals) return (0); /*autonomouse control*/
/*      printf      ("\n      Select  :      %s        i
",Goal_tree[goal_disp[rep]].name,goal_disp[rep]+1);*/
        return ((goal_disp[rep]+1));
}




/*float      find_sbelief(int hl_belief_id);*/

void  action_gen();
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :    Start_goal                                        -
 -    Input       :    none                                              -
 -    Output      :    none                                              -
 -    Action      :    reads the primary knowledge base form file        -
 -    Date        :    26 Mar 90                                         -
 -    UpDate      :                                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int          Start_goal(init_goal)
int          init_goal;

{
        extern              metarl      Goal_tree[NUMGOAL];
        int                 nextgoal;
        metarl*             this_goal;
        conseq*             next;
        int                 depth;
        float        belf;


        /*    check if it is a valid start goal */

        if (init_goal<=0||init_goal>NUMGOAL) terror ("Envalid goal ID",.
        if   (!Goal_tree[init_goal-1].start)   terror   ("Not  a  valid  start
goal");
        nextgoal = init_goal;
        for (depth=1;nextgoal!=TERMINATE;depth++) {
                printf ("\n\n\nDepth :%d",depth);
                this_goal = &Goal_tree[nextgoal-1];
                printf    (",   Currently    working    on    goal    [illegible]
%s",this_goal->name);
/*              printf("Requesting a sec belief : %d",this_goal->belief);*
                belf=find_sbelief(this_goal->belief);
                printf("\nReturn prob: %f",belf);
                if (belf<0) {
                        printf("\n Could not succesfuly solve the goal at depth
%d",depth);
                        return (0);
                }
                if (chk_logic (&belf,this_goal->operat,&this_goal->number)),
                        next = &this_goal->thn;
```

```
            else next = &this_goal->els;
            /* do the action */
            if (next->action!=NOACTION)
                  action_gen(next->action);
            else  {
                  printf ("\nNo action needed");
            }
            /* next goal to do */
            nextgoal = next->chain;
            /* check for excessive looping */
            if (depth > MAXDEPTH) terror ("Goal solving gone to deep");
      }
      printf ("\nApplication terminated at depth %d, no errors.",depth);
      wait();
      return (depth);
}/*end function Start_goal*/


/*
float find_sbelief(int hl_belief_id)
{
      float prob;

      printf ("\n**Enter the secondary belief [%d] :",hl_belief_id);
      scanf ("%f",&prob);
      getchar();
      return (prob);
}
*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function      :   action_gen                                          -
 -    Input         :   token                                               -
 -    Output        :   none                                                -
 -    Action        .   sends a token to COSI                               -
 -    Date          :   28 Mar 90                                           -
 -    UpDate        :   28 Mar 90                                           -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

void  action_gen(action)
int action;
{
      extern int WCOSI$COMMAND$DMB;
      extern int WCOSI$READY$1US;

      int   error_state;

      printf ("\nWe are sending the action token [%d] to COSI",action);
#ifdef            USECOSI
      rq$send$data(WCOSI$COMMAND$DMB, &action,sizeof(int), &error_state);
      rq$receive$units (WCOSI$READY$1US, ONEUNIT, FOREVER, &error_state);
#else
      wait();
#endif

}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function      :   gm_constructor                                      -
 -    Input         :   none                                                -
```

```
 -    Output      :   none                                        -
 -    Action      :   reads the meta knowledge base from file    -
 -    Date        :   28 Mar 90                                   -
 -    UpDate      :   28 Mar 90                                   -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *
/*
      Principle storage of secondary knowledge base
*/
metarl       Goal_tree[NUMGOAL];

void gm_constructor()
/**#define   TESTCONS*/
{
      extern             metarl      Goal_tree[NUMGOAL];
      char         date_stamp[DATESIZE+56];
      char               dummy[80];
      FILE*              datafile;
      metarl*            this_goal;
      int                i;
      int                strt_flg;

      sie_constructor();
      /* open the input data file terminate if error */
      if ((datafile = fopen(DATAFILE, "r"))==NULL)
           terror("Missing the meta KB file");

      /* remove the header store date stamp */
      for (i=1;i<DATELOC;i++) fscanf(datafile, "%s",dummy);
      fscanf(datafile, "          %s ",date_stamp);
      printf ("\nMeta KB date stamp: %s\n",date_stamp);
      /* load the rules */
      for(this_goal=Goal_tree;;this_goal++) {
          printf(".");

            /* read the name of the goal */
            if (fscanf(datafile, "%d",&strt_flg)==EOF) break;
#ifdef TESTCONS
            printf("Startablility of this goal: %d",strt_flg);
#endif
            this_goal->start = strt_flg;

            /* enter the rest of the goal */



            fgets(dummy,CLSPACE,datafile);
            this_goal->name[NAMESIZE]= '\0';
            fgets(this_goal->name,NAMESIZE,datafile);


/*          fscanf(datafile, "%s",this_goal->name);*/
            fscanf(datafile, "%d",&(this_goal->belief));
            fscanf(datafile, "%d",&(this_goal->operat));
            fscanf(datafile, "%f",&(this_goal->number));
            fscanf(datafile, "%d",&(this_goal->thn.chain));
            fscanf(datafile, "%d",&(this_goal->els.chain));
            fscanf(datafile, "%d",&(this_goal->thn.action));
            fscanf(datafile, "%d",&(this_goal->els.action));

#ifdef TESTCONS
            printf("\nName: %s",this_goal->name);
            printf("\nBelief :%d",this_goal->belief);
```

```
            printf("\nOprerator :%d",this_goal->operat);
            printf("\nNumbert :%f",this_goal->number);
            printf("\n then chain:%d  ",this_goal->thn.chain);
            printf("else chain:%d    ",this_goal->els.chain);
            printf("\n then action:%d  ",this_goal->thn.action);
            printf("then action    %d",this_goal->els.action);
            wait();
#endif
      } /*until EOF */
      fclose(datafile);
      printf("\nMeta Knowledge loaded \n");
} /*end gm_constructor()*/

/*
end file gm.c */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell                                   =
   =    Sub-Project   :   Goal Manager                                  =
   =    File          :   gm.h                                          =
   =    Author        :   Robert D. Rourke                              =
   =    Start Date    :   28 Mar 1990                                   =
   =    Update        :   28 Mar 1990                                   =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#ifdef RMXON
void        gm_dump();
void        gm_constructor();
int         Start_goal();
int         List_goal();
#else

void        gm_constructor(void);
void        gm_dump(void);
int         Start_goal(int init_goal);
int         List_goal(void);
#endif
/*
 end file gm.h */

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   KOOLA shell                                   =
   =    Sub-Project   :   Goal Manager                                  =
   =    File          :   gm.h                                          =
   =    Author        :   Robert D. Rourke                              =
   =    Start Date    :   28 Mar 1990                                   =
   =    Update        :   10 Apr 1990                                   =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

/*    RMX constants */
#define     PRIORIRY     150          /* for any task started */
#define     DATAMB             0x0020      /* data      FIFO mailbox      */
#define     TOKONMB            0           /* token FIFO mailbox    */
#define ONEUNIT          1            /* one token for the semephore */
#define FOREVER          0xffff       /* wait at an rmx call for ever */

/*    Entering data from file */
#define     DATAFILE     "mtknbase.dat"
#define     NAMESIZE     16
#define     CLSPACE            7
```

```
#define     DATESIZE    15
#define     TYPESIZE    4
#define     DATELOC           9

#define     EXCESS            500                     /* diferenc between
primar & second */


#define TERMINATE 0
#define NOACTION  0
#define MAXDEPTH  100                     /*  maximum  depth  of  goal
processing */

#define INVALIDPR       -444
#define NOTAVALABLE           -666
#define MAXGOALDISPLAY  15                /*  maximum  number  of  goals
displayed */

/*
 *    One component of the secondary knowledge base
 */

typedef struct
{
      int         chain;
      int               action;
} conseq;

typedef struct
{
      char        name[NAMESIZE+1];
      int               start;
      int               belief;
      int               operat;
      float       number;
      conseq            thn;
      conseq            els;
} metarl;


#ifdef RMXON
#else
#endif
/*
 end file gm.h */
```

```
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =   Project         :   KOOLA programing language Shell            =
*  =   Sub-project     :   Entry for objects                          =
*  =   File            :   main.prg                                   =
*  =   Author          :   Robert D. Rourke                           =
*  =   Date            :   16 Mar 1989                                =
*  =   Update          :   25 Sep 1989                                =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
*     Databases:
*     Select      Name  Index Code
*     -------+---------+---------+--------
*     1           rule        rule        r
*     2           goal        goal        g
*     3           belief      belief      b
*     4           internal    internal    i
*     5           extreq      extreq      x
*     6           action      action      a
*     all index set for var_class+name except goal
*

EXTERNAL Add_belief
EXTERNAL Add_rule
EXTERNAL Print_rule
EXTERNAL Add_goal


Date =' 2 Feb 1990'
Version = '2.20'

PUBLIC Deflt_path, Escape
* set the colour for a colour monitor
In_colour = .f.
IF iscolor ()
     PUBLIC Colour_Edit
     PUBLIC Colour_Menu
     In_colour = .T.
     Colour_Edit = "GR+/B,GR+/B,B,B,W+/RB"
     Colour_Menu = "GR+,W+/B,B,B,W+/BG"
     setcolor(Colour_Menu)
ENDIF
SET CENTURY ON
SET DATE BRITISH
SET WRAP ON
CLEAR

Deflt_Path = Get_direct('sim\')
File_error = .F.

SELECT 1
File_name = 'rule.dbf'
IF file (File_name)
      USE &File_name
*     index on var_class+name to rule
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

 ELECT 3
 le_name = 'belief.dbf'
 F file (File_name)
```

```
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 4
File_name = 'internal.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 5
File_name = 'extreq.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 6
File_name = 'action.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF


SELECT 2
File_name = 'goal.dbf'
IF file (File_name)
      USE &File_name
*     INDEX ON name to Goal
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF



IF File_error
      ? '*** File Error termination of KCOLA   ***'
      ?
      RETURN
ENDIF
CLEAR
PRIVATE Main_loop
Main_loop = 1
*
* main program loop repeat utill
*
DO WHILE .T.
      CLEAR
     SET MESSAGE TO 2
     MsgCent ('MENU')
     SET INDEX TO
```

```
     Main_loop = Main_menu(Main_loop)
      DO CASE
      CASE Main_loop = 1
            Koola_title()
            CLEAR
      CASE Main_loop = 2
            *Goal Frame
            SELECT 2    &&goal.dbf
            SET INDEX TO goal
            DO_object('GOAL')
      CASE Main_loop = 3
            *Rule
            SELECT 1    &&rule.dbf
            SET INDEX TO rule
            DO_object('RULE')
      CASE Main_loop = 4
            *Belief
            SELECT 3    && belief.dbf
            SET INDEX TO belief
            DO_object('BELIEF')
      CASE Main_loop = 5
            *External inquiry
            SELECT 5    &&extreq.dbf
            SET INDEX TO extreq
            Do_object('EXTREQ')
      CASE Main_loop = 6
            *human interface enquiry
            SELECT 4    &&internal.dbf
            SET INDEX TO internal
            Do_object('INTERNAL')
      CASE Main_loop = 7
            *Action
            SELECT 6
            SET INDEX TO action
            Do_object('ACTION')
      CASE Main_loop = 8
            *file
            File_proc()

      OTHERWISE
            *quit
            MsgCent('QUIT')
          IF Quit_menu() = 2
                  EXIT
            ENDIF
      ENDCASE
ENDDO
CLEAR
Beep('F')
? '***  Normal termination of KOOLA  ***'
? '(c) R. Rourke 1989, 1990'
? '(c) ZenRob ISDN developments 1990'
?
RETURN

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function         :   Main_menu                                  -
*  -    Input            :   void                                       -
*  -    Output           :   value of choice                            -
*  -    Date             :   04 Mar 89                                  -
*  -    UpDate           :          89                                  -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
FUNCTION Main_menu
PARAMETER WaitKey
@ 1,0 CLEAR TO 2,79
@ 23, 0 CLEAR
@ 24,  0  SAY dtoc(date())
@ 24, 20  SAY Deflt_Path
SET COLOR TO 7+
@ 0, 0 SAY 'KOOLA Production Environment  V'+Version+'  '+Date
IF In_colour
     setcolor(Colour_Menu)
ELSE
     SET COLOR
ENDIF
@ 01, 0       PROMPT 'Info' MESSAGE 'Info about KOOLA'
@ 01, col()+2 PROMPT 'Goal' MESSAGE 'Goal rules that starts an inference'
@ 01, col()+2 PROMPT 'Rule' MESSAGE 'Backward chanining production rule'
@ 01, col()+2 PROMPT 'Belief' MESSAGE 'An infered belief'
@ 01, col()+2 PROMPT 'External' MESSAGE 'External inquiry'
@ 01, col()+2 PROMPT 'Human' MESSAGE 'Human interface inquiry'
@ 01, col()+2 PROMPT 'Action' MESSAGE 'External action'
@ 01, col()+2 PROMPT 'File' MESSAGE 'Directory List'
@ 01, col()+2 PROMPT 'Quit' MESSAGE 'Quit the program'
MENU TO WaitKey
RETURN(WaitKey)
*end menu main_menu

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function        :   Obj_menu                                  -
* -    Input           :   void                                      -
* -    Output          :   value of choice                           -
* -    Date            :   23 May 89                                 -
* -    UpDate          :          89                                 -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Obj_menu
PARAMETER WaitKey
@ 1,0 CLEAR TO 2,79
@ 01, 0       PROMPT 'Change' MESSAGE 'Add or edit'
@ 01, col()+2 PROMPT 'Examine' MESSAGE ;
 'Observe the objects in a spread sheet'
@ 01, col()+2 PROMPT 'Reindex' MESSAGE 'Verify the order of the objects'
@ 01, col()+2 PROMPT 'Print' MESSAGE 'Print the current object'
@ 01, col()+2 PROMPT 'Quit' MESSAGE 'Return to previous menu'
MENU TO WaitKey
RETURN(WaitKey)
*end menu Obj_menu

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function        :   Rule_menu
* -    Input           :   void
* -    Output          :   value of choice                           -
* -    Date            :   23 May 89                                 -
* -    UpDate          :          89                                 -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Rule_menu
PARAMETER WaitKey
 @ 1,0 CLEAR TO 2,79
 @ 01, 0       PROMPT 'Fact' MESSAGE 'Add some new fact-based rules'
 @ 01, col()+2 PROMPT 'Belief' MESSAGE 'Add some new belief-based rules'
 @ 01, col()+2 PROMPT 'Examine' MESSAGE ;
  'Observe the objects in a spread sheet'
 @ 01, col()+2 PROMPT 'Reindex' MESSAGE 'Verify the order of the rules'
 @ 01, col()+2 PROMPT 'Print' MESSAGE 'Print the current object'
```

```
 @ 01, col()+2 PROMPT 'Quit' MESSAGE 'Return to main menu'
MENU TO WaitKey
RETURN(WaitKey)
*end menu Rule_menu

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -     Function        :   Goal_menu                               -
* -     Input           :   void                                    -
* -     Output          :   value of choice                         -
* -     Date            :   23 May 89                               -
* -     UpDate          :          89                               -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Goal_me .u
PRIVATE WaitKey
@ 1,0 CLEAR TO 2,79
@ 01, 0       PROMPT 'Change' MESSAGE 'Add some new Goals'
@ 01, col()+2 PROMPT 'Examine' MESSAGE ;
 'Observe the goals in a spread sheet'
@ 01, col()+2 PROMPT 'Reindex' MESSAGE 'Verify the order of the goals'
@ 01, col()+2 PROMPT 'Print' MESSAGE 'Print the current object'
@ 01, col()+2 PROMPT 'Quit' MESSAGE 'Return to main menu'
MENU TO WaitKey
RETURN(WaitKey)
*end menu Goal_menu

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -     Function        :   Not_yet                                 -
* -     Input           :   void                                    -
* -     Output          :   value of choice                         -
* -     Date            :   23 May 89                               -
* -     UpDate          :          89                               -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Not_yet
beep()
ErrWait ("This routine is not yet supported, Press Esc DU")
@ 24, 0 CLEAR
RETURN(.T.)

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -     Function        :   DO_object                               -
* -     Input           :   void                                    -
* -     Output          :   value of choice                         -
* -     Date            :   23 May 89                               -
* -     UpDate          :          89                               -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION DO_object
PARAMETER Obj_name

DECLARE Ans[3]
PRIVATE Levl, Lev2, Macro, S_buf
Levl = 1
SAVE SCREEN TO S_buf
DO WHILE .T.
     MsgCent (Obj_name)
      Levl = Obj_menu(Levl)
      DO CASE
      CASE Levl = 1
            *build
            Macro = 'Add_'+Obj_name
            PRIVAT Contn
            DO WHILE .T.
```

```
                  DO &Macro
                  IF !Escape
                        IF !AskN ("Enter another object?")
                              EXIT
                        ENDIF
                  ELSE
                        EXIT
                  ENDIF
            ENDDO
            RESTORE SCREEN FROM S_buf
      CASE Levl = 2
            *examine
            browse()
      CASE Levl = 3
            *reindex
            MsgWait()
            PACK
      CASE Levl = 4
            *print
            IF file (Obj_name+'.frm')
                  Macro = Obj_name
                  PntWait()
                  SET CONSOLE OFF
                  REPORT FORM &Macro TO PRINT
                  SET CONSOLE ON
            ELSE
                  Macro = 'PRINT_'+Obj_name
                  DO &Macro
            ENDIF
      OTHERWISE
            EXIT
      ENDCASE
ENDDO
RETURN(.T.)
*end function DO_object

*
*end file main.prg

*   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*   =    Project          :   KOOLA programming language
*   =    Sub-project      :   Entry for objects
*   =    File             :   action.prg                                 -
*   =    Author           :   Robert D. Rourke                           -
*   =    Date             :   19 Mar 1989                                -
*   =    Update           :   02 Oct 1989
*   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -
*
*      FUNCTIONS:
*            Add_action
*            Get_action
*            act_var
*            Get_var
*   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*   -    Function         :   Get_action                               -
*   -    Input            :   The variable class                       -
*   -    Output           :   action chosen                            -
*   -    Date             :   19 May 89                                -
*   -    UpDate           :   02 Sep 89                                -
*   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Get_action
PARAMETER Var           &&the var class to use
```

```
SELECT 6    && action.dbf
SET INDEX TO action
PRIVATE Act_new, Act_pnt, B_action,S_buf
SAVE SCREEN TO S_buf
*
*make a list of previous actions
*
Act_pnt = 0
Act_new = 0
PRIVATE Act_list
DECLARE Act_list[50]
SEEK Var
DO WHILE var_class = var
      Act_pnt = Act_pnt + 1
      Act_list[Act_pnt] = name
      SKIP
ENDDO
Act_pnt = Act_pnt + 1
Act_list[Act_pnt] = 'NO ACTION'


PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            RESTORE SCREEN FROM S_buf
            RETURN('NUL')
      CASE Navigate = 1
            RESTORE SCREEN FROM S_buf
            @ 2,40 CLEAR TO 22, 78
            @ 1,39 TO 23,79 DOUBLE
            @ 2, 40 SAY 'Select an action: '
            @ 3,40 SAY 'Var Class = '+Var
            *
            *     make selection
            *
            PRIVATE i
            FOR i = 1 TO Act_pnt
                  @ 4+i,55 PROMPT Act_list[i]
            NEXT
            @ 4+i, 55 PROMPT '*NEW           '
            PRIVATE Select
            Select = Act_pnt
            MENU TO Select
            DO CASE
            CASE Select = 0
                  Navigate = Navigate - 1
            CASE Select > Act_pnt
                  Navigate = Navigate + 1
            OTHERWISE
                  RESTORE SCREEN FROM S_buf
                  RETURN (Act_list[Select])
            ENDCASE

      CASE Navigate = 2
            * add a new action
            B_action = space(15)
            Act_new = Act_new + 1
            @ 4+Act_pnt+Act_new, 40 SAY 'Enter action: ' GET B_action
            READ
            IF ( !updated()  .OR. lastkey() = 27)
```

```
                  Act_new = Act_new - 1
                  Navigate = Navigate - 1
            ELSE
                  SEEK (var+B_action)
                  IF eof()
                        *does not already exists
                        @ 4+Act_pnt+Act_new, 55 SAY B_action
                        Navigate = Navigate + 1
                  ELSE
                        Act_new = Act_new - 1
                        ErrWait ('action name already exists')
                  ENDIF
            ENDIF

      CASE Navigate = 3
            * add the rest of the enquiry
            IF Cmplt_action(B_action)
                  Navigate = Navigate + 1
                  REPLACE var_class WITH var
                  REPLACE name WITH B_action
            ELSE
                  Act_new = Act_new - 1
                  Navigate = Navigate - 1
            ENDIF

      CASE Navigate = 4
            RESTORE SCREEN FROM S_buf
            RETURN (B_action)
      ENDCASE

ENDDO
* end function Find_action

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function          :  Add_action                              -
* -    Input             :  none                                    -
* -    Output            :  sets Escape                             -
* -    Date              :  19 May 89                               -
* -    UpDate            :  25 Sep 89                               -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Synopses:
*           Adds new actions using new or old variable classes.
*     Pseudo:
*     While add more actions
*                 Decide the variable class
*                 While using the same variable class
*                       Add a action
*
*
FUNCTION Add_action
*
* display all of the possible variables
*
PRIVATE Var, Act_pnt, B_action,S_buf
SAVE SCREEN TO S_buf
PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            EXIT
      CASE Navigate = 1
```

```
            *find a Var_class
            RESTORE SCREEN FROM S_buf
            @ 1,39 TO 23,79 DOUBLE
            @ 2, 40 SAY 'Enter an action: '
            Var = Get_var()
            IF Var = 'NIL'
                  Navigate = Navigate - 1
            ELSE
                  Navigate = Navigate + 1
                  @ 3,40 SAY 'Var Class = '+Var
                  *list previous ones
                  Act_pnt = 0
                  Act_new = 0
                  SEEK Var
                  DO WHILE var_class = var
                        Act_pnt = Act_pnt + 1
                        @ 4+Act_pnt,55 SAY name
                        SKIP
                  ENDDO
            ENDIF

      CASE Navigate = 2
            * add a new action
            B_action = space(15)
            Act_new = Act_new + 1
            @ 4+Act_pnt+Act_new, 40 SAY 'Enter a name: ' GET B_action
            READ
            IF (len(trim(B_action)) < 1 .OR. lastkey() = 27)
                  Navigate = Navigate - 1
            ELSE
                  SEEK (var+B_action)
                  IF eof()
                        *does not already exists
                        @ 4+Act_pnt+Act_new, 55 SAY B_action
                        Navigate = Navigate + 1
                  ELSE
                        Act_new = Act_new - 1
                        ErrWait ('extenal enquiry already exists')
                  ENDIF
            ENDIF

      CASE Navigate = 3
            * add the rest of the enquiry
            IF Cmplt_action(B_action)
                  Navigate = Navigate + 1
                  REPLACE var_class WITH var
                  REPLACE name WITH B_action
            ELSE
                  Act_new = Act_new - 1
                  Navigate = Navigate - 1
            ENDIF

      CASE Navigate = 4
            * add a new object

            IF AskY("Enter another enquiry for the variable class
"+rtrim(Var)+'?')
                  @ 24,0 CLEAR
                  Navigate = 2      &&start at the name
            ELSE
                  Navigate = Navigate + 1
            ENDIF
```

```
      CASE Navigate = 5
            EXIT
      ENDCASE
ENDDO
RESTORE SCREEN FROM S_buf
IF Navigate > 0
      Escape = .F.
      RETURN(.T.)
ELSE
      Escape = .T.
      RETURN(.F.)
ENDIF
* end function Find_action

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function         :   Cmplt_action()                          -
* -    Input            :   none                                    -
* -    Output           :   error                                   -
* -    Date             :   02 Oct 89                               -
* -    UpDate           :   02 Oct 89                               -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Assumes the file is pointing to the record to add
FUNCTION Cmplt_action
PARAMETER Enqire
PRIVATE S_buf
SAVE SCREEN TO S_buf
PRIVATE B_token
B_token = space(8)
@ 15,40 CLEAR TO 22,78
@ 15,39 TO 15,79
@ 15,39 SAY " "
@ 15,79 SAY " "
@ 15, 55 SAY rtrim("Enq: "+Enqire)
@ 17,41 SAY "OS token:" GET B_token PICTURE "999999"
READ
IF (lastkey() # 27 .AND. updated())
      APPEND BLANK
      REPLACE token WITH b_token
      RESTORE SCREEN FROM S_buf
      RETURN (.T.)
ELSE
      RESTORE SCREEN FROM S_buf
      RETURN (.F.)
ENDIF




* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function         :   Act_var                                 -
* -    Input            :   none                                    -
* -    Output           :   variable class                          -
* -    Date             :   19 May 89                               -
* -    UpDate           :   25 Sep 89                               -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Act_var
SELECT 6    && action.dbf
SET INDEX TO action
PRIVATE B_var
B_var = Get_var()
SET INDEX  TO
```

```
RETURN (B_var)

*
*end function Act_var

*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project           :   KOOLA programming language                =
*  =    Sub-project       :   Entry for objects                         =
*  =    File              :   belief.prg                                =
*  =    Author            :   Robert D. Rourke                          =
*  =    Date              :   19 Mar 1989                               =
*  =    Update            :   10 Nov 1989                               =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
*     FUNCTIONS:
*           Add_belief
*           Get_belief
*           Bel_var
*           Get_var
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function          :   Get_belief                              -
*  -    Input             :                                           -
*  -    Output            :                                           -
*  -    Date              :   19 May 89                               -
*  -    UpDate            :   10 Nov 89                               -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Get_belief
PARAMETER Var          &&the var class to use
SELECT 3    && belief.dbf
SET INDEX TO belief
PRIVATE Blf_new, Blf_pnt, B_belief,S_buf
SAVE SCREEN TO S_buf
*
*make a list of previous beliefs
*
Blf_pnt = 0
Blf_new = 0
PRIVATE Blf_list
DECLARE Blf_list[50]
SEEK Var
DO WHILE var_class = var
      Blf_pnt = Blf_pnt + 1
      Blf_list[Blf_pnt] = name
      SKIP
ENDDO

PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            RESTORE SCREEN FROM S_buf
            RETURN('NUL')
      CASE Navigate = 1
            RESTORE SCREEN FROM S_buf
            @ 2,40 CLEAR TO 22, 78
            @ 1,39 TO 23,79 DOUBLE
            @ 2, 40 SAY 'Enter a belief: '
            @ 3,40 SAY 'Var Class = '+Var
            *
            *     make selection
            *
```

```
            PRIVATE i
            FOR i = 1 TO Blf_pnt
                  @ 4+i,55 PROMPT Blf_list[i]
            NEXT
            @ 4+i, 55 PROMPT '*NEW           '
            PRIVATE Select
            Select = Blf_pnt
            MENU TO Select
            DO CASE
            CASE Select = 0
                  Navigate = Navigate - 1
            CASE Select > Blf_pnt
                  Navigate = Navigate + 1
            OTHERWISE
                  RESTORE SCREEN FROM S_buf
                  RETURN (Blf_list[Select])
            ENDCASE

      CASE Navigate = 2
            * add a new belief
            B_belief = space(15)
            Blf_new = Blf_new + 1
            @ 4+Blf_pnt+Blf_new, 40 SAY 'Enter belief: ' GET B_belief
            READ
            IF (len(trim(B_belief)) < 1 .OR. lastkey() = 27)
                  Navigate = Navigate - 1
            ELSE
                  SEEK (var+B_belief)
                  IF eof()
                        *does not already exists
                        @ 4+Blf_pnt+Blf_new, 55 SAY B_belief
                        APPEND BLANK
                        REPLACE var_class WITH var
                        REPLACE name WITH B_belief
                        RESTORE SCREEN FROM S_buf
                        RETURN (B_belief)
                  ELSE
                        Blf_new = Blf_new - 1
                        ErrWait ('Belief name already exists')
                  ENDIF
            ENDIF
      ENDCASE
ENDDO
* end function Find_beleif

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Add_belief
*  -    Input           :   none
*  -    Output          :   sets Escape
*  -    Date            :   19 May 89
*  -    UpDate          :   25 Sep 89
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Synopses:
*           Adds new beliefs using new or old variable classes.
*     Pseudo:
*     While add more beliefs
*                 Decide the variable class
*                 While using the same variable class
*                       Add a belief
*
*
FUNCTION Add_belief
```

```
* alternate super class for goal
SELECT 3    && belief.dbf
SET INDEX TO belief
*
* display all of the possible variables
*
PRIVATE Var, Blf_pnt, B_belief,S_buf
SAVE SCREEN TO S_buf
PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            EXIT
      CASE Navigate = 1
            *find a Var_class
            RESTORE SCREEN FROM S_buf
            @ 1,39 TO 23,79 DOUBLE
            @ 2, 40 SAY 'Enter a belief: '
            Var = Get_var()
            IF Var = 'NIL'
                  Navigate = Navigate - 1
            ELSE
                  Navigate = Navigate + 1
                  @ 3,40 SAY 'Var Class = '+Var
                  *list previous ones
                  Blf_pnt = 0
                  Blf_new = 0
                  SEEK Var
                  DO WHILE var_class = var
                        Blf_pnt = Blf_pnt + 1
                        @ 4+Blf_pnt,55 SAY name
                        SKIP
                  ENDDO
            ENDIF

      CASE Navigate = 2
            * add a new belief
            B_belief = space(15)
            Blf_new = Blf_new + 1
            @ 4+Blf_pnt+Blf_new, 40 SAY 'Enter belief: ' GET B_belief
            READ
            IF (len(trim(B_belief)) < 1 .OR. lastkey() = 27)
                  Navigate = Navigate - 1
            ELSE
                  SEEK (var+B_belief)
                  IF eof()
                        *does not already exists
                        @ 4+Blf_pnt+Blf_new, 55 SAY B_belief
                        APPEND BLANK
                        REPLACE var_class WITH var
                        REPLACE name WITH B_belief
                        Navigate = Navigate + 1
                  ELSE
                        Blf_new = Blf_new - 1
                        ErrWait ('Belief name already exists')
                  ENDIF
            ENDIF


      CASE Navigate = 3
            * add a new object
```

```
            IF  AskY("Enter  another  belief  for  the  variable  class
"+rtrim(Var)+'?')
                  @ 24,0 CLEAR
                  Navigate = Navigate - 1
            ELSE
                  Navigate = Navigate + 1
            ENDIF

      CASE Navigate = 4
            EXIT
      ENDCASE
ENDDO
RESTORE SCREEN FROM S_buf
IF Navigate > 0
      Escape = .F.
      RETURN(.T.)
ELSE
      Escape = .T.
      RETURN(.F.)
ENDIF
* end function Find_belief

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Bel_var                                    -
*  -    Input           :   none                                       -
*  -    Output          :   variable class                             -
*  -    Date            :   19 May 89                                  -
*  -    UpDate          :   25 Sep 89                                  -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Bel_var
SELECT 3    && belief.dbf
SET INDEX TO belief
PRIVATE B_var
B_var = Get_var()
SET INDEX  TO
RETURN (B_var)

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Get_var
*  -    Input           :                                              -
*  -    Output          :   var
*  -    Date            :   25 Sep 89
*  -    UpDate          :          89
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Assumes: Database is index to var_class
FUNCTION Get_var
*
*     create a list of variables
*
PRIVATE Var_lst, Var_pnt, S_buf
SAVE SCREEN TO S_buf
Var_pnt = 0
DECLARE Vat_lst[50]
GO TOP
DO WHILE !eof()
      Var_pnt = Var_pnt + 1
      Vat_lst[Var_pnt] = var_class
      DO WHILE Vat_lst[Var_pnt] = var_class .AND. !eof()
            SKIP 1
      ENDDO
ENDDO
PRIVATE Navigate
```

```
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            RESTORE SCREEN FROM S_buf
            RETURN ('NIL')
      CASE Navigate = 1
            *
            * Make the choice
            *
            RESTORE SCREEN FROM S_buf
            @ 2,40 CLEAR TO 22, 78
            @ 1,39 TO 23,79
            PRIVATE i
            @3, 40 SAY 'Select a variable class:'
            FOR i = 1 TO Var_pnt
                  @ 4+i, 40 PROMPT Vat_lst[i]
            NEXT
            @ 4+i, 40 PROMPT '*NEW            '
            PRIVATE Select
            Select = Var_pnt
            MENU TO Select
            DO CASE
            CASE Select = 0
                  Navigate = Navigate - 1
            CASE Select > Var_pnt
                  Navigate = Navigate + 1
            OTHERWISE
                  RESTORE SCREEN FROM S_buf
                  RETURN (Vat_lst[Select])
            ENDCASE

      CASE Navigate = 2
            PRIVATE B_var
            B_var = space(15)
            @ 4+i, 40 Get B_var
            READ
            IF (len(trim(B_var)) < 1 .OR. lastkey() = 27)
                  Navigate = Navigate - 1
            ELSE
                  RESTORE SCREEN FROM S_buf
                  RETURN (lower(B_var))
            ENDIF
      ENDCASE
ENDDO
*return Get_var

*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project          :   KOOLA programming language               =
*  =    Sub-project      :   Entry for objects                        =
*  =    File             :   goal.prg                                 =
*  =    Author           :   Robert D. Rourke                         =
*  =    Date             :   06 Nov 1989                              =
*  =    Update           :   08 Nov 1989                              =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
*     FUNCTIONS:
*           Add_goal
*           Get_goal
*           goal_var

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
*  -    Function         :   Add_goal                                    -
*  -    Input            :   none                                        -
*  -    Output           :   sets Escape                                 -
*  -    Date             :   06 Nov 89                                   -
*  -    UpDate           :   08 Nov 89                                   -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*
FUNCTION Add_goal

PRIVATE S_buf
SAVE SCREEN TO S_buf
PRIVATE Iner_var,T_action, Goal_select, This_goal
PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            RESTORE SCREEN FROM S_buf
            RETURN (.F.)
      CASE Navigate = 1
            *find the name of the goal
            *display the old goal
            @ 24,0 CLEAR
            Goal_select =  Goal_name()
            This_goal = recno()
            IF Goal_select = 0
                  Navigate = Navigate - 1
            ELSE
                  @ 5, 0 SAY "GOAL: "+name
                  @ 6, 0 SAY 'FOR: '+var_class
                  @ 7, 0 SAY 'IF: '+belif+" "+operand
                  @ 7, 25 SAY prob
                  ??"%"
                  @ 8, 0 SAY 'THEN DO: '+then_do
                  @ 9, 0 SAY 'THEN CHAIN: '+then_chain
                  @10, 0 SAY 'ELSE DO: '+else_do
                  @11, 0 SAY 'ELSE CAHIN: '+else_chain
            ENDIF
            IF Goal_select = 1
                  *edit an old
                  Iner_var = var_class
                  *
                  *     Special check if it is porly defiend
                  *
                  IF empty(Iner_var)
                        Navigate = Navigate + 1
                  ELSE
                        Navigate = 10
                  ENDIF
            ELSEIF Goal_select = 2
                  *new
                  Navigate = Navigate + 1
            ENDIF
      CASE Navigate = 2
            *find the variable class for the goal
            *from the list of beilfs
            @ 6, 0 SAY 'FOR:  *
            SELECT 3    && belief.dbf
            SET INDEX TO belief
            Iner_var = Get_var()
            SELECT 2    && goal.dbf
            GO This_goal
```

```
      IF Iner_var = 'NIL'
            Navigate = Navigate - 1
            @ 6, 0 SAY 'FOR:   '
            SELECT 2    && goal.dbf
            SET INDEX TO goal
            GO This_goal
            DELETE
            PACK
            @ 5, 6 SAY space (20)
      ELSE
            REPLACE var_class WITH Iner_var
            @ 6, 0 SAY 'FOR: '+var_class
            Navigate = Navigate + 1
      ENDIF
CASE Navigate = 3
      *find the belief to base it on
      @ 7, 0 SAY 'IF:  *
      B_belief = Get_belief(Iner_var)
      SELECT 2    && goal.dbf
      GO This_goal
      IF B_belief = 'NUL'
            Navigate = Navigate - 1
            @ 7, 0 SAY 'IF:   '
      ELSE
            REPLACE belif WITH B_belief
            @ 7, 0 SAY 'IF: '+belif
            Navigate = Navigate + 1
      ENDIF

CASE Navigate = 4
      *find an operand
      SELECT 2    && goal.dbf
      GO This_goal
      @ 7, 0 SAY 'IF: '+belif GET operand
      @ 24, 0 CLEAR
      @ 24, 0 SAY "Enter an operand"
      READ
      @ 7, 0 SAY 'IF: '+belif+" "+operand
      IF lastkey() = 27
            Navigate = Navigate - 1
      ELSE
            Navigate = Navigate + 1
      ENDIF

CASE Navigate = 5
      *find the prob to compair to
      SELECT 2    && goal.dbf
      GO This_goal
      @ 7, 25 GET prob RANGE 0,100
      ??"%"
      @ 24, 0  CLEAR
      @ 24, 0 SAY "Enter the probability"
      READ
      @ 24, 0  CLEAR
      @ 7, 25 SAY prob
      IF lastkey() = 27
            Navigate = Navigate - 1
      ELSE
            Navigate = Navigate + 1
      ENDIF

CASE Navigate = 6
```

```
      *find the then action
      @ 8, 0 SAY 'THEN DO:  *                           '
      T_action = Get_action(Iner_var)
      SELECT 2    && goal.dbf
      IF T_action = 'NUL'
            Navigate = Navigate - 1
            @ 8, 0 SAY 'THEN DO:   '
      ELSE
            REPLACE then_do WITH T_action
            @ 8, 0 SAY 'THEN DO: '+then_do
            Navigate = Navigate + 1
      ENDIF

CASE Navigate = 7
      *find the then chain
      @ 9, 0 SAY 'THEN CHAIN:  *                            '
      T_nxgoal = Goal_name('FINISHED CHIAN')
      IF T_nxgoal = 0
            @ 9, 0 SAY 'THEN CHAIN:   '
            Navigate = Navigate - 1
      ELSE
            IF T_nxgoal = 3
                  t_buf = 'TERMINATE'
            ELSE
                  t_buf = name
            ENDIF
            SELECT 2    && goal.dbf
            GO This_goal
            REPLACE then_chain WITH t_buf
            @ 9, 0 SAY 'THEN CHAIN: '+then_chain
            Navigate = Navigate + 1
      ENDIF

CASE Navigate = 8
      *find the then action
      SELECT 2    && goal.dbf
      GO This_goal
      @10, 0 SAY 'ELSE DO:  *                                '
      T_action = Get_action(Iner_var)
      SELECT 2    && goal.dbf
      IF T_action = 'NUL'
            Navigate = Navigate - 1
            @10, 0 SAY 'ELSE DO:   '
      ELSE
            REPLACE else_do WITH T_action
            @10, 0 SAY 'ELSE DO: '+else_do
            Navigate = Navigate + 1
      ENDIF

CASE Navigate = 9
      *find the else chain
      @11, 0 SAY 'ELSE CAHIN:  *                                  '
      T_nxgoal = Goal_name('FINISHED CHIAN')
      IF T_nxgoal = 0
            Navigate = Navigate - 1
            @11, 0 SAY 'ELSE CAHIN:   '
      ELSE
            IF T_nxgoal = 3
                  t_buf = 'TERMINATE'
            ELSE
                  t_buf = name
            ENDIF
```

```
                  SELECT 2    && goal.dbf
                  GO This_goal
                  REPLACE else_chain WITH t_buf
                  @11, 0 SAY 'ELSE CAHIN: '+else_chain
                  Navigate = Navigate + 1
            ENDIF

      CASE Navigate = 10
            * finished
                  @23,0
                  WAIT "This is the complete goal, Pres Esc to change it."
                  IF lastkey() # 27
                        Navigate = Navigate + 1
                  ELSE
                        Navigate = Navigate - 1
                  ENDIF

      CASE Navigate = 11
            RESTORE SCREEN FROM S_buf
            RETURN(.T.)

      OTHERWISE
            ?'Falling out of the loop ::'
            ??Navigate
            WAIT
            EXIT
      ENDCASE
ENDDO

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function         :   goal_fact()                              -
*  -    Input            :   no input                                 -
*  -    Output           :   number enterered                         -
*  -    Date             :   06 Oct 89                                -
*  -    UpDate           :   27 Oct 89                                -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Convention: this is a member of Cmplt_goal, and should not be
*           called by any other function.
*     Assumes:    many variables and files and screen
*           and that the goal_type is FCT
*
FUNCTION goal_fact
PARAMETER Antc_pnt
*
*     Establish the buffers from the record or start as blank
*
@8, 0SAY "IF:"
IF Edit_flg
      FOR i = 0 TO Max_Antec-1
            Mac = 'if'+str(i,1)
            if_lst[i+1]=&Mac
            Mac = "oper"+str(i,1)
            oper_lst[i+1]=&Mac
            Mac = "extrn"+str(i,1)
            Extrn_lst[i+1]=&Mac
            Mac = "ans"+str(i,1)
            ans_lst[i+1]=&Mac
            Mac = "num"+str(i,1)
            num_lst[i+1]=&Mac
      NEXT
      FOR i = 1 TO Antc_pnt
            IF Extrn_lst[Antc_pnt]
```

```
                  @8+i, 5 SAY "Extern:"
            ELSE
                  @8+i, 5 SAY "Intern:"
            ENDIF
            @8+i,12 SAY if_lst[i]
            @8+i, 30 SAY oper_lst[i]
            IF !Extrn_lst[Antc_pnt] .AND. (len(trim(Ans_lst[i]))) > 1)
                  @8+i, 35 SAY Ans_lst[i]
            ELSE
                  @8+i, 35 SAY num_lst[i]
            ENDIF
      NEXT
      PRIVATE Navigate
      Navigate = 5
ELSE
      FOR i = 0 TO Max_Antec-1
            oper_lst[i+1]=space(2)
            ans_lst[i+1]=space(30)
            num_lst[i+1]=0.0
            Extrn_lst[i+1]=.F.
      NEXT
      PRIVATE Navigate
      Navigate = 1
ENDIF
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            *backed out
            IF Antc_pnt > 1
                  Antc_pnt = Antc_pnt - 1
                  Navigate = 4
            ELSE
                  SELECT 2    &&goal.dbf
                  SET INDEX TO goal
                  RETURN (0)
            ENDIF

      CASE Navigate = 1
            * find out what type of fact human or external
            @24, 0 CLEAR
            @8+Antc_pnt, 5 SAY "*              "
            @24, 0 SAY "Is this antecedent based on external facts?",
            GET Extrn_lst[Antc_pnt] &&a logic
            READ
            IF lastkey() # 27
                  Navigate = Navigate + 1
                  IF Extrn_lst[Antc_pnt]
                        @8+Antc_pnt, 5 SAY "Extern:"
                  ELSE
                        @8+Antc_pnt, 5 SAY "Intern:"
                  ENDIF
            ELSE
                  Navigate = Navigate - 1
            ENDIF

      CASE Navigate = 2
            * enter the next fact name
            @8+Antc_pnt, 12 CLEAR TO 8+Antc_pnt, 30
            IF Extrn_lst[Antc_pnt]
                  if_lst[Antc_pnt] = Get_extreq(Var_cls)
            ELSE
                  if_lst[Antc_pnt] = Get_internal(Var_cls)
```

```
            ENDIF
            IF if_lst[Antc_pnt]# 'NIL'
                  Navigate = Navigate + 1
                  @8+Antc_pnt,12 SAY if_lst[Antc_pnt]
            ELSE
                  Navigate = Navigate - 1
            ENDIF
      CASE Navigate = 3
            * get the operand of the fact
            @24,0 CLEAR
            @24,0 SAY "Enter an operand"
            @8+Antc_pnt, 30 GET oper_lst[Antc_pnt]
            READ
            IF lastkey() # 27
                  Navigate = Navigate + 1
                  @8+Antc_pnt, 30 SAY oper_lst[Antc_pnt]
            ELSE
                  @8+Antc_pnt, 30 CLEAR TO 8+Antc_pnt, 36
                  Navigate = Navigate - 1
            ENDIF
      CASE Navigate = 4
            * get the number/symbol of the fact
            @24,0 CLEAR
            @8+Antc_pnt, 35 CLEAR TO 8+Antc_pnt, 60
            IF Extrn_lst[Antc_pnt]
                  @8+Antc_pnt, 35 GET num_lst[Antc_pnt] PICTURE "99999.99"
                  READ
                  IF lastkey() # 27
                        @8+Antc_pnt, 35 SAY num_lst[Antc_pnt] PICTURE
"99999.99"
                        Navigate = Navigate + 1
                  ELSE
                        @8+Antc_pnt, 35 CLEAR TO 8+Antc_pnt, 60
                        Navigate = Navigate - 1
                  ENDIF
            ELSE
                  *human interface
                  PRIVATE text_ans, answ_text, answ_num, error_flag
                  text_ans = .T.
                  answ_text = ''
                  answ_num = 0
                  error_flag = .T.
                  DO Ans_inter WITH
Text_ans,Answ_text,Answ_num,Error_flag,;
                   Var_cls,if_lst[Antc_pnt]
                  IF Error_flag
                        Navigate = Navigate - 1
                  ELSE
                        IF Text_ans
                              ans_lst[Antc_pnt] = Answ_text
                              @8+Antc_pnt, 35 SAY Answ_text
                        ELSE
                              num_lst[Antc_pnt] = Answ_num
                              @8+Antc_pnt, 35 SAY Answ_num
                        ENDIF
                        Navigate = Navigate + 1
                  ENDIF
            ENDIF

      CASE Navigate = 5
            * if there is space get another antecedent
            @24,0 CLEAR
```

```
            IF Antc_pnt < Max_Antec
                  IF AskY ("Enter another antecedent?")
                        Antc_pnt = Antc_pnt + 1
                        Navigate = 1
                  ELSE
                        Navigate = Navigate + 1
                  ENDIF
            ELSE
                  @23,0
                  WAIT "Antecedent full, Pres Esc to change"
                  IF lastkey() # 27
                        Navigate = Navigate + 1
                  ELSE
                        Navigate = Navigate - 1
                  ENDIF
            ENDIF
      CASE Navigate = 6
                  EXIT
      ENDCASE
ENDDO
*
*     Do much saving
*
SELECT 2    &&goal.dbf
SET INDEX TO goal
IF !Edit_flg
      APPEND BLANK
      REPLACE goal_typ WITH 'FCT'
ELSE
      SEEK Var_cls+rul_name
ENDIF

FOR i = 0 TO Max_Antec-1
      Mac = 'if'+str(i,1)
      REPLACE &Mac WITH if_lst[i+1]
      Mac = "oper"+str(i,1)
      REPLACE &Mac WITH oper_lst[i+1]
      Mac = "extrn"+str(i,1)
      REPLACE &Mac WITH Extrn_lst[i+1]
      Mac = "ans"+str(i,1)
      REPLACE &Mac WITH ans_lst[i+1]
      Mac = "num"+str(i,1)
      REPLACE &Mac WITH num_lst[i+1]
      REPLACE num_antec WITH Antc_pnt
NEXT
RETURN(Antc_pnt)

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   goal_belief()                            -
*  -    Input           :   no input                                 -
*  -    Output          :   number enterered                         -
*  -    Date            :   06 Oct 89                                -
*  -    UpDate          :   27 Oct 89                                -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Convention: this is a member of Cmplt_goal, and should not be
*           called by any other function.
*     Assumes:    many variables and files and screen
*           and that the goal_type is FCT
*
FUNCTION goal_belief
PARAMETER Antc_pnt
*
```

```
*     Establish the buffers from the record or start as blank
*
@8, 0SAY "IF:"
IF Edit_flg
      b_then = then
      b_then_p = then_p
      b_then_w = then_w
      FOR i = 0 TO Max_Antec-1
            Mac = 'if'+str(i,1)
            if_lst[i+1]=&Mac
      NEXT
      FOR i = 1 TO Antc_pnt
            @8+i,8 SAY if_lst[i]
      NEXT
      @ 20, 7 SAY +b_then

      PRIVATE Navigate
      Navigate = 4
ELSE
      PRIVATE Navigate
      b_then_p = 0
      b_then_w = 0
      Navigate = 1
ENDIF
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            *backed out
            IF Antc_pnt > 1
                  Antc_pnt = Antc_pnt - 1
                  Navigate = 2
            ELSE
                  SELECT 2     &&goal.dbf
                  SET INDEX TO goal
                  RETURN (0)
            ENDIF

      CASE Navigate = 1
            * enter the next fact name
            @8+Antc_pnt, 8 SAY "*"+space(30)
            if_lst[Antc_pnt] = Get_belief(Var_cls)
            IF if_lst[Antc_pnt]# 'NUL'
                  Navigate = Navigate + 1
                  @8+Antc_pnt,8 SAY if_lst[Antc_pnt]
            ELSE
                  Navigate = Navigate - 1
            ENDIF

      CASE Navigate = 2
            * if there is space get another antecedent
            @24,0 CLEAR
            IF Antc_pnt < Max_Antec
                  IF AskY ("Enter another antecedent?")
                        IF lastkey() # 27
                              Antc_pnt = Antc_pnt + 1
                              Navigate = 1
                        ELSE
                              Navigate = Navigate - 1
                        ENDIF
                  ELSE
                        Navigate = Navigate + 1
                  ENDIF
```

```
            ELSE
                  @23,0
                  WAIT "Antecedent full, Pres Esc to change"
                  IF lastkey() # 27
                        Navigate = Navigate + 1
                  ELSE
                        Navigate = Navigate - 1
                  ENDIF
            ENDIF

      CASE Navigate = 3
            *the new belief
            b_then = Get_belief(Var_cls)
            IF b_then = 'NUL'
                  Navigate = Navigate - 1
            ELSE
                  Navigate = Navigate + 1
                  @ 20, 7 SAY +b_then
            ENDIF

      CASE Navigate = 4
            *the weight of the new belief
            @ 20, 25 SAY "prob:" GET b_then_p PICTURE "999"  RANGE 0, 100
            @ 20, 37 SAY "Weight:" GET b_then_w PICTURE "999" RANGE 0, 100
            READ
            IF lastkey() = 27
                  Navigate = Navigate - 1
            ELSE
                  Navigate = Navigate + 1
                  @ 20, 31 SAY b_then_p PICTURE "999"
                  @ 20, 45 SAY b_then_w PICTURE "999"
            ENDIF


      CASE Navigate = 5
                  EXIT
      ENDCASE
ENDDO
*
*     Do much saving
*
SELECT 2    &&goal.dbf
SET INDEX TO goal
IF !Edit_flg
      APPEND BLANK
      REPLACE goal_typ WITH 'BLF'
ELSE
      SEEK Var_cls+rul_name
ENDIF
REPLACE num_antec WITH Antc_pnt
FOR i = 0 TO Max_Antec-1
      Mac = 'if'+str(i,1)
      REPLACE &Mac WITH if_lst[i+1]
NEXT
REPLACE then WITH b_then
REPLACE then_p WITH b_then_p
REPLACE then_w WITH b_then_w

RETURN(Antc_pnt)
*end function goal_belief

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
*  -    Function          :   goal_name                                -
*  -    Input             :   none                                     -
*  -    Output            :   0- error 1- old 2-new                    -
*  -    Date              :   06 Oct 89                                -
*  -    UpDate            :   06 Oct 89                                -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Goal_name
PARAMETER Other_chs

PRIVATE S_buf
SAVE SCREEN TO S_buf

SELECT 2    && goal.dbf
SET INDEX TO goal
PRIVATE  Num_goal
Num_goal = reccount()
DECLARE Goal_list[Num_goal]
PRIVATE goal_pnt
goal_pnt = 0
GO TOP
DO WHILE !eof()
      goal_pnt = Goal_pnt + 1
      Goal_list[Goal_pnt] = name
      SKIP
ENDDO
PRIVATE i
DO WHILE .T.
      @ 2,40 CLEAR TO 22, 78
      @ 1,39 TO 23,79 DOUBLE
      @ 2, 40 SAY 'Enter a Goal: '
      FOR i = 1 TO Num_goal
            @ 4+i,55 PROMPT Goal_list[i]
      NEXT
      @ 5+goal_pnt, 55 PROMPT 'NEW           '
      IF pcount() > 0
            @ 6+goal_pnt, 55 PROMPT Other_chs
      ENDIF
      MENU TO Select
      DO CASE
      CASE Select = 0
            RESTORE SCREEN FROM S_buf
            RETURN (0)
      CASE Select <= goal_pnt
            *edit an old old
            RESTORE SCREEN FROM S_buf
            SEEK (Goal_list[Select])
            RETURN (1)
      CASE Select = goal_pnt+2
            RETURN (3)
      OTHERWISE
            * add a new goal
            B_goal = space(15)
            @ 5+goal_pnt, 40 SAY 'Enter a name: ' GET B_goal
            READ
            IF (len(trim(B_goal)) < 1 .OR. lastkey() = 27)
                  * loop around again
            ELSE
                  SEEK (B_goal)
                  IF eof()
                        *does not already exists
                        @ 5+goal_pnt, 40 CLEAR  TO 5+goal_pnt,55
                        @ 5+goal_pnt, 55 SAY B_goal
```

```
                        APPEND BLANK
                        REPLACE name WITH b_goal
                        RESTORE SCREEN FROM S_buf
                        RETURN (2)
                  ELSE
                        ErrWait ('goal already exists')
                        * loop around again
                  ENDIF
            ENDIF
      ENDCASE
ENDDO
*end function

*
*end file goal.prg

*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project          :   KOOLA programing language Shell           =
*  =    Sub-project      :   Entry for objects                         =
*  =    File             :   p_rule.prg                                =
*  =    Author           :   Robert D. Rourke                          =
*  =    Date             :   19 Nov 1989                               =
*  =    Update           :   19 Nov 1989                               =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function         :   Rept_mnu                                 -
*  -    Input            :   previous selection                       -
*  -    Output           :   new selection                            -
*  -    Date             :   19 Nov 1989                              -
*  -    UpDate           :   19 Nov 1989                              -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Print_rule
SELECT 1    &&rule.dbf
SET INDEX TO rule
PRIVATE Page_Message
Page_Message = 'KOOLA  Rule Listing  Date: '+dtoc(date())
PRIVATE S_buf
SAVE SCREEN TO S_buf
PRIVATE Left_Margin, Top_Margin, Bottom_Margin,
Left_Margin = 5
Top_Margin = 3
Bottom_Margin = 59

PRIVATE Line_counter,Page_Counter
Line_Counter = Top_Margin
Page_Counter = 1
PRIVATE B_var_class, B_name, R_count
GO TOP
SET PRINTER TO myout

@  4, 25 CLEAR TO 14, 53
@  4, 25 TO 14, 53
@  5, 28  SAY "RELEVE DE L'IMPRESSION"
@  9, 32  SAY 'PAGE     LIGNE'
*@  5, 31  SAY 'PRINTING STATUS'
*@  9, 32  SAY 'PAGE      LINE'
@ 11, 32  SAY '['
@ 11, 33  SAY '01'
@ 11, 35  SAY ']      ['
@ 11, 43  SAY '00'
@ 11, 45  SAY ']'
@  6, 29  TO 13, 49    DOUBLE
```

```
@  9, 39  TO 12, 39
@  8, 30  TO  8, 47


WriteLn ()
WriteLn ('KOOLA Production System      Rule Listing  '+;
 'Date: '+dtoc(date()) )
WriteLn ()
WriteLn ()

DO WHILE !eof()
      IF Line_Counter >  (Bottom_Margin - 6)
            NewPage()
      ENDIF
      B_var_class = var_class
      R_count = 1
      WriteLn ('FOR: '+B_var_class)
      WriteLn ()
      DO WHILE B_var_class = var_class .AND. !eof()
            IF rule_typ = "FCT"
                  fct_rule(R_count)
            ELSE
                  blf_rule(R_count)
            ENDIF
            WriteLn()
            R_count = R_count + 1
            SKIP + 1
      ENDDO
      WriteLn()
ENDDO

WriteLn ("End of report.")
EJECT
RESTORE SCREEN FROM S_buf
*end function go_print

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   fct_rule()                               -
*  -    Input           :   previous selection                       -
*  -    Output          :   new selection                            -
*  -    Date            :   19 Nov 1989                              -
*  -    UpDate          :   19 Nov 1989                              -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Fct_rule
PARAMETER Counter
WriteLn('Primary Rule ('+str(Counter,2)+'): '+name)
IF extrn0
      WriteLn('  IF  External: '+trim(if0)+' '+oper0+' '+str(num0,8,2))
ELSE
      IF  (len(trim(ans0)) > 1)
            WriteLn('   IF: '+trim(if0)+' '+oper0+' '+trim(ans0))
      ELSE
            WriteLn('      IF       Question:   '+trim(if0)+'   '+oper0+'
'+str(num0,8,2))
      ENDIF
ENDIF
PRIVATE i, Macro, Pointer,M_oper,M_if,M_ans
FOR i = 1 TO num_antec-1
      Pointer = str(i,1)
      Macro = 'extrn'+Pointer
      M_if = 'IF'+Pointer
      M_oper= 'OPER'+Pointer
```

```
      IF &Macro
            M_ans = 'NUM'+Pointer
            WriteLn('     AND    External:  '+trim(&M_if)+'  '+&M_oper+'
'+str(&M_ans,8,2))
      ELSE
            M_ans = 'ANS'+Pointer
            IF  (len(trim(&M_ans)) > 1)
                  WriteLn('        AND:   '+trim(&M_if)+'   '+&M_oper+'
'+trim(&M_ans))
            ELSE
                  M_ans = 'NUM'+Pointer
                  WriteLn('   AND  Question: '+trim(&M_if)+' '+&M_oper+'
'+str(&M_ans,8,2))
            ENDIF
      ENDIF
NEXT
WriteLn('   THEN: '+trim(then)+' '+str(then_p,3)+' Weight: '+str(then_w,3)
)
IF  (else # 'NUL')
      WriteLn('      ELSE:  '+trim(else)+'   '+str(else_p,3)+'   Weight:
'+str(else_w,3) )
ENDIF
*end function fct_rule

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -   Function         :  blf_rule()                                        -
* -   Input            :  previous selection                                -
* -   Output           :  new selection                                     -
* -   Date             :  19 Nov 1989                                       -
* -   UpDate           :  19 Nov 1989                                       -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION blf_rule
PARAMETER Counter
WriteLn('Secondary Rule ('+str(Counter,2)+'): '+name)
WriteLn('   IF: '+trim(if0))
PRIVATE i, Macro, Pointer,M_if
FOR i = 1 TO num_antec-1
      Pointer = str(i,1)
      M_if = 'IF'+Pointer
      WriteLn('   AND: '+trim(&M_if))
NEXT
WriteLn('   THEN: '+trim(then)+' '+str(then_p,3)+' Weight: '+str(then_w,3)
)
IF else# 'NUL'
      WriteLn('      ELSE:  '+trim(else)+'   '+str(else_p,3)+'   Weight:
'+str(else_w,3) )
ENDIF
*end function blf_rule

*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION WriteLn
PARAMETERS Chr_String
*
*  Write a line to the default device.  use the global variables :
*  Left_Margin, Top_Margin, Bottom_Margin,Line_Counter, Page_Counter
*
Line_Counter = Line_Counter + 1
*IF Line_Counter >  (Bottom_Margin - 2)
IF Line_Counter >  Bottom_Margin
      NewPage()
ENDIF
@ 11, 43 SAY Line_Counter PICTURE '99'
```

```
SET DEVICE TO PRINT
IF pcount() > 0
      @ Line_Count, Left_Margin  SAY Chr_String
ENDIF
SET DEVICE TO SCREEN
*end procedure WriteLn

*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION NewPage
*
* starts a new Page for printing
*
PRIVATE Page_String
Page_Counter = Page_Counter + 1
Page_String = str(Page_Count,2,0)
@ 11, 33  SAY Page_Count PICTURE '99'
SET DEVICE TO PRINT
*@ Bottom_Margin, 65 SAY '.../'+Page_String
Line_Counter = Top_Margin
@ Line_Counter, Left_Margin SAY Page_Message+;
space(20)+'Page: '+Page_String
SET DEVICE  TO SCREEN
Line_Counter = Line_Counter + 2
*end procedure NewPage
*
*end file p_rule.prg

*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project          :  KOOLA programming language                     =
*  =    Sub-project      :  Entry for objects                              =
*  =    File             :  extern.prg                                     =
*  =    Author           :  Robert D. Rourke                               =
*  =    Date             :  19 Mar 1989                                    =
*  =    Update           :  10 Oct 1989                                    =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
*     FUNCTIONS:
*           Add_extreq
*           Get_extreq
*           exr_var
*           Get_var
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function         :  Add_extreq                                   -
*  -    Input            :                                               -
*  -    Output           :                                               -
*  -    Date             :  19 May 89                                    -
*  -    UpDate           :  02 Sep 89                                    -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Get_extreq
PARAMETER Var          &&the var class to use
SELECT 5    && extreq.dbf
SET INDEX TO extreq
PRIVATE Exr_new, Exr_pnt, B_extreq,S_buf
SAVE SCREEN TO S_buf
*
*make a list of previous extreqs
*
Exr_pnt = 0
Exr_new = 0
PRIVATE Exr_list
DECLARE Exr_list[50]
SEEK Var
```

```
DO WHILE var_class = var
      Exr_pnt = Exr_pnt + 1
      Exr_list[Exr_pnt] = name
      SKIP
ENDDO

PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            RESTORE SCREEN FROM S_buf
            RETURN('NIL')
      CASE Navigate = 1
            RESTORE SCREEN FROM S_buf
            @ 1,39 TO 23,79 DOUBLE
            @ 2, 40 SAY 'Enter a extreq: '
            @ 3,40 SAY 'Var Class = '+Var
            *
            *     make selection
            *
            PRIVATE i
            FOR i = 1 TO Exr_pnt
                  @ 4+i,55 PROMPT Exr_list[i]
            NEXT
            @ 4+i, 55 PROMPT '*NEW            '
            PRIVATE Select
            Select = Exr_pnt
            MENU TO Select
            DO CASE
            CASE Select = 0
                  Navigate = Navigate - 1
            CASE Select > Exr_pnt
                  Navigate = Navigate + 1
            OTHERWISE
                  RESTORE SCREEN FROM S_buf
                  RETURN (Exr_list[Select])
            ENDCASE

      CASE Navigate = 2
            * add a new extreq
            B_extreq = space(15)
            Exr_new = Exr_new + 1
            @ 4+Exr_pnt+Exr_new, 40 SAY 'Enter extreq: ' GET B_extreq
            READ
            IF ( !updated()  .OR. lastkey() = 27)
                  Exr_new = Exr_new - 1
                  Navigate = Navigate - 1
            ELSE
                  SEEK (var+B_extreq)
                  IF eof()
                        *does not already exists
                        @ 4+Exr_pnt+Exr_new, 55 SAY B_extreq
                        Navigate = Navigate + 1
                  ELSE
                        Exr_new = Exr_new - 1
                        ErrWait ('extreq name already exists')
                  ENDIF
            ENDIF

      CASE Navigate = 3
            * add the rest of the enquiry
```

```
            IF Cmplt_extreq(B_extreq)
                  Navigate = Navigate + 1
                  REPLACE var_class WITH var
                  REPLACE name WITH B_extreq
            ELSE
                  Exr_new = Exr_new - 1
                  Navigate = Navigate - 1
            ENDIF

      CASE Navigate = 4
            RESTORE SCREEN FROM S_buf
            RETURN (B_extreq)
      ENDCASE

ENDDO
* end function Find_extreq

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Add_extreq                               -
*  -    Input           :   none                                     -
*  -    Output          :   sets Escape                              -
*  -    Date            :   19 May 89                                -
*  -    UpDate          :   25 Sep 89                                -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Synopses:
*           Adds new extreqs using new or old variable classes.
*     Pseudo:
*     While add more extreqs
*                 Decide the variable class
*                 While using the same variable class
*                       Add a extreq
*
*
FUNCTION Add_extreq
*
* display all of the possible variables
*
PRIVATE Var, Exr_pnt, B_extreq,S_buf
SAVE SCREEN TO S_buf
PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            EXIT
      CASE Navigate = 1
            *find a Var_class
            RESTORE SCREEN FROM S_buf
            @ 1,39 TO 23,79 DOUBLE
            @ 2, 40 SAY 'Enter a external enquiry: '
            Var = Get_var()
            IF Var = 'NIL'
                  Navigate = Navigate - 1
            ELSE
                  Navigate = Navigate + 1
                  @ 3,40 SAY 'Var Class = '+Var
                  *list previous ones
                  Exr_pnt = 0
                  Exr_new = 0
                  SEEK Var
                  DO WHILE var_class = var
                        Exr_pnt = Exr_pnt + 1
```

```
                @ 4+Exr_pnt,55 SAY name
                SKIP
            ENDDO
        ENDIF

    CASE Navigate = 2
        * add a new extreq
        B_extreq = space(15)
        Exr_new = Exr_new + 1
        @ 4+Exr_pnt+Exr_new, 40 SAY 'Enter a name: ' GET B_extreq
        READ
        IF (len(trim(B_extreq)) < 1 .OR. lastkey() = 27)
            Navigate = Navigate - 1
        ELSE
            SEEK (var+B_extreq)
            IF eof()
                *does not already exists
                @ 4+Exr_pnt+Exr_new, 55 SAY B_extreq
                Navigate = Navigate + 1
            ELSE
                Exr_new = Exr_new - 1
                ErrWait ('extenal enquiry already exists')
            ENDIF
        ENDIF

    CASE Navigate = 3
        * add the rest of the enquiry
        IF Cmplt_extreq(B_extreq)
            Navigate = Navigate + 1
            REPLACE var_class WITH var
            REPLACE name WITH B_extreq
        ELSE
            Exr_new = Exr_new - 1
            Navigate = Navigate - 1
        ENDIF

    CASE Navigate = 4
        * add a new object

        IF AskY("Enter another enquiry for the variable class
"+rtrim(Var)+'?')
            @ 24,0 CLEAR
            Navigate = 2        &&start at the name
        ELSE
            Navigate = Navigate + 1
        ENDIF

    CASE Navigate = 5
        EXIT
    ENDCASE
ENDDO
RESTORE SCREEN FROM S_buf
IF Navigate > 0
    Escape = .F.
    RETURN(.T.)
ELSE
    Escape = .T.
    RETURN(.F.)
ENDIF
* end function Find_extreq

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
*  -    Function         :   Cmplt_extreq()                              -
*  -    Input            :   none                                        -
*  -    Output           :   error                                       -
*  -    Date             :   02 Oct 89                                   -
*  -    UpDate           :   02 Oct 89                                   -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Assumes the file is pointing to the record to add
FUNCTION Cmplt_extreq
PARAMETER Enqire
PRIVATE S_buf
SAVE SCREEN TO S_buf
PRIVATE B_shelf_life
B_shelf_life = 0
PRIVATE B_token
B_token = space(8)
@ 15,40 CLEAR TO 22,78
@ 15,39 TO 15,79
@ 15,39 SAY " "
@ 15,79 SAY " "
@ 15, 55 SAY rtrim("Enq: "+Enqire)
@ 17,41 SAY "OS token:" GET B_token PICTURE "999999"
@ 19, 41 SAY "Self life (seconds):" GET B_shelf_life RANGE 0,10000000
READ
IF (lastkey() # 27 .AND. update())
      APPEND BLANK
      REPLACE token WITH b_token
      REPLACE shelf_life WITH b_shelf_life
      RESTORE SCREEN FROM S_buf
      RETURN (.T.)
ELSE
      RESTORE SCREEN FROM S_buf
      RETURN (.F.)
ENDIF




*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function         :   exr_var
*  -    Input            :   none                                        -
*  -    Output           :   variable class                              -
*  -    Date             :   19 May 89                                   -
*  -    UpDate           :   25 Sep 89
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION exr_var
SELECT 5    && extreq.dbf
SET INDEX TO extreq
PRIVATE B_var
B_var = Get_var()
SET INDEX  TO
RETURN (B_var)

*
*end function exr_var

*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project          :   KOOLA programming language
*  =    Sub-project      :   Entry for objects
*  =    File             :   internal.prg
*  =    Author           :   Robert D. Rourke
*  =    Date             :   02 Oct 1989
```

```
*  =    Update           :   23 Oct 1989                                  =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
*     FUNCTIONS:
*           Add_internal
*           Get_internal
*           inr_var
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Procedure        :   Ans_inter                                    -
*  -    Input            :                                                -
*  -    Output           :                                                -
*  -    Date             :   23 Oct 89                                    -
*  -    UpDate           :   23 Oct 89                                    -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
PROCEDURE Ans_inter
PARAMETER    Text_ans,;  &&return logic of type
             Answ_text,; &&if text answer stored here
             Answ_num,;  &&if numeric stored here
             Error_flg,;
             Var,;       &&variable class
             Qu_name          &&question name

PRIVATE S_buf
SAVE SCREEN TO S_buf
@ 2,40 CLEAR TO 22, 78
@ 1,39 TO 23,79 DOUBLE
@ 2, 40 SAY 'Enter an answere: '
@ 3,40 SAY 'Var Class = '+Var
@ 4, 40 SAY 'Name ='+Qu_name
SELECT 4    && internal.dbf
SET INDEX TO internal
SEEK Var+Qu_name
IF eof()
      ? 'Internal error question missing'
      WAIT
      Error_flg = .T.
      RESTORE SCREEN FROM S_buf
      RETURN
ENDIF

IF inquir_typ = 'NUM'
      Answ_num = lower
      @15, 41 GET Answ_num RANGE lower, upper
      READ
      IF lastkey() = 27
            Error_flg = .T.
            RESTORE SCREEN FROM S_buf
            RETURN
      ENDIF
      Text_ans = .F.
ELSE
      PRIVATE i, Mac_st, Ans_buf
      DECLARE Ans_buf[num_ams]
      FOR i = 0 TO num_ams - 1
            Mac_st = 'ANS'+str(i,1)
            Ans_buf[i+1] = &Mac_st
            @ 15+i, 42 PROMPT Ans_buf[i+1]
      NEXT
      PRIVATE Select
      MENU TO Select
      IF Select = 0
            Error_flg = .T.
```

```
            RESTORE SCREEN FROM S_buf
            RETURN
      ELSE
            Answ_text = Ans_buf[Select]
            Text_ans = .T.
      ENDIF
ENDIF
Error_flg = .F.
RESTORE SCREEN FROM S_buf
RETURN
*end procedure Ans_inter

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function         :   Get_internal                          -
*  -    Input            :                                         -
*  -    Output           :                                         -
*  -    Date             :   02 Oct 89                             -
*  -    UpDate           :   02 Oct 89                             -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Get_internal
PARAMETER Var           &&the var class to use
SELECT 4    && internal.dbf
SET INDEX TO internal
PRIVATE Inr_new, Inr_pnt, B_internal,S_buf
SAVE SCREEN TO S_buf
*
*make a list of previous internals
*
Inr_pnt = 0
Inr_new = 0
PRIVATE Inr_list
DECLARE Inr_list[50]
SEEK Var
DO WHILE var_class = var
      Inr_pnt = Inr_pnt + 1
      Inr_list[Inr_pnt] = name
      SKIP
ENDDO

PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            RESTORE SCREEN FROM S_buf
            RETURN('NIL')
      CASE Navigate = 1
            RESTORE SCREEN FROM S_buf
            @ 2,40 CLEAR TO 22, 78
            @ 1,39 TO 23,79 DOUBLE
            @ 2, 40 SAY 'Enter an internal: '
            @ 3,40 SAY 'Var Class = '+Var
            *
            *     make selection
            *
            PRIVATE i
            FOR i = 1 TO Inr_pnt
                  @ 4+i,55 PROMPT Inr_list[i]
            NEXT
            @ 4+i, 55 PROMPT 'NEW              '
            PRIVATE Select
            Select = Inr_pnt
```

```
        MENU TO Select
        DC CASE
        CASE Select = 0
             Navigate = Navigate - 1
        CASE Select > Inr_pnt
             Navigate = Navigate + 1
        OTHERWISE
             RESTORE SCREEN FROM S_buf
             RETURN (Inr_list[Select])
        ENDCASE

    CASE Navigate = 2
        * add a new internal
        B_internal = space(15)
        Inr_new = Inr_new + 1
        @ 4+Inr_pnt+Inr_new, 40 SAY 'Enter a name: ' GET B_internal
        READ
        IF ( !updated()  .OR. lastkey() = 27)
             Inr_new = Inr_new - 1
             Navigate = Navigate - 1
        ELSE
             SEEK (var+B_internal)
             IF eof()
                  *does not already exists
                  @ 4+Inr_pnt+Inr_new, 55 SAY B_internal
                  Navigate = Navigate + 1
             ELSE
                  Inr_new = Inr_new - 1
                  ErrWait ('internal name already exists')
             ENDIF
        ENDIF

    CASE Navigate = 3
        * add the rest of the enquiry
        IF Cmplt_internal(B_internal,.F.)   &&not edit mode
             Navigate = Navigate + 1
             REPLACE var_class WITH var
             REPLACE name WITH B_internal
        ELSE
             Inr_new = Inr_new - 1
             Navigate = Navigate - 1
        ENDIF

    CASE Navigate = 4
        RESTORE SCREEN FROM S_buf
        RETURN (B_internal)
    ENDCASE

ENDDO
* end function Find_internal

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :  Add_internal                                  -
*  -    Input           :  none                                         -
*  -    Output          :  sets Escape                                  -
*  -    Date            :  02 Oct 89                                    -
*  -    UpDate          :  02 Oct 89                                    -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Synopses:
*          Adds new internals using new or old variable classes.
*     Pseudo:
*     While add more internals
```

```
*                 Decide the variable class
*                 While using the same variable class
*                       Add a internal
*
*
FUNCTION Add_internal
*
* display all of the possible variables
*
PRIVATE Var, Inr_pnt, B_internal,S_buf
SAVE SCREEN TO S_buf
PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            EXIT
      CASE Navigate = 1
            *find a Var_class
            RESTORE SCREEN FROM S_buf
            @ 1,39 TO 23,79 DOUBLE
            @ 2, 40 SAY 'Enter an internal inquriry: '
            Var = Get_var()
            IF Var = 'NIL'
                  Navigate = Navigate - 1
            ELSE
                  PRIVATE Select
                  Select = 99 &&used by the next step
                  PRIVATE Inr_list, Inr_pnt
                  DECLARE Inr_list[50]
                  Navigate = Navigate + 1
                  @ 3,40 SAY 'Var Class = '+Var
*                 *list previous ones
*                 Inr_pnt = 0
*                 Inr_new = 0
*                 SEEK Var
*                 DO WHILE var_class = var
*                       Inr_pnt = Inr_pnt + 1
*                       @ 4+Inr_pnt,55 SAY name
*                       Inr_list[Clas_pnt] = name
*                       SKIP
*                 ENDDO
            ENDIF

      CASE Navigate = 2
            PRIVATE Inr_pnt
            Inr_pnt = 0
            SEEK Var
            DO WHILE var_class = var
                  Inr_pnt = Inr_pnt + 1
                  Inr_list[Inr_pnt] = name
                  @ 4+Inr_pnt,55 PROMPT Inr_list[Inr_pnt]
                  SKIP
            ENDDO
            @ 5+Inr_pnt, 55 PROMPT 'NEW            '
            MENU TO Select
            DO CASE
            CASE Select = 0
                  Navigate = Navigate - 1
            CASE Select <= Inr_pnt
                  *edit an old old
                  SEEK var+Inr_list[Select]
```

```
                  IF Cmplt_internal(Inr_list[Select],.T.)&& edit mode
                        Navigate = Navigate - 1
                  ELSE
                        *back to this one
                  ENDIF
            OTHERWISE
                  * add a new internal
                  B_internal = space(15)
                  @ 5+Inr_pnt, 40 SAY 'Enter a name: ' GET B_internal
                  READ
                  IF (len(trim(B_internal)) < 1 .OR. lastkey() = 27)
                        Navigate = Navigate - 1
                  ELSE
                        SEEK (var+B_internal)
                        IF eof()
                              *does not already exists
                              @ 5+Inr_pnt, 40 CLEAR TO 5+Inr_pnt,55
                              @ 5+Inr_pnt, 55 SAY B_internal
                              Navigate = Navigate + 1
                        ELSE
                              ErrWait ('extenal enquiry already exists')
                        ENDIF
                  ENDIF
            ENDCASE

      CASE Navigate = 3
            * add the rest of the enquiry
            IF Cmplt_internal(B_internal,.F.)   &&not edit mode
                  Navigate = Navigate + 1
                  REPLACE var_class WITH var
                  REPLACE name WITH B_internal
            ELSE
                  Navigate = Navigate - 1
            ENDIF

      CASE Navigate = 4
            * add a new object

            IF   AskY("Enter   another   enquiry   for   the   variable   class
"+rtrim(Var)+'?')
                  @ 24,0 CLEAR
                  Select = 99
                  Navigate = 2     &&start at the name
            ELSE
                  Navigate = Navigate + 1
            ENDIF

      CASE Navigate = 5
            EXIT
      ENDCASE
ENDDO
RESTORE SCREEN FROM S_buf
IF Navigate > 0
      Escape = .F.
      RETURN(.T.)
ELSE
      Escape = .T.
      RETURN(.F.)
ENDIF
* end function Find_internal

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
*  -   Function        :  Cmplt_internal()                       -
*  -   Input           :  Enquiry name, edit flag                -
*  -   Output          :  error                                  -
*  -   Date            :  02 Oct 89                              -
*  -   UpDate          :  02 Oct 89                              -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Assumes the file is pointing to the record to add
FUNCTION Cmplt_internal
PARAMETER Enqire, Edit_flg
PRIVATE S_buf
SAVE SCREEN TO S_buf
*
*     load the buffers
*
PRIVATE Text_based
PRIVATE B_question, Ans_lst, i
DECLARE Ans_lis [7]
PRIVATE B_shelf_life
PRIVATE B_lower, B_upper
IF Edit_flg
      Text_based = (inquir_typ='TEX')
      B_shelf_life =    shelf_life
      B_question = subst(question+space(60),1,60)
      IF Text_based
            PRIVATE Mac
            FOR i = 0 To 6
                  Mac = "ANS"+str(i,1)
                  Ans_lis[i+1]=subst(&Mac+space(30),1,30)
            NEXT
      ELSE
            B_upper = upper
            B_lower = lower
      ENDIF

ELSE
      Text_based = AskY (" Does this question have a texed based answer?")
      @24, 0 CLEAR
      B_shelf_life = 0
      B_question = space (60)
      IF Text_based
            FOR i = 1 TO 7
                  Ans_lis [i] = space (30)
            NEXT
      ELSE
            B_upper = 0
            B_lower = 0
      ENDIF
ENDIF

@ 8,0 CLEAR TO 22,77
@ 7,4 TO 22,77 DOUBLE
@ 7, 10 SAY "[ "+trim(Enqire)+" ]"
@ 22, 45 SAY 'Press PgDn when finished'
IF Text_based .AND. !In_colour
      SET COLOUR TO W/N,U/N
ENDIF
@ 9, 30 SAY "Shelf life:"GET B_shelf_life
@ 12, 5 SAY "Question :" GET B_question
IF Text_based
      FOR i = 1 TO 7
            @ 13+i, 30 SAY "Answer "+str(i,1) GET Ans_lis[i]
      NEXT
```

```
ELSE
      @ 14, 30 SAY "Lower bound:"GET B_lower
      @ 16, 30 SAY "Upper bound:"GET B_upper
ENDIF
READ
IF Text_based .AND. !In_colour
      SET COLOUR TO
ENDIF
IF (lastkey() # 27 .AND. updated())
      IF !Edit_flg
            APPEND BLANK
      ENDIF
      REPLACE shelf_life WITH B_shelf_life
      REPLACE question WITH B_question
      IF Text_based
            REPLACE inquir_typ WITH 'TEX'
            i = 0
            DO WHILE (len(trim(Ans_lis[i+1])) > 0)
                  Mac = "ANS"+str(i,1)
                  REPLACE &Mac WITH Ans_lis[i+1]
                  i = i + 1
                  IF i = 7
                        EXIT
                  ENDIF
            ENDDO
            REPLACE num_ams WITH i
      ELSE
            REPLACE inquir_typ WITH 'NUM'
            REPLACE lower WITH B_lower
            REPLACE upper WITH B_upper
      ENDIF
      RESTORE SCREEN FROM S_buf
      RETURN (.T.)
ELSE
      RESTORE SCREEN FROM S_buf
      RETURN (.F.)
ENDIF


* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function       :   Inr_var                                     -
* -    Input          :   none                                        -
* -    Output         :   variable class                              -
* -    Date           :   02 Oct 89                                   -
* -    UpDate         :   02 Oct 89                                   -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Inr_var
SELECT 4     && internal.dbf
SET INDEX TO internal
PRIVATE B_var
B_var = Get_var()
SET INDEX   TO
RETURN (B_var)

*
*end function Inr_var

* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* =    Project        :   KOOLA programing language Shell             =
* =    Sub-project    :   Entry for objects                           =
* =    File           :   p_rule.prg                                  =
```

```
*  =    Author          :    Robert D. Rourke                              =
*  =    Date            :    12 Dec 1989                                   =
*  =    Update          :    12 Dec 1989                                   =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :    Print_internal                              -
*  -    Input           :    previous selection                          -
*  -    Output          :    new selection                               -
*  -    Date            :    12 Dec 1989                                 -
*  -    UpDate          :    12 Dec 1989                                 -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Print_internal
SELECT 4    &&internal.dbf
SET INDEX TO internal
PRIVATE Page_Message
Page_Message = 'KOOLA  Question  Listing  Date:  '+dtoc(date())
PRIVATE S_buf
SAVE SCREEN TO S_buf
PRIVATE Left_Margin, Top_Margin, Bottom_Margin,
Left_Margin = 5
Top_Margin = 3
Bottom_Margin = 59

PRIVATE Line_counter,Page_Counter
Line_Counter = Top_Margin
Page_Counter = 1
PRIVATE B_var_class, B_name, R_count
GO TOP

@  4,  25 CLEAR TO 14, 53
@  4,  25 TO 14, 53
@  5,  28  SAY "RELEVE DE L'IMPRESSION"
@  9,  32  SAY 'PAGE     LIGNE'
*@  5,  31  SAY 'PRINTING STATUS'
*@  9,  32  SAY 'PAGE      LINE'
@ 11,  32  SAY '['
@ 11,  33  SAY  '01'
@ 11,  35  SAY ']       ['
@ 11,  43  SAY  '00'
@ 11,  45  SAY ']'
@  6,  29  TO  13, 49    DOUBLE
@  9,  39  TO  12, 39
@  8,  30  TO   8, 47

WriteLn ()
WriteLn ('KOOLA Production  System      Question  Listing   '+;
  'Date: '+dtoc(date()) )
WriteLn ()
WriteLn ()

DO WHILE !eof()
      B_var_class = var_class
      R_count = 1
      IF Line_Counter >  (Bottom_Margin - 6)
            NewPage()
      ENDIF
      WriteLn ('FOR: '+B_var_class)
      WriteLn ()
      DO WHILE B_var_class = var_class .AND. !eof()
            IF inquir_typ = "TEX"
                  Txt_internal(R_count)
```

```
          ELSE
               Num_internal(R_count)
          ENDIF
          WriteLn()
          R_count = R_count + 1
          SKIP + 1
     ENDDO
     IF !eof()
          WriteLn()
          WriteLn('------------------------')
          WriteLn()
     ENDIF
ENDDO

WriteLn ("End of report.")
EJECT
RESTORE SCREEN FROM S_buf
*end function go_print

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -   Function        :   Tex_internal()                            -
*  -   Input           :   previous selection                        -
*  -   Output          :   new selection                             -
*  -   Date            :   12 Dec 1989                               -
*  -   UpDate          :   12 Dec 1990                               -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Txt_internal
PARAMETER Counter

WriteLn('Question ('+str(Counter,2)+'): '+name)
WriteLn('   '+question)
PRIVATE Macro
FOR i = 0 TO num_ams - 1
     Macro = 'ANS'+str(i,1)
     WriteLn('                 '+&Macro)
NEXT
*end function Txt_internal

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -   Function        :   Num_internal()                          -
*  -   Input           :   previous selection                      -
*  -   Output          :   new selection                           -
*  -   Date            :   12 Dec 1989                             -
*  -   UpDate          :   12 Dec 1990                             -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Num_internal
PARAMETER Counter
WriteLn('Question ('+str(Counter,2)+'): '+name)
WriteLn('   '+question)
WriteLn('   Ans: from '+str(lower)+' to '+str(upper))
*end function Num_internal


*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =   Project         :   KOOLA programming language                  =
*  =   Sub-project     :   Entry for objects                           =
*  =   File            :   rule.prg                                    =
*  =   Author          :   Robert D. Rourke                            =
*  =   Date            :   06 Oct 1989                                 =
*  =   Update          :   03 Jan 1990                                 =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
```

```
*    FUNCTIONS:
*         Add_rule
*         Get_rule
*         rule_var

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function         :    Rule_consq()                              -
* -    Input            :    no input                                  -
* -    Output           :    number enterered                          -
* -    Date             :    27 Oct 89                                 -
* -    UpDate           :    06 Nov 89                                 -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*    Convention: this is a member of Cmplt_rule, and should not be
*         called by any other function.
*    Assumes:    many variables and files and screen
*         and that the rule_type is FCT
*    Function: adds an then and an else to a rule
FUNCTION Rule_consq

*
*    Creat the buffers
*
PRIVATE b_then, b_eles
PRIVATE Navigate
IF Edit_flg
     b_then = then
     b_then_p = then_p
     b_then_w = then_w
     @ 20, 25 SAY "prob: "
     ??then_p
     @ 20, 37 SAY "Weight: "
     ??then_w
     b_else = else
     b_else_p = else_p
     b_else_w = else_w
     * put them on the sreen
     @ 20, 0 SAY 'THEN:  '+b_then
     IF B_else # 'NUL'
          @21, 0 SAY 'ELSE:  '+b_else
          @ 21, 25 SAY "prob: "
          ??else_p
          @ 21, 37 SAY "Weight: "
          ??else_w
          Navigate = 4
     ELSE
          Navigate = 5
     ENDIF

ELSE
     b_then = 'NUL'
     b_then_w = 0
     b_then_p = 0
     b_else = 'NUL'
     b_else_w = 0
     b_else_p = 0
     Navigate = 1
ENDIF

DO WHILE .T.
     DO CASE
     CASE Navigate = 0
          * backed outs
```

```
      SELECT 1    &&rule.dbf
      SET INDEX TO rule
      RETURN (.F.)

CASE Navigate = 1
      * add the then
      @ 20,  0 CLEAR
      @ 20,  0 SAY 'THEN:   '
      b_then = Get_belief(Var_cls)
      IF b_then = 'NUL'
            Navigate = Navigate - 1
      ELSE
            Navigate = Navigate + 1
            @ 20,  7  SAY +b_then
      ENDIF

CASE Navigate = 2
      *add the weight
      @ 20,  25 SAY "prob:" GET b_then_p PICTURE "999"  RANGE 0,  100
      @ 20,  37 SAY "Weight:" GET b_then_w PICTURE "999" RANGE 0,  100
      READ
      IF lastkey()  = 27
            Navigate = Navigate - 1
      ELSE
            Navigate = Navigate + 1
            @ 20,  31  SAY b_then_p PICTURE "999"
            @ 20,  45  SAY b_then_w PICTURE "999"
      ENDIF

CASE Navigate = 3
      * add the else
      @ 21,  0 CLEAR
      IF AskN ('Is there an ELSE clause in the consequence?')
            b_else = Get_belief(Var_cls)
            IF b_else # 'NUL'
                  Navigate = Navigate + 1
                  @ 21, 0  SAY  ELSE:  '+b_else
            ENDIF
      ELSE
            IF lastkey() = 27
                  Navigate = Navigate - 1
            ELSE
                  Navigate = Navigate + 2
            ENDIF
      ENDIF

CASE Navigate = 4
      *add the weight
      @ 21,  25 SAY "prob:" GET b_else_p PICTURE "999"  RANGE 0,  100
      @ 21,  37 SAY "Weight:" GET b_else_w PICTURE "999" RANGE 0,  100
      READ
      IF lastkey()  = 27
            Navigate = Navigate - 1
      ELSE
            Navigate = Navigate + 1
            @ 21,  31  SAY b_else_p PICTURE "999"
            @ 21,  45  SAY b_else_w PICTURE "999"
      ENDIF

CASE Navigate = 5
      *last  chance
      @ 24,  0 CLEAR
```

```
      IF Var = 'NIL'
            Navigate = Navigate - 1
      ELSE
            PRIVATE Select
            Select = 99 &&used by the next step
            PRIVATE rule_list
            DECLARE rule_list[50]
            Navigate = Navigate + 1
            @ 3,40 SAY 'Var Class = '+Var
      ENDIF

CASE Navigate = 2
      PRIVATE rule_pnt
      rule_pnt = 0
      SEEK Var
      DO WHILE var_class = var
            rule_pnt = rule_pnt + 1
            rule_list[rule_pnt] = name
            @ 4+rule_pnt,55 PROMPT rule_list[rule_pnt]
            SKIP
      ENDDO
      @ 5+rule_pnt, 55 PROMPT 'NEW            '
      MENU TO Select
      DO CASE
      CASE Select = 0
            Navigate = Navigate - 1
      CASE Select <= rule_pnt
            *edit an old old
            SEEK var+rule_list[Select]
            IF Cmplt_rule(var, rule_list[Select],.T.)&& edit mode
                  Navigate = Navigate - 1
            ELSE
                  *back to this one
            ENDIF
      OTHERWISE
            * add a new rule
            B_rule = space(15)
            @ 5+rule_pnt, 40 SAY 'Enter a name: ' GET B_rule
            READ
            IF (len(trim(B_rule)) < 1 .OR. lastkey() = 27)
                  Navigate = Navigate - 1
            ELSE
                  SEEK (var+B_rule)
                  IF eof()
                        *does not already exists
                        @ 5+rule_pnt, 40 CLEAR  TO 5+rule_pnt,55
                        @ 5+rule_pnt, 55 SAY B_rule
                        Navigate = Navigate + 1
                  ELSE
                        ErrWait ('rule already exists')
                  ENDIF
            ENDIF
      ENDCASE

CASE Navigate = 3
      * add the rest of the rule
      IF Cmplt_rule(var, B_rule,.F.)      &&not edit mode
            Navigate = Navigate + 1
            REPLACE var_class WITH var
            REPLACE name WITH B_rule
      ELSE
            Navigate = Navigate - 1
```

```
            @ 23,0
            WAIT 'Rule finished, Press Esc to modify'
            IF lastkey() = 27
                  IF B_else # 'NUL'
                        Navigate = Navigate - 1
                  ELSE
                        Navigate = Navigate - 2
                  ENDIF
            ELSE
                  EXIT
            ENDIF
      ENDCASE
ENDDO
*
*     Save the new values
*
SELECT 1     &&rule.dbf
SET INDEX TO rule
SEEK Var_cls+rul_name
REPLACE then WITH b_then
REPLACE then_w WITH b_then_w
REPLACE then_p WITH b_then_p
REPLACE else WITH b_else
REPLACE else_w WITH b_else_w
REPLACE else_p WITH b_else_p
RETURN (.T.)


*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function          :   Add_rule                                -
*  -    Input             :   none                                    -
*  -    Output            :   sets Escape                             -
*  -    Date              :   06 Oct 89                               -
*  -    UpDate            :   27 Oct 89                               -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Synopses:
*           Adds new rules using new or old variable classes.
*     Pseudo:
*     While add more rules
*                 Decide the variable class
*                 While using the same variable class
*                       Add a rule
*
*
FUNCTION Add_rule
EXTERNAL Ans_inter

PRIVATE Var, rule_pnt, B_rule,S_buf
SAVE SCREEN TO S_buf
PRIVATE Navigate
Navigate = 1
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            EXIT
      CASE Navigate = 1
            *find a Var_class
            RESTORE SCREEN FROM S_buf
            @ 1,39 TO 23,79 DOUBLE
            @ 2, 40 SAY 'Enter an rule: '
            Var = Get_var()
```

```
              ENDIF

        CASE Navigate = 4
              * add a new object

              IF   AskY("Enter   another   rule   for   the   variable   class
"+rtrim(Var)+'?')
                    @ 24,0 CLEAR
                    Select = 99
                    Navigate = 2      &&start at the name
              ELSE
                    Navigate = Navigate + 1
              ENDIF

        CASE Navigate = 5
              EXIT
        ENDCASE
ENDDO
RESTORE SCREEN FROM S_buf
IF Navigate > 0
        Escape = .F.
        RETURN(.T.)
ELSE
        Escape = .T.
        RETURN(.F.)
ENDIF
* end function Find_rule

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function          :   Cmplt_rule()                            -
*  -    Input             :   Enquiry name, edit flag                 -
*  -    Output            :   error                                   -
*  -    Date              :   06 Oct 89                               -
*  -    UpDate            :   03 Jan 90                               -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Assumes the file is pointing to the record to add
*     Called from Find_rule or Add_rule, once the name of the rule is
known
FUNCTION Cmplt_rule
PARAMETER Var_cls,;            && the varable class of the rule
          rul_name,;  && the name of this rule
          Edit_flg             && edit an old rule, or modif an existing
rule


PRIVATE S_buf
SAVE SCREEN TO S_buf
@ 1,30 CLEAR
@ 4, 0 SAY Var_cls+'::'+rul_name
PRIVATE Max_Antec
Max_Antec = 6
*
*     buffers to store the rule
*
PRIVATE belf_based
PRIVATE B_question, Ans_lst, i
PRIVATE Num_antec
PRIVATE if_lst, oper_lst, ans_lst, num_lst, Extrn_lst
DECLARE if_lst[Max_Antec]
DECLARE oper_lst[Max_Antec]
DECLARE Extrn_lst[Max_Antec]
DECLARE ans_lst[Max_Antec]
```

```
DECLARE num_lst[Max_Antec]
PRIVATE Mac
DO WHILE .T.
      IF Edit_flg
            PRIVATE rec_pnt
            belf_based = (rule_typ='BLF')
            Antec_pnt = num_antec
            * set the antecedent
            FOR i = 0 TO Max_Antec-1
                  * load all possible ones since they are blank if not
defined
                  Mac = "if"+str(i,1)
                  if_lst[i+1]=&Mac
            NEXT
            *
            *     construct the defined section of the rule
            *
            IF belf_based
                  Antec_pnt = Rule_belief(Antec_pnt)
            ELSE
                  Antec_pnt = Rule_fact(Antec_pnt)
            ENDIF
            IF Antec_pnt = 0
                  ? 'Delete this rule'
                  SEEK Var_cls+rul_name
                  ? name
                  DELETE
                  PACK
                  WAIT
                  RESTORE SCREEN FROM S_buf
                  RETURN (.F.)
            ENDIF
      ELSE
            * new rule
            belf_based = AskN (" Is this a belief-based rule?")
            @24, 0 CLEAR
            * set all storage buffer to empty
            FOR i = 0 TO Max_Antec-1
                  if_lst[i+1]=space(15)
            NEXT
            IF belf_based
                  Antec_pnt = Rule_belief(1)
            ELSE
                  * Add the body of the rule starting with no antecedents
                  Antec_pnt = Rule_fact(1)
            ENDIF
            IF Antec_pnt = 0
                  RESTORE SCREEN FROM S_buf
                  RETURN .F.
            ELSE
                  REPLACE var_class WITH Var_cls
                  REPLACE name            WITH rul_name
            ENDIF
      ENDIF
      *
      *     Now add the consequence of the rule
      *
            IF Rule_consq()
                  EXIT
            ELSE
                  Edit_flg = .T.
                  SEEK Var_cls+rul_name
```

```
            ENDIF

***     IF !belf_based
***         IF Rule_consq()
***             EXIT
***         ELSE
***             Edit_flg = .T.
***             SEEK Var_cls+rul_name
***         ENDIF
***     ELSE
***         EXIT
***     ENDIF
ENDDO
RESTORE SCREEN FROM S_buf
*end function

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function         :   Rule_fact()                              -
*  -    Input            :   no input                                 -
*  -    Output           :   number enterered                         -
*  -    Date             :   06 Oct 89                                -
*  -    UpDate           :   27 Oct 89                                -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*       Convention: this is a member of Cmplt_rule, and should not be
*           called by any other function.
*       Assumes:    many variables and files and screen
*           and that the rule_type is FCT
*
FUNCTION Rule_fact
PARAMETER Antc_pnt
*
*       Establish the buffers from the record or start as blank
*
@8, 0SAY "IF:"
IF Edit_flg
        FOR i = 0 TO Max_Antec-1
            Mac = 'if'+str(i,1)
            if_lst[i+1]=&Mac
            Mac = "oper"+str(i,1)
            oper_lst[i+1]=&Mac
            Mac = "extrn"+str(i,1)
            Extrn_lst[i+1]=&Mac
            Mac = "ans"+str(i,1)
            ans_lst[i+1]=&Mac
            Mac = "num"+str(i,1)
            num_lst[i+1]=&Mac
        NEXT
        FOR i = 1 TO Antc_pnt
            IF Extrn_lst[i]
                @8+i, 5 SAY "Extern:"
            ELSE
                @8+i, 5 SAY "Intern:"
            ENDIF
            @8+i,12 SAY if_lst[i]
            @8+i, 30 SAY oper_lst[i]
            IF !Extrn_lst[i] .AND. (len(trim(Ans_lst[i])) > 0)
                @8+I, 35 SAY Ans_lst[i]
            ELSE
                @8+i, 35 SAY num_lst[i]
            ENDIF
        NEXT
        PRIVATE Navigate
```

```
      Navigate = 5
ELSE
      FOR i = 0 TO Max_Antec-1
            oper_lst[i+1]=space(2)
            ans_lst[i+1]=space(30)
            num_lst[i+1]=0.0
            Extrn_lst[i+1]=.F.
      NEXT
      PRIVATE Navigate
      Navigate = 1
ENDIF
DO WHILE .T.
      DO CASE
      CASE Navigate = 0
            *backed out
            IF Antc_pnt > 1
                  Antc_pnt = Antc_pnt - 1
                  Navigate = 4
            ELSE
                  SELECT 1    &&rule.dbf
                  SET INDEX TO rule
                  RETURN (0)
            ENDIF

      CASE Navigate = 1
            * find out what type of fact human or external
            @24, 0 CLEAR
            @8+Antc_pnt, 5 SAY "*              "
            @24, 0 SAY "Is this antecedent based on external facts?";
            GET Extrn_lst[Antc_pnt] &&a logic
            READ
            IF lastkey() # 27
                  Navigate = Navigate + 1
                  IF Extrn_lst[Antc_pnt]
                        @8+Antc_pnt, 5 SAY "Extern:"
                  ELSE
                        @8+Antc_pnt, 5 SAY "Intern:"
                  ENDIF
            ELSE
                  Navigate = Navigate - 1
            ENDIF

      CASE Navigate = 2
            * enter the next fact name
            @8+Antc_pnt, 12 CLEAR TO 8+Antc_pnt, 30
            IF Extrn_lst[Antc_pnt]
                  if_lst[Antc_pnt] = Get_extreq(Var_cls)
            ELSE
                  if_lst[Antc_pnt] = Get_internal(Var_cls)
            ENDIF
            IF if_lst[Antc_pnt]# 'NIL'
                  Navigate = Navigate + 1
                  @8+Antc_pnt,12 SAY if_lst[Antc_pnt]
            ELSE
                  Navigate = Navigate - 1
            ENDIF
      CASE Navigate = 3
            * get the operand of the fact
            @24,0 CLEAR
            @24,0 SAY "Enter an operand"
            @8+Antc_pnt, 30 GET oper_lst[Antc_pnt]
            READ
```

```
            IF lastkey() # 27
                  Navigate = Navigate + 1
                  @8+Antc_pnt, 30 SAY oper_lst[Antc_pnt]
            ELSE
                  @8+Antc_pnt, 30 CLEAR TO 8+Antc_pnt, 36
                  Navigate = Navigate - 1
            ENDIF
      CASE Navigate = 4
            * get the number/symbol of the fact
            @24,0 CLEAR
            @8+Antc_pnt, 35 CLEAR TO 8+Antc_pnt, 60
            IF Extrn_lst[Antc_pnt]
                  @8+Antc_pnt, 35 GET num_lst[Antc_pnt] PICTURE "99999.99"
                  READ
                  IF lastkey() # 27
                        @8+Antc_pnt, 35 SAY num_lst[Antc_pnt] PICTURE
"99999.99"
                        Navigate = Navigate + 1
                  ELSE
                        @8+Antc_pnt, 35 CLEAR TO 8+Antc_pnt, 60
                        Navigate = Navigate - 1
                  ENDIF
            ELSE
                  *human interface
                  PRIVATE text_ans, answ_text, answ_num, error_flag
                  text_ans = .T.
                  answ_text = ''
                  answ_num = 0
                  error_flag = .T.
                  DO Ans_inter WITH
Text_ans,Answ_text,Answ_num,Error_flag,;
                   Var_cls,if_lst[Antc_pnt]
                  IF Error_flag
                        Navigate = Navigate - 1
                  ELSE
                        IF Text_ans
                              ans_lst[Antc_pnt] = Answ_text
                              @8+Antc_pnt, 35 SAY Answ_text
                        ELSE
                              num_lst[Antc_pnt] = Answ_num
                              @8+Antc_pnt, 35 SAY Answ_num
                        ENDIF
                        Navigate = Navigate + 1
                  ENDIF
            ENDIF

      CASE Navigate = 5
            * if there is space get another antecedent
            @24,0 CLEAR
            IF Antc_pnt < Max_Antec
                  IF AskY ("Enter another antecedent?")
                        Antc_pnt = Antc_pnt + 1
                        Navigate = 1
                  ELSE
                        Navigate = Navigate + 1
                  ENDIF
            ELSE
                  @23,0
                  WAIT "Antecedent full, Pres Esc to change"
                  IF lastkey() # 27
                        Navigate = Navigate + 1
                  ELSE
```

```
                        Navigate = Navigate - 1
                  ENDIF
            ENDIF
      CASE Navigate = 6
                  EXIT
      ENDCASE
ENDDO
*
*     Do much saving
*
SELECT 1    &&rule.dbf
SET INDEX TO rule
IF !Edit_flg
      APPEND BLANK
      REPLACE rule_typ WITH ’FCT’
ELSE
      SEEK Var_cls+rul_name
ENDIF

FOR i = 0 TO Max_Antec-1
      Mac = ’if’+str(i,1)
      REPLACE &Mac WITH if_lst[i+1]
      Mac = "oper"+str(i,1)
      REPLACE &Mac WITH oper_lst[i+1]
      Mac = "extrn"+str(i,1)
      REPLACE &Mac WITH Extrn_lst[i+1]
      Mac = "ans"+str(i,1)
      REPLACE &Mac WITH ans_lst[i+1]
      Mac = "num"+str(i,1)
      REPLACE &Mac WITH num_lst[i+1]
      REPLACE num_antec WITH Antc_pnt
NEXT
RETURN(Antc_pnt)

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :  Rule_belief()                          -
*  -    Input           :  no input                               -
*  -    Output          :  number enterered                       -
*  -    Date            :  06 Oct 89                              -
*  -    UpDate          :  03 Jan 90                              -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Convention: this is a member of Cmplt_rule, and should not be
*           called by any other function.
*     Assumes:    many variables and files and screen
*           and that the rule_type is FCT
*
FUNCTION Rule_belief
PARAMETER Antc_pnt
*
*     Establish the buffers from the record or start as blank
*
@8, 0SAY "IF:"
IF Edit_flg
      FOR i = 0 TO Max_Antec-1
            Mac = ’if’+str(i,1)
            if_lst[i+1]=&Mac
      NEXT
      FOR i = 1 TO Antc_pnt
            @8+i,8 SAY if_lst[i]
      NEXT

      PRIVATE Navigate
```

```
            Navigate = 2

    ELSE
            PRIVATE Navigate
            Navigate = 1
    ENDIF
    DO WHILE .T.
            DO CASE
            CASE Navigate = 0
                    *backed out
                    IF Antc_pnt > 1
                            Antc_pnt = Antc_pnt - 1
                            Navigate = 2
                    ELSE
                            SELECT 1    &&rule.dbf
                            SET INDEX TO rule
                            RETURN (0)
                    ENDIF

            CASE Navigate = 1
                    * enter the next fact name
                    @8+Antc_pnt, 8 SAY "*"+space(30)
                    if_lst[Antc_pnt] = Get_belief(Var_cls)
                    IF if_lst[Antc_pnt]# 'NUL'
                            Navigate = Navigate + 1
                            @8+Antc_pnt,8 SAY if_lst[Antc_pnt]
                    ELSE
                            Navigate = Navigate - 1
                    ENDIF

            CASE Navigate = 2
                    * if there is space get another antecedent
                    @24,0 CLEAR
                    IF Antc_pnt < Max_Antec
                            IF AskY ("Enter another antecedent?")
                                    IF lastkey() # 27
                                            Antc_pnt = Antc_pnt + 1
                                            Navigate = 1
                                    ELSE
                                            Navigate = Navigate - 1
                                    ENDIF
                            ELSE
                                    Navigate = Navigate + 1
                            ENDIF
                    ELSE
                            @23,0
                            WAIT "Antecedent full, Pres Esc to change"
                            IF lastkey() # 27
                                    Navigate = 5

                            ELSE
                                    Navigate = Navigate - 1
                            ENDIF
                    ENDIF

            CASE Navigate = 3
                            EXIT
            ENDCASE
    ENDDO
    *
    *       Do much saving
    *
```

```
SELECT 1    &&rule.dbf
SET INDEX TO rule
IF !Edit_flg
      APPEND BLANK
      REPLACE rule_typ WITH 'BLF'
ELSE
      SEEK Var_cls+rul_name
ENDIF
REPLACE num_antec WITH Antc_pnt
FOR i = 0 TO Max_Antec-1
      Mac = 'if'+str(i,1)
      REPLACE &Mac WITH if_lst[i+1]
NEXT

RETURN(Antc_pnt)
*end function Rule_belief

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   rule_var                                 -
*  -    Input           :   none                                    -
*  -    Output          :   variable class                          -
*  -    Date            :   06 Oct 89                               -
*  -    UpDate          :   06 Oct 89                               -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION rule_var
SELECT 4    && rule.dbf
SET INDEX TO rule
PRIVATE B_var
B_var = Get_var()
SET INDEX   TO
RETURN (B_var)

*
*end function rule_var
```

```
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* =   Project         :  KOOLA programing language Shell                 =
* =   Sub-project     :  Compiler                                        =
* =   File            :  main.prg                                        =
* =   Author          :  Robert D. Rourke                                =
* =   Date            :  19 Dec 1989                                     =
* =   Update          :  04 Jan 1990                                     =
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
*     Databases:
*     Select      Name  Index Code
*     --------+---------+---------+--------
*     1           rule        rule        r
*     2           goal        goal        g
*     3           belief      belief      b
*     4           internal    internal    i
*     5           extreq      extreq      x
*     6           action      action      a
*     7           primary     primary     p
*     8           second      second      s
*     all index set for var_class+name except goal
*
*     This file contains the rourtines to do the first two passes of the
*     KOOLA compiler.  This includes generating the primary and secondary
*     belief files.
*
Date ='19 Dec 1989'
Version = 'x.10'

Deflt_Path = Get_direct('sim\')

PUBLIC Deflt_path, Escape
* set the colour for a colour monitor
In_colour = .f.
IF iscolor ()
     PUBLIC Colour_Edit
     PUBLIC Colour_Menu
     In_colour = .T.
     Colour_Edit = "GR+/B,GR+/B,B,B,W+/RB"
     Colour_Menu = "GR+,W+/B,B,B,W+/BG"
     setcolor(Colour_Menu)
ENDIF
SET CENTURY ON
SET DATE BRITISH
SET WRAP ON
CLEAR

*Deflt_Path = Get_direct()
Deflt_Path = ''
File_error = .F.

SELECT 1
File_name = 'rule.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 2
File_name = 'goal.dbf'
```

```
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 7
File_name = 'primary.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 8
File_name = 'second.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 3
File_name = 'belief.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 4
File_name = 'internal.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 5
File_name = 'extreq.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 6
File_name = 'action.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF
```

```
IF File_error
      ? '*** File Error termination of KOOLA  ***'
      ?
      RETURN
ENDIF

*
*     Start of routine
*




* -----------------------------------------------
* -                                             -
* -      Pass One of the Compiler               -
* -                                             -
* -----------------------------------------------
*
*     copy all the belief into the primary belief file
*
SELECT 7          &&primary.dbf
ZAP
APPEND FROM belief


SELECT 1    &&rule
GO 1
PRIVATE Rec
Rec = 1
?'Passe one...'
?
DO WHILE !eof()
      ??'.'
      *check the rule type
      DO CASE
      CASE rule_typ =  'FCT'
            Prim_rule(Rec)
      CASE rule_typ =  'BLF'
            *skip over
      OTHERWISE
            ?'Error unkown rule type'
            WAIT
      ENDCASE
      Rec = Rec + 1
      SELECT 1    &&rule
      GO Rec
ENDDO
*
*     Find out what beliefs in the primary base have no facts atached.
*     These type of beliefs are secondary beliefs and should be moved
*     to the secondary rule base.
*
SET INDEX TO
SELECT 7    &&primary.dbf
GO TOP
DO WHILE !eof()
      IF num_facts > 0
            *
            *     This check indicaces that there is no facts supporting
```

```
the
            *     belief.  Therefor, the belief is not primary.  As a
result,
            *     the belief is keep if concidered primary, and copied if
not
            *
            REPLACE defined WITH .F.
      ELSE
            REPLACE defined WITH .T.
      ENDIF
      SKIP 1
ENDDO
USE   &&close the file to permit appending

SELECT 8     &&second.dbf
ZAP
APPEND FROM primary FOR defined
INDEX ON var_class+name to second

SELECT 7     &&primary.dbf
USE primary
DELETE FOR defined
PACK
INDEX ON var_class+name to primary

* -----------------------------------------------
* -                                             -
* -      Pass Two of the Compiler               -
* -                                             -
* -----------------------------------------------
SELECT 1     &&rule
GO 1
PRIVATE Rec
Rec = 1
?'Pass two...'
?
DO WHILE !eof()
      ?? '.'
      *check the rule type
      DO CASE
      CASE rule_typ =  'FCT'
            *skip over
      CASE rule_typ =  'BLF'
            Secd_rule(Rec)
      OTHERWISE
            ?'Error unkown rule type'
            WAIT
      ENDCASE
      Rec = Rec + 1
      SELECT 1    &&rule
      GO Rec
ENDDO

* -----------------------------------------------
* -                                             -
* -      Pass three copy secblf pointers to goal-
* -                                             -
* -----------------------------------------------

SELECT 2          &&goal.dbf
PRIVATE Num_rec, Cur_rec, B_beliefpnt
Num_rec = reccount()
```

```
?'Pass three'
FOR Cur_rec=1 TO Num_rec
      GO Cur_rec
      REPLACE operand_cd WITH Map_opcode(operand)
      B_beliefpnt = F_Sbelf(var_class,belif)
      SELECT 2          &&goal.dbf
      REPLACE belif_pnt WITH B_beliefpnt
NEXT


*
*     Generate the goal pointers for forward chaining
*
SELECT 2          &&goal.dbf
SET INDEX TO goal
GO TOP
PRIVATE b_then,b_thenpnt,b_else,b_elsepnt

FOR Cur_rec=1 TO Num_rec
      GO Cur_rec
      b_then = then_chain
      b_else = else_chain

      IF b_then # 'TERMINATE'
            SEEK B_then
            IF eof()
                  ? 'Error in looking up a goal: '
                  ? B_then
                  wait
                  b_thenpnt = -999
            ELSE
                  b_thenpnt = recno()
            ENDIF
      ELSE
            b_thenpnt = 0
      ENDIF

      IF b_else # 'TERMINATE'
            SEEK B_else
            IF eof()
                  ? 'Error in looking up a goal'
                  wait
                  b_elsepnt = -999
            ELSE
                  b_elsepnt = recno()
            ENDIF
      ELSE
            b_elsepnt = 0
      ENDIF

      SELECT 2          &&goal.dbf
      GO Cur_rec
      REPLACE tchn_pnt WITH b_thenpnt
      REPLACE echn_pnt WITH b_elsepnt
NEXT


* -----------------------------------------------
* -                                             -
* -      Make an include file of all HU quest   -
* -                                             -
```

```
* -----------------------------------------------
Generat_inc()
* -----------------------------------------------
* -                                             -
* -       Make data files for the rule base     -
* -                                             -
* -----------------------------------------------
* this should be after the compiling
Generat_ie()

RETURN

*
*end file main.prg
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project          :   KOOLA programing language Shell               =
*  =    Sub-project      :   Compiler                                      =
*  =    File             :   geninclu.prg                                  =
*  =    Author           :   Robert D. Rourke                              =
*  =    Date             :   25 Jan 1990                                   =
*  =    Update           :   24 Feb 1990                                   =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
FUNCTION Generat_inc
? 'Please stand by...'

SELECT 4           &&internal.dbf'
*
*     Find the number of each type
*
GO TOP
Num_text = 0
Num_numr = 0
DO WHILE !eof()
      IF inquir_typ = 'TEX'
            Num_text = Num_text + 1
      ElSE
            Num_numr = Num_numr + 1
      ENDIF
      SKIP
ENDDO

*     set up output file
SET INDEX TO internal
GO TOP

SET PRINTER TO huminter.h
SET DEVICE TO PRINT
SET PRINT ON
SET CONSOLE OFF

@ 0, 0
printf(' /*= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =')
printf(' =    Project           :   KOOLA programing language
   =')
printf(' =    Sub-project       :   Include file for human interface
   =')
printf(' =    Language          :   C-286 for intel RMX operating System
   =')
printf(' =    File              :   huminter.h
   =')
```

```
printf(' =    Date              :    '+dtoc(date())+;
 '                                    =')
printf(' = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =')

printf(' ')
printf('    Warning:    do not make any changes to this file because it
can be')
printf('                automaticaly update by the KOOLA compiler')
printf('*/')

printf('#ifndef    _HUMANINTER')
printf('#define    _HUMANINTER       1')
printf('/*')
printf('    Some maximum values used to compile the human interface:')
printf('*/')
printf('#define    MAXANSWERS              7      /* max number of questions
per question */')
printf('#define    NUMQUEST                '+ltrim(str(reccount()))+'    */
total number of questions defined */')
printf('#define    NUMTEXT                 '+ltrim(str(Num_text))+'      */
number of text based quest*/')
printf('#define    NUMNUMR                 '+ltrim(str(Num_numr))+'      */
number of numer based quest*/')
printf(' ')
printf('/*')
printf('    The constants for accessing the human interface:')
printf('*/')
GO TOP
printf(' ')
printf('/* variable class: '+trim(var_class)+' */')
DO WHILE var_class = 'OS'
      printf(' ')
      printf('/* type: '+trim(inquir_typ)+' */')
      printf('#define    '+'HIQST_'+upper(trim(name))+;
       '            '+str(recno(),5,0))
      printf('#define    '+'HISIZE_'+upper(trim(name))+;
       '            '+str(num_ams,5,0))
      SKIP
ENDDO
printf(' ')
printf('#endif')
printf(' ')
printf('/* end file huminter.h */')
*
*     Close the file
*
SET CONSOLE ON
SET DEVICE TO SCREEN
SET PRINT OFF
SET PRINTER TO


*
*     Start creating the data file
*
SET PRINTER TO huminter.dat
SET DEVICE TO PRINT
SET PRINT ON
SET CONSOLE OFF
prints('HUMAN INTERFACE DATA FILE  Rourke 90  Da[illegible] date: '+dtoc(date()) )
printf('')
```

```
printf('')
SET INDEX TO
GO TOP
DO WHILE !eof()
      prints (inquir_typ)
      prints (Shelf_life)
      prints (question)
      IF inquir_typ = 'TEX'
            prints (num_ams)
            FOR i= 0 TO num_ams-1
                  Macro = 'ANS'+str(i,1)
                  prints(&Macro)
            NEXT
      ELSE
            prints(upper)
            prints(lower)
      ENDIF
      printf('')
      SKIP
ENDDO
SET CONSOLE ON
SET DEVICE TO SCREEN
SET PRINT OFF
SET PRINTER TO



*
*     The external requests
*

SELECT 5    &&extreq.dbf

@ 0, 0

SET PRINTER TO extreq.h
SET DEVICE TO PRINT
SET PRINT ON
SET CONSOLE OFF

printf(' /*= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =')
printf(' =    Project          :   KOOLA programing language
   =')
printf(' =    Sub-project      :   Include file for external requests
   =')
printf(' =    Language         :   C-286 for intel RMX operating System
   =')
printf(' =    File             :   extreq.h
   =')
printf(' =    Date             :   '+dtoc(date())+;
 '                              =')
printf(' = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =')

printf(' ')
printf('    Warning:    do not make any changes to this file because it
can be')
printf('                automaticaly update by the KOOLA compiler')
printf('*/')
```

```
printf('#ifndef    _EXTRQINTER')
printf('#define    _EXTRQINTER        1')
printf('/*')
printf('    Some maximum values used to compile the interface to COSI:')
printf('*/')
printf('#define    NUMEXRQ                 '+ltrim(str(reccount()))+'    */
total number of questions defined */')
printf(' ')
printf('#endif')
printf(' ')
printf('/* end file huminter.h */')

SET CONSOLE ON
SET DEVICE TO SCREEN
SET PRINT OFF
SET PRINTER TO


@ 0, 0
SET PRINTER TO extreq.dat
SET DEVICE TO PRINT
SET PRINT ON
SET CONSOLE OFF
prints('EXTERNAL REQUESTS DATA FILE  Rourke 90  Data date: '+dtoc(date())
)
printf('')
printf('')
SET INDEX TO
GO TOP
DO WHILE !eof()
      printf (token)
      prints (shelf_life)
      SKIP
ENDDO
SET CONSOLE ON
SET DEVICE TO SCREEN
SET PRINT OFF
SET PRINTER TO




FUNCTION printf
PARAMETER string
? string

FUNCTION prints
PARAMETER string
?? string
?? '  '
*
*end file geninclude

* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* =   Project          :   KOLA programing language Shell              =
* =   Sub-project      :   Compiler                                    =
* =   File             :   secondary.prg                               =
* =   Author           :   Robert D. Rourke                            =
* =   Date             :   29 Dec 2089                                 =
* =   Update           :   29 Dec 1989                                 =
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

```
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function         :   Secd_rule                                   -
* -    Input            :   record number of the rule                   -
* -    Output           :   nul                                         -
* -    Action           :   Compiles one secondary rule                 -
* -    Date             :   29 Dec 89                                   -
* -    UpDate           :   29 Dec 89                                   -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION    Secd_rule
PARAMETER rule_pnt
SELECT 1    &&rule
GO rule_pnt
PRIVATE B_var
B_var = var_class
*
*     Loads all values from the rule
*

PRIVATE B_then, B_then_w, B_then_p
B_then   = then
B_then_w = then_w
B_then_p = then_p

PRIVATE b_else, B_else_w, B_else_p
B_else   = else
B_else_w = else_w
B_else_p = else_p


PRIVATE Num_Sbelf, Sbelf
Num_Sbelf = num_antec
DECLARE Sbelf[Num_Sbelf]
PRIVATE Sbelf_id
DECLARE Sbelf_id[Num_Sbelf]

*
*     Load the Sbelfs of the rule
*
PRIVATE i, Macro
FOR i = 0 TO Num_Sbelf-1
      Macro = 'IF'+str(i,1)
      Sbelf[i+1] = &Macro
***   ? 'The belief and the id:'
***   ??Sbelf[i+1]
NEXT i
FOR i = 0 TO Num_Sbelf-1
      Sbelf_id[i+1] = F_Sbelf(B_var,Sbelf[i+1])
***   ??Sbelf_id[i+1]
NEXT i

*
*     display the new belief for then
*
*SET PRINT ON
***? 'The then belief followed by prob and weight  then rule ID:'
***?  B_then
***?? B_then_p
***?? B_then_w
***?? rule_pnt


***FOR i = 1 TO Num_Sbelf
```

```
***    ?' Sbelf id: '
***    ??Sbelf_id[i]
***    ?
***NEXT

Ssecd_belf (B_var, B_then, rule_pnt, Sbelf_id[1],B_then_p,B_then_w)

FOR i = 2 TO Num_Sbelf
      Ssecd_belf (B_var, B_then, rule_pnt, Sbelf_id[i], -1,-1)
NEXT
*
*     store the else beliefs
*
IF  B_else # 'NUL'
      Ssecd_belf  (B_var,  B_else,  rule_pnt+500,
Sbelf_id[1],B_else_p,B_else_w)
      FOR i = 2 TO Num_Sbelf
            Ssecd_belf (B_var, B_else, rule_pnt+500, Sbelf_id[i],-2,-2)
      NEXT
ENDIF
*end function Prim_rule

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function        :   F_Sbelf                                  -
* -    Input           :   var and name of suport belief            -
* -    Output          :   Sbelf ID in excess 500   0 = not found   -
* -    Action          :   locates a belf in the prim or secd db    -
* -    Date            :   29 Dec 89                                -
* -    UpDate          :   04 Jan 90                                -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* An excess of 500 is added if the beief is found in the primary
* database.
FUNCTION F_Sbelf
PARAMETER Var,;          &&str, the variable class
            Sbelf_name   &&str, the name of the Sbelf
*
*     Check if its in the primary db first
*
SELECT 7    && primary
SET INDEX TO primary
SEEK Var+Sbelf_name
IF !eof()
      RETURN (recno()+500)
ENDIF
SELECT 8    && second.dbf'
SET INDEX TO second
SEEK Var+Sbelf_name
IF eof()
      ? 'belief not found var and name:'
      ??var+', '+Sbelf_name
      WAIT
      RETURN (0)
ELSE
      RETURN (recno())
ENDIF
*end function F_Sbelf

*
*end file secondar.prg

* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* =    Project         :   KOOLA programing language Shell           =
```

```
*  =    Sub-project     :   Compiler                                    =
*  =    File            :   ie.prg                                      =
*  =    Author          :   Robert D. Rourke                            =
*  =    Date            :   25 Jan 1990                                 =
*  =    Update          :   14 Apr 1990                                 =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
FUNCTION Generat_ie
? 'Please stand by...'

SET CONSOLE OFF
@ 0, 0
SET PRINTER TO prknbase.dat
SET DEVICE TO PRINT
SET PRINT ON
prints('PRIMARY  BELIEFS  KNOWLEDGE  BASE    Rourke  90    Data  date:
'+dtoc(date()) )
printf('')
printf('')
SET INDEX TO
SELECT 7    &&primary.dbf
GO TOP
PRIVATE Macrstr, rec
PRIVATE Numb_blf
Numb_blf = reccount()
FOR rec = 1 TO Numb_blf
       GO rec
       printf (num_facts)
       printf('')
       FOR i = 1 to num_facts
              Macrstr = 'R'+ltrim(str(i,2))
              prints (&Macrstr)
       NEXT
       printf('')
       FOR i = 1 to num_facts
              Macrstr = 'F'+ltrim(str(i,2))
              prints (&Macrstr)
       NEXT
       printf('')
       FOR i = 1 to num_facts
              Macrstr = 'P'+ltrim(str(i,2))
              prints (&Macrstr)
       NEXT
       printf('')
       FOR i = 1 to num_facts
              Macrstr = 'W'+ltrim(str(i,2))
              prints (&Macrstr)
       NEXT
       printf('')
       FOR i = 1 to num_facts
              Macrstr = 'O'+ltrim(str(i,2))
              prints (&Macrstr)
       NEXT
       printf('')
       FOR i = 1 to num_facts
              Macrstr = 'N'+ltrim(str(i,2))
              prints (&Macrstr)
       NEXT
       printf('')
       printf('')
NEXT rec
SET CONSOLE ON
```

```
SET DEVICE TO SCREEN
SET PRINT OFF
SET PRINTER TO


SET CONSOLE OFF
@ 0, 0
SET PRINTER TO sdknbase.dat
SET DEVICE TO PRINT
SET PRINT ON
prints('SECONDARY  BELIEFS  KNOWLEDGE  BASE    Rourke  90    Data  date:
'+dtoc(date()) )
printf('')
printf('')
SET INDEX TO
SELECT 8          &&second.dbf
GO TOP
PRIVATE Macrstr, Num_secrule
Num_secrule = 0

PRIVATE Numb_blf, Rec
Numb_blf = reccount()
FOR Rec = 1 TO Numb_blf
       GO Rec
       IF num_belief > 0
              Num_secrule = Num_secrule + 1
              printf (num_belief)
              printf('')
              FOR i = 1 to num_belief
                     Macrstr = 'R'+ltrim(str(i,2))
                     prints (&Macrstr)
              NEXT
              printf('')
              FOR i = 1 to num_belief
                     Macrstr = 'B'+ltrim(str(i,2))
                     prints (&Macrstr)
              NEXT
              printf('')
              FOR i = 1 to num_belief
                     Macrstr = 'P'+ltrim(str(i,2))
                     prints (&Macrstr)
              NEXT
              printf('')
              FOR i = 1 to num_belief
                     Macrstr = 'W'+ltrim(str(i,2))
                     prints (&Macrstr)
              NEXT
              printf('')
              printf('')
       ELSE
              printf(0)
              printf('')
       ENDIF
NEXT
SET CONSOLE ON
SET DEVICE TO SCREEN
SET PRINT OFF
SET PRINTER TO


SET CONSOLE OFF
@ 0, 0
```

```
SET PRINTER TO mtknbase.dat
SET DEVICE TO PRINT
SET PRINT ON
prints('META-KNOWLEDGE GOAL KNOWLEDGE BASE   Rourke 90   Data date:
'+dtoc(date()) )
printf('')
printf('')
SET INDEX TO
SELECT 2          &&goal.dbf
GO TOP
PRIVATE num_goal
num_goal = reccount()
FOR i = 1 TO num_goal
      GO i
      IF start_goal
            printf('    1      ')
      ELSE
            printf('    0      ')
      ENDIF
      prints(name)
      printf(belif_pnt)
      printf(operand_cd)
      printf(prob)
      printf(tchn_pnt)
      prints("    ")
      prints(echn_pnt)
      printf(F_Action(var_class,then_do))
      SELECT 2          &&goal.dbf
      GO i
      prints("    ")
      prints(F_Action(var_class,else_do))
      SELECT 2          &&goal.dbf
      printf(' ')
      printf(' ')
      SKIP
NEXT
SET CONSOLE ON
SET DEVICE TO SCREEN
SET PRINT OFF
SET PRINTER TO


SET CONSOLE OFF
@ 0, 0
SET PRINTER TO knowbase.h
SET DEVICE TO PRINT
SET PRINT ON

printf(' /*= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =')
printf(' =    Project        :   KOOLA programing language
   =')
printf(' =    Sub-project    :   Include file for primary inference eng
   =')
printf(' =    Language       :   C-286 for intel RMX operating System
   =')
printf(' =    File           :   knowbase.h
   =')
printf(' =    Date           :   '+dtoc(date())+;
 '                            =')
printf(' = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =')
```

```
printf(' ')
printf('    Warning:    do not make any changes to this file because it
can be')
printf('                automaticaly update by the KOOLA compiler')
printf('*/')

printf('#ifndef    _PRIMARYINF')
printf('#define    _PRIMARYINF')
printf('/*')
printf('    Some maximum values used by the PMX C compiler for the
inference engines:')
printf('*/')
SELECT 7     &&primary.dbf
**USE primary
printf('#define    NUMPRMBLF              '+ltrim(str(reccount()))+'    */
total number of primary beliefs */')
SELECT 8           &&second.dbf
printf('#define    NUMSECBLF              '+ltrim(str(reccount()))+'    */
total number of secondary beliefs */')
SELECT 2           &&goal.dbf
printf('#define    NUMGOAL                '+ltrim(str(reccount()))+'    */
total number of goals */')
printf(' ')
printf('#endif')
printf(' ')
printf('/* end file knowbase.h */')
?

SET CONSOLE ON
SET DEVICE TO SCREEN
SET PRINT OFF
SET PRINTER TO

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function       :    F_Action                                     -
* -    Input          :    var and name of action                       -
* -    Output         :    code number of acktions                      -
* -    Action         :    locates                                      -
* -    Date           :    17 Apr 90                                    -
* -    UpDate         :    17 Apr 90                                    -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION F_Action
PARAMETER Var,;         &&str, the variable class
             G_name     &&str, the name of the Sbelf

IF G_name = 'NO ACTION'
      RETURN (0)
ENDIF
PRIVATE action_id
SELECT 6     &&action.dbf
SET INDEX TO action
SEEK Var+G_name
IF eof()
      ? 'Action not found'
      ?var+g_name
      Action_id = 0
ELSE
      Action_id = token
ENDIF
SET INDEX TO
RETURN (Action_id)
*end function F_Action
```

```
*
*end file ie.prg

* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* =   Project          :   KOOLA programing language Shell               =
* =   Sub-project      :   Compiler                                      =
* =   File             :   prim.prg                                      =
* =   Author           :   Robert D. Rourke                              =
* =   Date             :   19 Dec 2089                                   =
* =   Update           :   04 Jan 1990                                   =
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -   Function         :   Prim_rule                                   -
* -   Input            :   record number of the rule                   -
* -   Output           :   nul                                         -
* -   Action           :   Compiles one primary rule                   -
* -   Date             :   19 Dec 89                                   -
* -   UpDate           :   04 Jan 90                                   -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Prim_rule
PARAMETER rule_pnt
SELECT 1    &&rule
GO rule_pnt

PRIVATE B_var
B_var = var_class
*
*     Loads all values from the rule
*

PRIVATE B_then, B_then_w, B_then_p
B_then    = then
B_then_w = then_w
B_then_p = then_p

PRIVATE b_else, B_else_w, B_else_p
B_else    = else
B_else_w = else_w
B_else_p = else_p

PRIVATE Num_Sfact, Fact, Operand, Operator, F_exter, Numer_oper
Num_Sfact = num_antec
DECLARE Fact[Num_Sfact]
DECLARE Operand[Num_Sfact]
DECLARE Operator[Num_Sfact]
DECLARE F_exter[Num_Sfact]
DECLARE Numer_oper[Num_Sfact]

*
*     Load the facts of the rule
*
PRIVATE i, Macro
FOR i = 0 TO Num_Sfact-1
      Macro = 'IF'+str(i,1)
      Fact[i+1] = &Macro
      Macro = 'OPER'+str(i,1)
      Operator[i+1] = Map_opcode(&Macro)
      Macro = 'EXTRN'+str(i,1)
      F_exter[i+1] = &Macro
      Macro = 'ANS'+str(i,1)
      Operand[i+1] = trim(&Macro)
```

```
      Macro = 'NUM'+str(i,1)
      Numer_oper[i+1] = &Macro
NEXT i

*
*     Find the IDs of those facts
*
PRIVATE Fact_id, Oper_code, Ans_code
DECLARE Fact_id[Num_Sfact]
DECLARE Oper_code[Num_Sfact]
DECLARE Ans_code[Num_Sfact]

FOR i = 0 TO Num_Sfact-1
      Fact_id[i+1] = F_fact(B_var,Fact[i+1],F_exter[i+1])
      *
      *      if the fact is human-text, then find an index for its correct
answer
      *
      IF (len(trim(Operand[i+1])) > 0)    && only text answers have one
             *a texted based answer
             Ans_code[i+1] = F_anscode(Fact_id[i+1],Operand[i+1])
      ELSE
             *
             *      The offset of 500 is automaticaly added for extern
             *
             Ans_code[i+1] = Numer_oper[i+1]
      ENDIF
NEXT
*
*     display the new belief for then
*
*SET PRINT ON
**? 'The then belief followed by prob and weight  then rule ID:'
**?  B_then
**?? B_then_p
**?? B_then_w
**?? rule_pnt


SPrim_belf (B_var, B_then, rule_pnt, Fact_id[1], Operator[1],;
 Ans_code[1], B_then_p,B_then_w)
**? 'The facts fol: id, operand code, answer code:'
FOR i = 2 TO Num_Sfact
      SPrim_belf (B_var, B_then, rule_pnt, Fact_id[i], Operator[i],;
       Ans_code[i], -1,-1)
**    ?' Fact id: '
**    ??Fact_id[i]
**    ?' Operantor: '
**    ??Operator[i]
**    ?' Answer code: '
**    ??Ans_code[i]
**    ?
NEXT
IF  B_else # 'NUL'
      SPrim_belf (B_var, B_else, rule_pnt+500, Fact_id[1], Operator[1],;
       Ans_code[1], B_else_p,B_else_w)
      FOR i = 2 TO Num_Sfact
             SPrim_belf   (B_var,   B_else,    rule_pnt+500,    Fact_id[i],
Operator[i],;
              Ans_code[i],-2,-2)
      NEXT
**    ? 'The else belief followed by prob and weight  else rule ID:'
```

```
**    ? B_else
**    ?? B_else_p
**    ?? B_else_w
**    ?? rule_pnt
ENDIF
**browse()
*SET PRINT OFF
*end function Prim_rule

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function        :    F_fact                            -
* -    Input           :    var and name and type             -
* -    Output          :    fact ID in excess 500   0 = not found  -
* -    Action          :    locates a fact and returns its ID -
* -    Date            :    06 Nov 89                         -
* -    UpDate          :    08 Nov 89                         -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION F_fact
PARAMETER Var,;           &&str, the variable class
          Fact_name,; &&str, the name of the fact
          Ext_fact         &&logic T= external F=human
IF Ext_fact
       SELECT 5    &&extreq
       SET INDEX TO extreq
ELSE
       SELECT 4    &&internal
       SET INDEX TO internal
ENDIF

SEEK Var+Fact_name
IF eof()
       ? 'Fact not found var and name:'
       ??var+', '+Fact_name
       WAIT
       RETURN (0)
ENDIF
IF Ext_fact
       RETURN (recno()+500)
ELSE
       RETURN (recno())
ENDIF
*end function F_fact


* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function        :    Map_opcode                        -
* -    Input           :    Opcode string                     -
* -    Output          :    number of the opcode              -
* -    Action          :    translate the string into opcode index  -
* -    Date            :    19 Dec 89                         -
* -    UpDate          :    01 Jan 90                         -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Map_opcode
PARAMETER String
DO CASE
CASE String = '=='.OR. String = '= '
       RETURN (1)
CASE String = '< '
       RETURN (2)
CASE String = '> '
       RETURN (3)
CASE String = '<='
```

```
       RETURN (4)
CASE String = '>='
       RETURN (5)
CASE String = '# '  .OR. String = '!=' .OR. String = '<>'
       RETURN (6)
OTHERWISE
       ?'Unknown operand : '
       ??String+','
       WAIT
       RETURN (0)
ENDCASE
*end function Map_opcode

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -     Function         :    F_anscode                              -
* -     Input            :    humand fact id , operand               -
* -     Output           :    index of fact                          -
* -     Action           :    finds the answer and returns the offset -
* -     Date             :    19 Dec 89                              -
* -     UpDate           :    01 Jan 90                              -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION F_anscode
PARAMETER Fact_id, Operadn
*? 'Looking for the answer :'+Operadn
SELECT 4    &&internal
GO Fact_id
*
*      Find the answer
*
PRIVATE This_ans, i
i = 0
DO WHILE .T.
       This_ans = 'ANS'+str(i,1)
       IF (&This_ans = Operadn)
             RETURN (i+1)
       ENDIF
       i = i + 1
       IF i = 7
             EXIT
       ENDIF
ENDDO
?'Error answer not found!'
WAIT
RETURN (0)
*end function F_anscode

*
*end file prim.prg

* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* =     Project          :    KOOLA programing language Shell        =
* =     Sub-project      :    Compiler                               =
* =     File             :    store.prg                              =
* =     Author           :    Robert D. Rourke                       =
* =     Date             :    28 Dec 1989                            =
* =     Update           :    28 Dec 1989                            =
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -     Function         :    SPrim_belf                             -
* -     Input            :    belief, ruleID, factID, Prob, Weigh    -
* -     Output           :    nul                                    -
* -     Action           :    locates the belief and stores fact     -
```

```
*  -    Date              :    28 Dec 89                            -
*  -    UpDate            :    28 Dec 89                            -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION SPrim_belf
PARAMETER Var, Belf, RuleID, FactID, Operat, Number, Probt, Weigh

*
*     locate the belief
*
SELECT 7
LOCATE FOR var_class = Var .AND. name = Belf
IF eof()
      ?'Error primary belief not found'
      WAIT
      RETURN(.F.)
ENDIF
*
*     Locate everything using macros
*
IF num_facts = 20
      ?'Error overflow in the number of fact'
      WAIT
      RETURN(.F.)
ENDIF
PRIVATE Macro, Offset
REPLACE num_facts WITH num_facts + 1
Offset = ltrim(str(num_facts,2,0))
Macro = 'R'+Offset
REPLACE &Macro WITH RuleID
Macro = 'F'+Offset
REPLACE &Macro WITH FactID
Macro = 'O'+Offset
REPLACE &Macro WITH Operat
Macro = 'N'+Offset
REPLACE &Macro WITH Number
Macro = 'P'+Offset
REPLACE &Macro WITH Probt
Macro = 'W'+Offset
REPLACE &Macro WITH Weigh

*end function SPrim_belf


*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function          :    Ssecd_belf                           -
*  -    Input             :    belief, ruleID, BeliefID, Prob, Weigh -
*  -    Output            :    nul                                  -
*  -    Action            :    locates the belief and stores fact   -
*  -    Date              :    28 Dec 89                            -
*  -    UpDate            :    28 Dec 89                            -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Ssecd_belf
PARAMETER Var, Belf, RuleID, BeliefID, Probt, Weigh
*
*     locate the belief
*
SELECT 8          && second
SET INDEX TO second
SEEK Var+Belf
IF eof()
      ?'Error secondary belief not found (var and belf):'
      ?? var+' b='+Belf
```

```
      WAIT
      browse()
      RETURN(.F.)
ENDIF
*
*     Locate everything using macros
*

IF num_beliefs = 20
      * the current limit on the number of belief IDs that can be stored
      ?'Error overflow in the number of fact'
      WAIT
      RETURN(.F.)
ENDIF
PRIVATE Macro, Offset
REPLACE num_belief WITH num_belief + 1
Offset = ltrim(str(num_belief,2,0))
Macro = 'R'+Offset
REPLACE &Macro WITH RuleID
Macro = 'B'+Offset
REPLACE &Macro WITH BeliefID
Macro = 'P'+Offset
REPLACE &Macro WITH Probt
Macro = 'W'+Offset
REPLACE &Macro WITH Weigh

*end function Ssecd_belf
*
*end file store
```

```
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project          :   KOOLA programing language Shell          =
*  =    Sub-project      :   Simulator                                =
*  =    File             :   main.prg                                 =
*  =    Author           :   Robert D. Rourke                         =
*  =    Date             :   19 Dec 1989                              =
*  =    Update           :   15 Jan 1990                              =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*
*      Databases:
*      Select      Name  Index Code
*      -------+---------+---------+--------
*      1           rule        rule        r
*      2           goal        goal        g
*      3           belief      belief      b
*      4           internal    internal    i
*      5           extreq      extreq      x
*      6           action      action      a
*      7           primary     primary     p
*      8           second      second      s
*      all index set for var_class+name except goal
*
Date ='08 Jan 1990'
Version = 'x.10'

debug_on = .f.
debug_prm = .f.
debug_scd = .f.
debug_stack = .f.
debug_goal = .f.

PRIVATE  Max_store
Max_store = 20


PUBLIC Deflt_path, Escape
* set the colour for a colour monitor
In_colour = .f.
IF iscolor ()
     PUBLIC Colour_Edit
     PUBLIC Colour_Menu
     In_colour = .T.
     Colour_Edit = "GR+/B,GR+/B,B,B,W+/RB"
     Colour_Menu = "GR+,W+/B,B,B,W+/BG"
     setcolor(Colour_Menu)
ENDIF
SET CENTURY ON
SET DATE BRITISH
SET WRAP ON
CLEAR

*Deflt_Path = Get_direct()
MsgWait()
File_error = .F.

SELECT 1
File_name = 'rule.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
```

```
ENDIF

SELECT 2
File_name = 'goal.dbf'
IF file (File_name)
      USE &File_name
*     INDEX ON name to Goal
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 7
File_name = 'primary.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 8
File_name = 'second.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 3
File_name = 'belief.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 4
File_name = 'internal.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 5
File_name = 'extreq.dbf'
IF file (File_name)
      USE &File_name
ELSE
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF

SELECT 6
File_name = 'action.dbf'
IF file (File_name)
      USE &File_name
ELSE
```

```
      File_error = .T.
      ? 'Missing file: '+File_name
ENDIF


IF File_error
      ? '*** File Error termination of KOOLA  ***'
      ?
      RETURN
ENDIF


      MsgCent('MAIN')

      IF AskY('Reset all beliefs?')


            SELECT  5          &&extinqr.dbf
            PRIVATE Num_quest
            Num_quest = reccount()
            GO TOP
            FOR i = 1 TO Num_quest
            REPLACE defined WITH .F.
                  REPLACE available WITH .T.
                  SKIP
            NEXT

            SELECT 4           &&internal.dbf
            Num_quest = reccount()
            GO TOP
            FOR i = 1 TO Num_quest
                  REPLACE defined WITH .F.
                  REPLACE available WITH .T.
                  SKIP
            NEXT


            SELECT 7           &&primary.dbf
            GO TOP
            DO WHILE !eof()
                  * run through all of the beliefs
                  REPLACE defined WITH .F.
                  REPLACE available WITH .T.
                  REPLACE prob WITH -77
                  SKIP
            ENDDO


            SELECT 8           &&second.dbf
            GO TOP
            DO WHILE !eof()
                  * run through all of the beliefs
                  REPLACE defined WITH .F.
                  REPLACE available WITH .T.
                  REPLACE prob WITH -88
                  SKIP
            ENDDO
      ENDIF

PRIVATE Num_goals, Curt_goal
```

```
Variable = 'The var class'
DO WHILE .T.
      @ 0,0 SAY 'KOOLA Simulation Expert Shell'
      MsgCent('MAIN')
*
*     First set all facts to undefined
*     Including both human interface and external facts
*
      start_blf = 0
      Variable = subst( (Variable+space(20)),1,20 )

      @ 9, 0 TO 16, 79
      @ 12,1 SAY 'Please enter a starting goal rule:'GET start_blf
      @ 15, 1 SAY 'What is the current value of the variable class:';
       GET Variable PICTURE"@K"


      READ
      GoalChain(Start_blf, Variable)
***   ? Solve_goal(Start_blf, Variable)
      IF askY('exit the program?')
            EXIT
      ENDIF

      clear
ENDDO
CLEAR
SET COLOUR TO
?
?'Normal termination'
?

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   GoalChain                                 -
*  -    Input           :   starting goal                             -
*  -    Output          :                                             -
*  -    Action          :   does a complete chaining through goals    -
*  -    Date            :   11 Jan 90                                 -
*  -    UpDate          :   11 Jan 90                                 -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

FUNCTION GoalChain
PARAMETER Int_goal, Varb
CLEAR
PRIVATE Curt_goal, Curt_prob, Next_act

SELECT 2    &&goal.dbf
GO TOP
DO WHILE !eof()
      REPLACE Once_thr WITH .F.
      SKIP
ENDDO
Curt_goal = Int_goal
DO WHILE .T.
      *
      *     Solve the curent goal
      *
      CLEAR
      IF debug_goal
```

```
       ? 'current goal: '
       ??Curt_goal
       WAIT 'Top of goal solving algorithm curetn goal '
ENDIF
SELECT 2    &&goal.dbf
GO Curt_goal
IF Once_thr
       ?
       ? '***  Warning Circular reference error  ***'
       ?
       beep()
       beep()
       WAIT
ENDIF
REPLACE Once_thr WITH .T.
Curt_prob = Solve_goal(Belif_pnt, Varb)
CLEAR
SELECT 2    &&goal.dbf
GO Curt_goal
@ 5,0
?'Prob back: '
??Curt_goal
@ row()+1,3 SAY "Change the curt prob" GET Curt_prob
READ
*
*      Find out if antecedent meet
*
IF Curt_prob < 0
       * unavailable error
       CLEAR
       ? 'Can not solve the goal'
       wait
       Solved_goal (0)
       RETURN(1)
ENDIF
IF Check_value (Curt_prob,prob,operand_cd)
       *
       * then case
       *
       IF debug_goal
             ? 'then....'
             wait
       ENDIF
       Curt_goal = tchn_pnt
       Next_act = then_do
ELSE
       *
       * else case
       *
       IF debug_goal
             ? 'else....'
             wait
       ENDIF
       Curt_goal = echn_pnt
       Next_act = else_do

ENDIF
SAVE SCREEN TO S_buf
@ 7, 39 CLEAR TO 17, 73
@ 10,40 TO 16, 70 DOUBLE
@ 10,45 SAY 'DOING ACTION: '
@ 12,50 SAY Next_act
```

```
      WAIT 'Action...'
      IF Curt_goal = 0
            EXIT
      ENDIF
ENDDO
beep('g')
clear
? 'Goal rule soved'
wait
Solved_goal (0)
*end function GoalChain

*
* end file main.prg

*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project          :   KOOLA programing language Shell              =
*  =    Sub-project      :   Simulator                                    =
*  =    File             :   ask.prg                                      =
*  =    Author           :   Robert D. Rourke                             =
*  =    Date             :   08 Jan 1990                                  =
*  =    Update           :   08 Jan 1990                                  =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function         :   Ffact_value                                -
*  -    Input            :   fact pointer                               -
*  -    Output           :   value of the fact                          -
*  -    Action           :   finds the value of any fact                -
*  -    Date             :   11 Jan 90                                  -
*  -    UpDate           :   11 Jan 90                                  -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Convetion: It is eliegal to call this function if the fact is not
*                none
FUNCTION Ffact_value
PARAMETER Fact_ID
IF Fact_ID < 500
      SELECT 4          &&internal.dbf
      GO Fact_ID
ELSE
      SELECT 5          && extreq
      GO Fact_ID-500
ENDIF
IF !defined
      * grave error
      WAIT 'Trying to read an undefined fact...'
      RETURN (-9999)
ENDIF
RETURN (value)

*end function Ffact_value

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function         :   Ffact_state
*  -    Input            :   fact pointer
*  -    Output           :   1= defined, 2 = not availble 3 = undif:    -
*  -    Action           :   Checks the fact                            -
*  -    Date             :   11 Jan 90                                  -
*  -    UpDate           :   11 Jan 90                                  -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Ffact_state
PARAMETER Fact_ID
```

```
*
*     Find the Question
*
IF Fact_ID < 500
      SELECT 4          &&internal.dbf
      GO Fact_ID
ELSE
      SELECT 5          && extreq
      GO Fact_ID-500
ENDIF

IF defined
      *if defined most be available
      RETURN (1)
ELSEIF available
      * if available then only undefined
      RETURN (3)
ELSE
      RETURN (2)
ENDIF
*end function Fint_state

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Solve_fact                                -
*  -    Input           :   fact pointer                              -
*  -    Output          :   The value of the solved fact              -
*  -    Action          :   Solves the given fact                     -
*  -    Date            :   17 Jan 90                                 -
*  -    UpDate          :   17 Jan 90                                 -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Solve_fact
PARAMETER Fact_ID, rule_ID, Curr_var
*
*     Find the Question
*
IF Fact_ID < 500
      * internal human interface
      RETURN(AskUser(Fact_ID, rule_ID, Curr_var))
ELSE
      * external fact based
      RETURN(AskExter(Fact_ID-500, rule_ID, Curr_var))
ENDIF
*end function Solve_fact

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   AskUser                                   -
*  -    Input           :   record number of the rule                 -
*  -    Output          :   nul                                       -
*  -    Action          :   Compiles one primary rule                 -
*  -    Date            :   08 Jan 90                                 -
*  -    UpDate          :   08 Jan 90                                 -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION AskUser
PARAMETER Fact_ID, rule_ID, Curr_var
*
*     Find the Question
*
SELECT 4          &&internal.dbf
GO Fact_ID
PRIVATE Response
IF inquir_typ='TEX'
      Response = AskText()
```

```
ELSEIF inquir_typ='NUM'
      Response = AskNum()
ELSE
      ?'Error unkown type'
      ?inquir_typ
      WAIT
ENDIF

SELECT 4          &&internal.dbf
GO Fact_ID
REPLACE value WITH Response
REPLACE defined WITH .T.
RETURN (value)
*end function AskUser



* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function          :   AskExter                                 -
*  -    Input             :   nul                                      -
*  -    Output            :   value                                    -
*  -    Action            :                                            -
*  -    Date              :   12 Jan 90                                -
*  -    UpDate            :   12 Jan 90                                -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION AskExter
PARAMETER OSfact_ID, rule_ID, Curr_var
*
*     Find the OSfaction
*
SELECT 5          && extreq
GO OSfact_ID
PRIVATE Str_from
Str_from = 1
PRIVATE S_buf
SAVE SCREEN TO S_buf
*
*     Build screen
*
@ Str_from, 6 CLEAR TO  Str_from+5, 76
@ Str_from, 7 TO  Str_from+5, 75 DOUBLE
@ (5+Str_from), 50 SAY '^W for Explanation'
@ Str_from, 12 SAY upper(trim(name))
@ Str_from+1, 9 SAY 'FOR: '+trim(var_class)+' := '+Curr_var
@ Str_from+2, 9 SAY 'TOKEN: '+token
PRIVATE i, Ans, Macro
Ans = 0
DO WHILE .T.
      SELECT 5          && extreq
      GO OSfact_ID
      @ Str_from+3, 65 GET Ans PICTURE '99999.99'
      READ
      IF lastkey() = 23
            ExpRule(rule_ID)
      ELSEIF lastkey() # 27
            EXIT
      ENDIF
ENDDO
RESTORE SCREEN FROM S_buf
SELECT 5          && extreq
GO OSfact_ID
REPLACE value WITH Ans
```

```
REPLACE defined WITH .T.
RETURN (Ans)
*end function AskExter



* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function        :   AskText                                    -
* -    Input           :   nul                                        -
* -    Output          :   Answer                                     -
* -    Action          :   Ask the user a text question               -
* -    Date            :   08 Jan 90                                  -
* -    UpDate          :   08 Jan 90                                  -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  Assumes: database open and pointed to correct record
*     Global: rule_ID defined in the Askuser
*               Curr_var the current value of the variable class
*
FUNCTION AskText
PRIVATE Str_from
Str_from = 0
PRIVATE S_buf
SAVE SCREEN TO S_buf

PRIVATE this_qst
This_qst = recno()
*
*     Build screen
*
@ Str_from, 7 CLEAR TO  (3+num_ams+Str_from), 71
@ Str_from, 7 TO  (3+num_ams+Str_from), 71 DOUBLE
@ (3+num_ams+Str_from), 50 SAY '^W for Explanation'

@ Str_from, 12 SAY upper(trim(name))
@ Str_from+1, 9 SAY 'FOR: '+trim(var_class)+' := '+Curr_var
@ Str_from+2, 9 SAY question

PRIVATE i, Select, Macro
Select = 0
DO WHILE .T.
      SELECT 4          &&internal.dbf
      GO This_qst
      FOR i = 1 TO num_ams
            Macro = 'ANS'+str(i-1,1)
            @ (Str_from+2+i),40 PROMPT str(i,1)+'- '+trim(&Macro)
      NEXT
      MENU TO Select
      IF lastkey() = 23
            *escape pressed, explain
            ExpRule(rule_ID)
      ELSEIF lastkey() # 27
            EXIT
      ENDIF
ENDDO
RESTORE SCREEN FROM S_buf
RETURN (Select)
*end function AskText

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function        :   ExpRule                                    -
* -    Input           :   rule pointer                               -
```

```
*  -    Output          :   nul                                       -
*  -    Action          :                                             -
*  -    Date            :   08 Jan 90                                 -
*  -    UpDate          :   08 Jan 90                                 -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION ExpRule
PARAMETER ID
else_calse = .F.

IF ID > 500
      *then else classe
      else_calse = .T.
      ID = ID - 500
ENDIF

PRIVATE Str_from
Str_from = 2

PRIVATE S_buf
SAVE SCREEN TO S_buf

SELECT 1    &&rule.dbf
SET INDEX TO rule
GO ID
PRIVATE Curr_col
Curr_col = setcolor()
SET COLOUR TO I
@ Str_from+2,10 CLEAR TO (Str_from+13+num_antec),77
DsplRule()
else_calse = .F.
setcolor(Curr_col)
PRIVATE Last_key
DO WHILE .T.
      @ 23,0
      WAIT 'Any other key to continue'
      @ Str_from+2,10 CLEAR TO (Str_from+13+num_antec),77
      Last_key = lastkey()
      IF Last_key = 18
            *PgUp
            IF !bof()
                  SKIP -1
            ENDIF
      ELSEIF Last_key = 3
            *PgDn
            IF !eof()
                  SKIP 1
            ENDIF
      ELSE
            EXIT
      ENDIF
      DsplRule()
ENDDO
SET INDEX TO
RESTORE SCREEN FROM S_buf
*end function ExpRule

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -   -
*  -    Function        :   DsplRule                                  -
*  -    Input           :   nul                                       -
*  -    Output          :   nul                                       -
*  -    Action          :                                             -
*  -    Date            :   09 Jan 90                                 -
```

```
*  -    UpDate          :   09 Jan 90                              -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Assumes the current record to be displayed
FUNCTION DsplRule
*@ Str_from+2,10 CLEAR TO (Str_from+13+num_antec),77
@ Str_from+3, 13 TO (Str_from+12+num_antec), 74
@ Str_from+3, 18 SAY 'EXPLANATION'
@ Str_from+3, 65 SAY '^X PgUp'
@ (Str_from+12+num_antec), 65 SAY '^Y PgDn'

@ Str_from+4, 15 SAY 'Rule number: '+str(recno(),4)
@ Str_from+6, 15 SAY 'FOR: '+var_class
@ Str_from+7, 15 SAY 'Primary Rule: ' +name
IF extrn0
      @   Str_from+8,   18   SAY'IF,   COSI:   '+trim(if0)+'   '+oper0+'
'+str(num0,8,2)
ELSEIF  !empty(ans0)
      @ Str_from+8, 18 SAY'IF, Quest: '+trim(if0)+' '+oper0+' '+trim(ans0)
ELSE
      @  Str_from+8,  18  SAY'IF,  Num  Quest:  '+trim(if0)+'  '+oper0+'
'+str(num0,8,2)
ENDIF
PRIVATE i, Macro, Pointer,M_oper,M_if,M_ans
FOR i = 1 TO num_antec-1
      Pointer = str(i,1)
      Macro = 'extrn'+Pointer
      M_if = 'IF'+Pointer
      M_oper= 'OPER'+Pointer
      IF &Macro          &&external
            M_ans = 'NUM'+Pointer
            @   (Str_from+8+i),   18   SAY'AND,   COSI:   '+trim(&M_if)+'
'+&M_oper+' '+str(&M_ans,8,2)
      ELSE
            M_ans = 'ANS'+Pointer
            IF  !empty(&M_ans)
                  @ (Str_from+8+i), 18 SAY 'AND, Quest: '+trim(&M_if)+'
'+&M_oper+' '+trim(&M_ans)
            ELSE
                  M_ans = 'NUM'+Pointer
                  @   (Str_from+8+i),   18   SAY   'AND,   Num   Quest:
'+trim(&M_if)+' '+&M_oper+' '+str(&M_ans,8,2)
            ENDIF
      ENDIF
NEXT
@ (Str_from+9+i), 18 SAY 'THEN: '+trim(then)+' '+str(then_p,3)+' Weight:
'+str(then_w,3)
IF  (else # 'NUL')
      IF  else_calse
            @   (Str_from+10+i),   15   SAY   '-->ELSE:   '+trim(else)+'
'+str(else_p,3)+' Weight: '+str(else_w,3)
      ELSE
            @   (Str_from+10+i),   18   SAY   'ELSE:   '+trim(else)+'
'+str(else_p,3)+' Weight: '+str(else_w,3)
      ENDIF
ENDIF
*end function DsplRule


*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   AskNum                                 -
*  -    Input           :   nul                                    -
*  -    Output          :   nul                                    -
```

```
*  -    Action          :                                       -
*  -    Date            :   08 Jan 90                           -
*  -    UpDate          :   08 Jan 90                           -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Global: rule_ID defined in the Askuser
*             Curr_var the current value of the variable class
FUNCTION AskNum
PRIVATE Str_from
Str_from = 1
PRIVATE S_buf
SAVE SCREEN TO S_buf
PRIVATE this_qst
This_qst = recno()
*
*     Build screen
*
@ Str_from, 7 CLEAR TO  Str_from+5, 75
@ Str_from, 7 TO  Str_from+5, 75 DOUBLE
@ (5+Str_from), 50 SAY '^W for Explanation'
@ Str_from, 12 SAY upper(trim(name))
@ Str_from+1, 9 SAY 'FOR: '+trim(var_class)+'  := '+Curr_var
@ Str_from+2, 9 SAY question
@ Str_from+4, 20 SAY 'Range: '
??ltrim(str(lower,8,2))
??', TO: '
??ltrim(str(upper,8,2))
PRIVATE i, Ans, Macro
Ans = lower + (upper-lower)/2
DO WHILE .T.
      SELECT 4          &&internal.dbf
      GO This_qst
      @ Str_from+3, 65 GET Ans PICTURE '99999.99' RANGE lower, upper
      READ
      IF lastkey() = 23
            ExpRule(rule_ID)
      ELSEIF lastkey() # 27
            EXIT
      ENDIF
ENDDO
RESTORE SCREEN FROM S_buf
RETURN (Ans)
*end function AskNum
*
*end file ask.prg

*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -
*  =    Project         :   KOOLA programing language Shell          -
*  =    Sub-project     :   Simulator
*  =    File            :   prmblf.prg                               -
*  =    Author          :   Robert D. Rourke                         -
*  =    Date            :   19 Jan 1990
*  =    Update          :   31 Jan 1990                              -
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -
* debug_prm

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Expand_prim
*  -    Input           :   Values to be checked check type         -
*  -    Output          :   value of the fact                       -
*  -    Action          :   finds the value of any fact
*  -    Date            :   17 Jan 90                               -
*  -    UpDate          :   17 Jan 90                               -
```

```
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     This function is called if a primary belief is taken from the
*     queue.  It expands it out by supporting fact.  All facts that
*     are not known are put on the queue
*
FUNCTION Expand_prim
PARAMETER Prim_blf

SET DECIMALS TO 2
PRIVATE Final_prob, Solved
Solved = .F.


PRIVATE Sigma_prob, Sigma_weight
Sigma_prob   = 0
Sigma_weight = 0


SELECT 7          &&primary.dbf
GO Prim_blf
Curt_blief = name
Total_facts = num_facts
* debug_prm
IF debug_prm
      ? 'Expanding, Name of primary belief: '
      ?? name
      ? '   This belief hase supporting facts : '
      ??Total_facts
      wait
ENDIF
DECLARE   Fact_lst[Total_facts],   Oper_lst[Total_facts],
Fnum_lst[Total_facts]
DECLARE Frul_lst[Total_facts+1], Valu_lst[Total_facts]
Frul_lst[Total_facts+1] = -9999
FOR i = 1 TO Total_facts
      *
      * load all of the facts for this belief then decide what to
      * do with them
      *
      Macro = 'F'+ltrim(str(i,2))   && the factID
      Fact_lst[i] = &Macro
      Macro = 'O'+ltrim(str(i,2))   && the operator to test it
      Oper_lst[i] = &Macro
      Macro = 'R'+ltrim(str(i,2))   && rule reference number
      Frul_lst[i] = &Macro
      Macro = 'N'+ltrim(str(i,2))
      Fnum_lst[i] = &Macro          && the number it should equale
NEXT
*
*     Find out which of these facts are defined
*

PRIVATE True_flag, Solvable
PRIVATE Cur_prob, Cur_weight, Cur_rule
Reset_Expand_prim(1)
*
*     the facts are ordered by rule grouping
*
PRIVATE  CurtFct
CurtFct = 1
DO WHILE CurtFct <= Total_facts
      State_fct = Ffact_state(Fact_lst[CurtFct])
```

```
      IF debug_prm
            clear
            ? Curt_blief
            ?? '    Index :'
            ?? CurtFct
            ? 'R = '
            ?? Frul_lst[CurtFct]
            ? 'F = '
            ?? Fact_lst[CurtFct]
            ? 'S = '
            ?? State_fct
      ENDIF
      IF (State_fct = 3)
            * not defined
            IF debug_prm
                  ? 'Not defined Soving the fact '
                  wait
            ENDIF
            Solve_fact(Fact_lst[CurtFct], Frul_lst[CurtFct], Curr_var)
      ENDIF
      State_fct = Ffact_state(Fact_lst[CurtFct])
      IF (State_fct = 2)
            *not available
            Solvable = .F.
            wait
      ELSEIF (State_fct = 3)
            * not defined
            Solvable = .F.
            IF debug_prm
                  ? 'Fact still not defined'
                  WAIT
            ENDIF
      ELSE
            * At this point it must be defined
            Dyn_value = Ffact_value(Fact_lst[CurtFct])
            IF !Check_value(Dyn_value,Fnum_lst[CurtFct],Oper_lst[CurtFct])
                  IF debug_prm
                        ?'Found false, should finish'
                  ENDIF
                  True_flag = .F.
                  Solvable = .T.
                  *
                  * if the result is false, then the answer is completly
                  *  known for this sub rule.  Consequently the search
should
                  * stop.  Note the answer is false
                  *
                  * Start the next
                  DO WHILE Frul_lst[CurtFct+1] = Cur_rule
                        * remove all facts that apply to this rule
                        CurtFct = CurtFct + 1
                  ENDDO
            ENDIF
      ENDIF known
      *
      *     Check if it is the last part of a rule.  If it is then
      *     reset the true flag.  Also add its probability to the belief
      *
      IF Cur_rule # Frul_lst[CurtFct+1]
            *
            * next fact is not part of this rule set
            * may be the end of the fact list
```

```
            *
            IF debug_prm
                  beep()
                  ?
                  ?
            ENDIF
            IF !Solvable
                  IF debug_prm
                        ?'Fact Can not be sloved'
                  ENDIF
            ELSE
                  IF Cur_rule > 500
                        *else clause
                        True_flag = !True_flag
                  ENDIF
                  IF True_flag
                        Solved = .T.
                        *
                        *     Calculate the prob place in a tempory array
                        *
                        Sigma_prob   = Sigma_prob + Cur_prob*Cur_weight
                        Sigma_weight = Sigma_weight+Cur_weight
                        IF debug_prm
                              ? 'Sigma weight:: '
                              ?? Sigma_weight
                              ? 'Sigma_prob:: '
                              ?? Sigma_prob
                              ? 'Cur_prob  '
                              ?? Cur_prob
                              ? ' Cur_weight '
                              ??  Cur_weight
                        ENDIF
                  ELSE
                        IF debug_prm
                              ?'This set of facts did not affect the
belief'
                        ENDIF
                  ENDIF
                  IF CurtFct < Total_facts
                        Reset_Expand_prim(CurtFct+1)
                  ENDIF
            ENDIF over set
      ENDIF
      CurtFct = CurtFct + 1
ENDDO

SELECT 7          &&primary.dbf
GO Prim_blf
IF Solved
      IF debug_prm
            ? 'Sigma weight:: '
            ?? Sigma_weight
            ? 'Sigma_prob:: '
            ?? Sigma_prob
            ? 'Final prob:: '
            ??  Sigma_prob/Sigma_weight*100
      ENDIF
      REPLACE prob WITH Sigma_prob/Sigma_weight*100
      REPLACE Defined WITH .T.
      REPLACE Available WITH .T.
      * the probability -1 if undefined -2 if not available
      RETURN (Sigma_prob/Sigma_weight*100)
```

```
ELSE
      REPLACE prob WITH -2
      REPLACE Defined WITH .F.
      REPLACE Available WITH .F.
      RETURN (-2)
ENDIF
*end function Expand_prim

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function         :   Reset_Expand_prim                          -
* -    Input            :                                              -
* -    Output           :                                              -
* -    Action           :                                              -
* -    Date             :   17 Jan 90                                  -
* -    UpDate           :   17 Jan 90                                  -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Reset_Expand_prim
PARAMETER Offset
Solvable = .T.
True_flag = .T.
Cur_rule = Frul_lst[Offset]
SELECT 7          &&primary.dbf
GO Prim_blf
Macro = 'P'+ltrim(str(Offset,2))
Cur_prob = &Macro/100
Macro = 'W'+ltrim(str(Offset,2))
Cur_weight = &Macro/100
*end function Reset_Expand_prim

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function         :   Check_value                                -
* -    Input            :   Values to be checked check type            -
* -    Output           :   value of the fact                          -
* -    Action           :   finds the value of any fact                -
* -    Date             :   17 Jan 90                                  -
* -    UpDate           :   17 Jan 90                                  -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Check_value
PARAMETERS Value1, Value2, Operat

DO CASE
CASE Operat = 1
      * =
      IF Value1 = Value2
            IF debug_prm
                  ? 'Found TRUE...'
            ENDIF
            RETURN (.T.)
      ENDIF
CASE Operat = 2
      * <
      IF Value1 < Value2
            IF debug_prm
                  ? 'Found TRUE'
            ENDIF
            RETURN (.T.)
      ENDIF
CASE Operat = 3
      *>
      IF Value1 >  Value2
            IF debug_prm
                  ? 'Found TRUE'
```

```
            ENDIF
            RETURN (.T.)
      ENDIF
CASE Operat = 4
      *<=
      IF Value1 <= Value2
            IF debug_prm
                  ? 'Found TRUE'
            ENDIF
            RETURN (.T.)
      ENDIF
CASE Operat = 5
      *>=
      IF Value1 >= Value2
            IF debug_prm
                  ? 'Found TRUE'
            ENDIF
            RETURN (.T.)
      ENDIF
CASE Operat = 6
      * #
      IF Value1 # Value2
            IF debug_prm
                  ? 'Found TRUE'
            ENDIF
            RETURN (.T.)
      ENDIF
OTHERWISE
      ?'Error in the operator index'
      WAIT
ENDCASE
IF debug_prm
      ? 'Found to be False...'
ENDIF
RETURN (.F.)
*end function Check_value

*
*end file prmblf

*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project         :   KOOLA programing language Shell            =
*  =    Sub-project     :   Simulator                                  =
*  =    File            :   stack.prg                                  =
*  =    Author          :   Robert D. Rourke                           =
*  =    Date            :   18 Jan 1990                                =
*  =    Update          :   18 Jan 1990                                =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =


*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Modual          :   Stack modual                             -
*  -    Purpose         :   finds the value of any fact              -
*  -    Date            :   18 Jan 90                                -
*  -    UpDate          :   18 Jan 90                                -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Stack_constructor                        -
*  -    Input           :   nil                                      -
*  -    Output          :                                            -
*  -    Action          :   Initisilised the stack                   -
```

```
*  -    Date            :   18 Jan 90                             -
*  -    UpDate          :   18 Jan 90                             -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Stack_constructor
PUBLIC Max_stack
Max_stack = 100
PUBLIC Stack_ID[Max_stack]
PUBLIC Stack_pointer
Stack_pointer = 1
?'Stack initialised'
*end function Stack_constructor

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Stack_empty                           -
*  -    Input           :   nil                                   -
*  -    Output          :   true if empty                         -
*  -    Action          :                                         -
*  -    Date            :   18 Jan 90                             -
*  -    UpDate          :   18 Jan 90                             -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Stack_empty
RETURN(Stack_pointer <= 1)


*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Push_stack                            -
*  -    Input           :   Fact or belief ID / type true= belief -
*  -    Output          :   value of the fact                     -
*  -    Action          :   Places one "belief" on the stack      -
*  -    Date            :   18 Jan 90                             -
*  -    UpDate          :   18 Jan 90                             -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Push_stack
PARAMETER ID
IF debug_stack
      WAIT 'Push down stack : '+str(ID)
ENDIF
IF Stack_pointer > Max_stack
      ?'Error, stack overflow..'
ELSE
      Stack_ID[Stack_pointer] = ID
      Stack_pointer = Stack_pointer + 1
ENDIF

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Pop_stack                             -
*  -    Input           :   nil                                   -
*  -    Output          :   An id of a fact or blief              -
*  -    Action          :   Places one "belief" on the stack      -
*  -    Date            :   18 Jan 90                             -
*  -    UpDate          :   18 Jan 90                             -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     Globals: sets/resets GIs_belief
FUNCTION Pop_stack
IF Stack_pointer <= 1
      ?'Error, stack underflow..'
      WAIT
ELSE
      IF debug_stack
            WAIT 'Pop stack : '+str(Stack_ID[Stack_pointer-1])
      ENDIF
      Stack_pointer = Stack_pointer - 1
```

```
        RETURN (Stack_ID[Stack_pointer])
ENDIF
RETURN (-9999)
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Examine_stack                              -
*  -    Input           :   nil                                        -
*  -    Output          :   An id of a fact or blief                   -
*  -    Action          :   retruns the belief on the statck           -
*  -    Date            :   18 Jan 90                                  -
*  -    UpDate          :   18 Jan 90                                  -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*       Globals: sets/resets GIs_belief
FUNCTION Examine_stack
IF Stack_pointer <= 1
        ?'Error, stack underflow..'
        WAIT
ELSE
        IF debug_stack
                WAIT 'Examine stack : '+str(Stack_ID[Stack_pointer-1])
        ENDIF
        RETURN (Stack_ID[Stack_pointer-1])
ENDIF
RETURN (-9999)
*end function Examine_stack
*
*end file stack.prg

*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*  =    Project         :   KOOLA programing language Shell             =
*  =    Sub-project     :   Simulator                                   =
*  =    File            :   secbelf.prg                                 =
*  =    Author          :   Robert D. Rourke                            =
*  =    Date            :   26 Jan 1990                                 =
*  =    Update          :   31 Jan 1990                                 =
*  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*


*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -    Function        :   Solve_goal                                 -
*  -    Input           :   fact pointer                               -
*  -    Output          :   value of the fact                          -
*  -    Action          :   finds the value of any fact                -
*  -    Date            :   11 Jan 90                                  -
*  -    UpDate          :   11 Jan 90                                  -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Solve_goal
PARAMETER Goal_blf, Curr_var
*
*       Add a goal belief
*
Stack_constructor()
Push_stack (Goal_blf)
*
*       Go into the main loop
*
PRIVATE Head_blf, State_head_blf
DO WHILE .T.
        CLEAR
        IF debug_on
                ?'*** Top of the algorithm'
        ENDIF
```

```
        Head_blf = Examine_stack()
        State_head = Fbelief_state(Head_blf)

        IF State_head = 1
                *defined
                IF Head_blf = Goal_blf
                        *success!
                        Solved_goal(Goal_blf)
                        EXIT
                ELSE
                        *just remove the belief
                        ?'Popping a defined belief'
                        Pop_stack()
                ENDIF

        ELSEIF State_head = 2
                *not available
                IF Head_blf = Goal_blf
                        *fail
                        Solved_goal(Goal_blf)
                        RETURN(-2)
                ELSE
                        *just remove the belief
                        IF debug_on
                                ?'Popping a not available belief'
                        ENDIF
                        Pop_stack()
                ENDIF

        ELSEIF State_head = 3
                *not defined leave on stack
                IF Fbelief_type(Head_blf)                       && if secondary
                        IF debug_on
                                WAIT 'belief is a secondary belief and will be
expanded'
                        ENDIF
                        Expand_scdn(Head_blf)
                ELSE
                        IF debug_on
                                WAIT 'belief is a primary belief and will be
expanded'
                        ENDIF
                        Expand_prim(Head_blf-500)
                ENDIF
        ELSE
                ?'Error in belief state unknown'
                WAIT
        ENDIF
ENDDO
RETURN(Fbelief_prob(Goal_blf))

*end function solve_goal


*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -   Function          :   Expand_scdn                                   -
*  -   Input             :   A secondary belief                            -
*  -   Output            :   value of the belief                           -
*  -   Action            :   tries to find the value of a sec belief       -
*  -   Date              :   17 Jan 90                                     -
```

```
* -    UpDate          :    17 Jan  90                                   -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     This function is called if a secondary belif is taken from the
*     queue.  It expands it out by supporting fact.  All facts that
*     are not known are put on the queue
*
FUNCTION Expand_scdn
PARAMETER Secd_blf

SELECT 8           &&second.dbf
GO Secd_blf
Curt_blief = name
Numb_blf = num_belief
IF debug_scd
      ?? name
      ? '   This belief hase supporting beliefs : '
      ??Numb_blf
      wait
ENDIF
DECLARE PrmBlf_lst[Numb_blf], PrmR_lst[Numb_blf+1]
DECLARE Valu_lst[Numb_blf]
DECLARE Parm_wt[Numb_blf]
DECLARE Parm_pr[Numb_blf]

PrmR_lst[Numb_blf+1] = -9999

FOR i = 1 TO Numb_blf
      *
      * load all of the supporting beliefs
      *
      Macro = 'B'+ltrim(str(i,2))    && the factID
      PrmBlf_lst[i] = &Macro
      Macro = 'R'+ltrim(str(i,2))    && rule reference number
      PrmR_lst[i] = &Macro
      Macro = 'W'+ltrim(str(i,2))
      Parm_wt[i]   = (&Macro)
      Macro = 'P'+ltrim(str(i,2))
      Parm_pr[i]  = (&Macro)
NEXT
*
*     Find out which of these beliefs are defined
*     If so use there number, if not put on stack
*
PRIVATE Solve_pas1          &&determines if it is not solvable pass one
Solve_pas1 = .T.


PRIVATE Cur_prob, Cur_weight, Cur_rule
Cur_rule = PrmR_lst[1]
FOR CrtBlf = 1 TO Numb_blf
      State_blf = Fbelief_state(PrmBlf_lst[CrtBlf])
      IF debug_scd
            ? 'R = '
            ?? PrmR_lst[CrtBlf]
            ? 'Bel = index'
            ?? PrmBlf_lst[CrtBlf]
            ? 'Value of rule weight: '
            ?? Parm_wt[CrtBlf]
            ? 'S = '
            ?? State_blf
      ENDIF
      IF (State_blf = 3)
```

```
        * not defined
        IF debug_scd
            ? 'Belief not defined placing on stack'
        ENDIF
        Valu_lst[CrtBlf] = -1
        Solve_pas1 = .F.
        Push_stack (PrmBlf_lst[CrtBlf])
    ELSEIF (State_blf = 2)
        * not avaliable
        Valu_lst[CrtBlf] = -2
        IF debug_scd
            ? 'Belief not avalable'
        ENDIF
    ELSEIF (State_blf = 1)
        *Available
        Valu_lst[CrtBlf] = Fbelief_prob(PrmBlf_lst[CrtBlf])
        IF debug_scd
            ? Valu_lst[CrtBlf]
            WAIT 'Belief Available saving its value for future'
        ENDIF

    ELSE
        ? 'Error state value wrong'
        WAIT
    ENDIF

NEXT
PRIVATE Cur_rule
IF Solve_pas1
    * may be solved
    PRIVATE Cur_rule
    Cur_rule = PrmR_lst[1]
    PRIVATE Solve_pas2
    Sove_pas2 = .F.
    PRIVATE Sig_weight, Sig_prob
    Sig_weight = 0
    Sig_prob = 0
    CrtBlf = 1
    DO WHILE CrtBlf <= Numb_blf
        IF debug_scd
            WAIT 'New rule'
        ENDIF
        Cur_weight = Parm_wt[CrtBlf]
        Cur_prob   = Parm_pr[CrtBlf]
        Cur_rule = PrmR_lst[CrtBlf]
        DO WHILE Cur_rule = PrmR_lst[CrtBlf]
            State_blf = Fbelief_state(PrmBlf_lst[CrtBlf])
            IF debug_scd
                ? CrtBlf
                ? 'The rule :'
                ??PrmR_lst[CrtBlf]
                ? 'The value of the belief: '
                ?? Valu_lst[CrtBlf]
                ? 'The rule  probs: and weight'
                ?? Parm_pr[CrtBlf]
                ?? Parm_wt[CrtBlf]
            ENDIF

            IF (State_blf = 3)
                *not defined
                ?'Error in belief routine should not have reached
unsolved'
```

```
                WAIT
            ELSEIF (State_blf = 2)
                *Not available
            ELSEIF (State_blf = 1)
                *
                * known, Calculate the an accumalated prob based
on the worst
                * worst supporting belief in a common rule
                *
                Sove_pas2 = .T.
                Worst_prob = Valu_lst[CrtBlf]
                DO WHILE PrmR_lst[CrtBlf] = PrmR_lst[CrtBlf+1]
                    CrtBlf = CrtBlf + 1
                    IF Worst_prob > Valu_lst[CrtBlf] .AND.;
                     Fbelief_state(PrmBlf_lst[CrtBlf]) = 1
                        Worst_prob = Valu_lst[CrtBlf]
                    ENDIF
                ENDDO
                Sig_weight = Sig_weight + Cur_weight/100
                Sig_prob = Sig_prob +
Cur_prob*Worst_prob*Cur_weight/1000000
                IF debug_on
                    ? 'Worst prob: '
                    ?? Worst_prob
                    ? 'Sigma prob and weight: '
                    ?? Sig_prob
                    ?? Sig_weight
                    WAIT
                ENDIF
            ELSE
                ?'Error internal state'
                WAIT
            ENDIF
            CrtBlf = CrtBlf + 1
        ENDDO while same rule
    ENDDO while crtBlf < number of belf
    IF Sove_pas2
        return_prob = Sig_prob/Sig_weight
    ELSE
        return_prob = -2
    ENDIF
ELSE
    return_prob = 0
ENDIF
IF debug_on
    @ row()+1, 10 SAY "What value of prob :" GET return_prob
    READ
ENDIF
SELECT 8          &&second.dbf
GO Secd_blf
IF return_prob < 0
    REPLACE Defined WITH .F.
    REPLACE Available WITH .F.
ELSEIF return_prob = 0
    REPLACE Defined WITH .F.
    REPLACE Available WITH .T.
ELSE
    REPLACE prob WITH return_prob*100
    REPLACE Defined WITH .T.
    REPLACE Available WITH .T.
ENDIF
RETURN (return_prob)
```

```
*end function Expand_scdn


*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -   Function          :  Solved_goal                                 -
*  -   Input             :  belief pointe                               -
*  -   Output            :                                              -
*  -   Action            :  finds the type of belief                    -
*  -   Date              :  11 Jan 90                                   -
*  -   UpDate            :  11 Jan 90                                   -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Solved_goal
PARAMETER Goal_blief
CLEAR
IF Goal_blief > 0
       IF Goal_blief > 500
              SELECT 7          &&primary.dbf
              GO Goal_blief-500
       ELSE
              SELECT 8          &&second.dbf
              GO Goal_blief
       ENDIF
       IF defined
              ??  'The  goal  belief:  '+trim(var_class)+':'+name+'     Pr=
'+str(prob,3)+'%'
       ELSE
              ??  'The  goal  belief:  '+trim(var_class)+':'+name+'   (Not
defined)'
       ENDIF
       IF !AskN('Would you like to see a sumery of all beliefs?')
              RETURN(1)
       ENDIF
ENDIF
@ 2,2

SELECT 7            &&primary.dbf
GO TOP
? 'The primary beliefs: '
DO WHILE !eof()
       * run through all of the beliefs
       IF defined
              ?trim(var_class)+':'+name+'   Pr=  '+str(prob)+'%'
       ENDIF
       IF !available
              ?trim(var_class)+':'+name+'  (Not available)'
       ENDIF
       SKIP
ENDDO


SELECT 8            &&second.dbf
GO TOP
?
? 'The secondary beliefs:'
DO WHILE !eof()
       * run through all of the beliefs
       IF defined
              ? name+'  Pr= '+str(prob)+'%'
       ENDIF
       IF !available
              ? name+' (Not available)'
```

```
*end function Expand_scdn


* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -     Function          :   Solved_goal                           -
* -     Input             :   belief pointe                         -
* -     Output            :                                         -
* -     Action            :   finds the type of belief              -
* -     Date              :   11 Jan 90                             -
* -     UpDate            :   11 Jan 90                             -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Solved_goal
PARAMETER Goal_blief
CLEAR
IF Goal_blief > 0
       IF Goal_blief > 500
              SELECT 7          &&primary.dbf
              GO Goal_blief-500
       ELSE
              SELECT 8          &&second.dbf
              GO Goal_blief
       ENDIF
       IF defined
              ?? 'The goal belief: '+trim(var_class)+':'+name+'    Pr=
'+str(prob,3)+'%'
       ELSE
              ?? 'The goal belief: '+trim(var_class)+':'+name+' (Not
defined)'
       ENDIF
       IF !AskN('Would you like to see a sumery of all beliefs?')
              RETURN(1)
       ENDIF
ENDIF
@ 2,2

SELECT 7          &&primary.dbf
GO TOP
? 'The primary beliefs: '
DO WHILE !eof()
       * run through all of the beliefs
       IF defined
              ?trim(var_class)+':'+name+'  Pr= '+str(prob)+'%'
       ENDIF
       IF !available
              ?trim(var_class)+':'+name+' (Not available)'
       ENDIF
       SKIP
ENDDO


SELECT 8          &&second.dbf
GO TOP
?
? 'The secondary beliefs:'
DO WHILE !eof()
       * run through all of the beliefs
       IF defined
              ? name+'  Pr= '+str(prob)+'%'
       ENDIF
       IF !available
              ? name+' (Not available)'
```

```
     ENDIF
     SKIP
ENDDO
WAIT

*end function solved_goal

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - .
* -    Function        :    Fbelief_type                        -
* -    Input           :    belief pointe                       ..
* -    Output          :
* -    Action          :    finds the type of belief            -
* -    Date            :    11 Jan 90
* -    UpDate          :    11 Jan 90
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Fbelief_type
PARAMETER belief_ID
RETURN (belief_ID<500)



* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function        :    Fbelief_state                              -
* -    Input           :    belief pointer                             -
* -    Output          :    1= defined, 2 = not availble 3 = undifi   -
* -    Action          :    Checks the belief                          -
* -    Date            :    11 Jan 90                                 -
* -    UpDate          :    11 Jan 90                                 -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Fbelief_state
PARAMETER belief_ID
IF belief_ID > 500
     SELECT 7         &&primary.dbf
     GO belief_ID-500
ELSE
     SELECT 8         &&second.dbf
     GO belief_ID
ENDIF

IF defined
     *if defined most be available
     RETURN (1)
ELSEIF available
     * if available then only undefined
     RETURN (3)
ELSE
     RETURN (2)
ENDIF
*end function Fbelief

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* -    Function        :    Fbelief_prob                              -
* -    Input           :    belief pointer                            -
* -    Output          :    1= defined, 2 = not availble 3 = undifi  -
* -    Action          :    Checks the belief                         -
* -    Date            :    11 Jan 90                                -
* -    UpDate          :    11 Jan 90                                -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Fbelief_prob
PARAMETER belief_ID
IF belief_ID > 500
```

```
        ENDIF
        SKIP
ENDDO
WAIT

*end function solved_goal

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -   Function        :  Fbelief_type                          -
*  -   Input           :  belief pointe                         -
*  -   Output          :                                        -
*  -   Action          :  finds the type of belief              -
*  -   Date            :  11 Jan 90                             -
*  -   UpDate          :  11 Jan 90                             -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Fbelief_type
PARAMETER belief_ID
RETURN (belief_ID<500)



*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -   Function        :  Fbelief_state                          -
*  -   Input           :  belief pointer                         -
*  -   Output          :  1= defined, 2 = not availble 3 = undifi -
*  -   Action          :  Checks the belief                      -
*  -   Date            :  11 Jan 90                             -
*  -   UpDate          :  11 Jan 90                             -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Fbelief_state
PARAMETER belief_ID
IF belief_ID > 500
        SELECT 7        &&primary.dbf
        GO belief_ID-500
ELSE
        SELECT 8        &&second.dbf
        GO belief_ID
ENDIF

IF defined
        *if defined most be available
        RETURN (1)
ELSEIF available
        * if available then only undefined
        RETURN (3)
ELSE
        RETURN (2)
ENDIF
*end function Fbelief

*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  -   Function        :  Fbelief_prob                          -
*  -   Input           :  belief pointer                        -
*  -   Output          :  1= defined, 2 = not availble 3 = undifi -
*  -   Action          :  Checks the belief                     -
*  -   Date            :  11 Jan 90                             -
*  -   UpDate          :  11 Jan 90                             -
*  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FUNCTION Fbelief_prob
PARAMETER belief_ID
IF belief_ID > 500
```

```
      SELECT 7                &&primary.dbf
      GO belief_ID-500
ELSE
      SELECT 8                &&second.dbf
      GO belief_ID
ENDIF

IF defined
      RETURN (prob)
ELSE
      ?'Error program is reading an undfined probability'
      WAIT
      RETURN (0)
ENDIF
*end function Fbelief_prob

*
*end file secbelf.prg
```

```
      SELECT 7           &&primary.dbf
      GO belief_ID-500
ELSE
      SELECT 8           &&second.dbf
      GO belief_ID
ENDIF

IF defined
      RETURN (prob)
ELSE
      ?'Error program is reading an undfined probability'
      WAIT
      RETURN (0)
ENDIF
*end function Fbelief_prob

*
*end file secbelf.prg
```