FOR REFERENCE ONLY

PhD
GIS/PHAR

SLC
Ref.

THE DEVELOPMENT OF A KNOWLEDGE BASED FRONT END FOR A COMPUTATIONAL FLUID DYNAMICS PACKAGE

BY

Stuart Lee Hartle, BEng(Hons), AMIMechE

A thesis submitted in partial fulfilment of the requirements
of The Nottingham Trent University
for the Degree of Doctor of Philosophy

August 1993

Dedicated to Suzanne

The Development of a Knowledge Based Front End
for a Computational Fluid Dynamics Package

by

Stuart Lee Hartle, BEng(Hons), AMIMechE

ABSTRACT

The overall aim of this study was to establish a knowledge based approach to the preparation of data for complex computer programs. This was achieved through the development of a Knowledge Based Front End which interacts with a user to extract data, performs inference on this data and then synthesises the data to generate appropriate commands acceptable to the original program.

Initial development of a Knowledge Based Front End to a Computational Fluid Dynamics package, PHOENICS, using a commercial expert system shell, LEONARDO, was found to be inadequate. The limitations of the shell lead to the re-development of the Front End using the traditional Artificial Intelligence language, LISP.

LISP was used to create knowledge representation formalisms, data storage techniques and a purpose built inference engine for the target application. Knowledge representation formalisms included factual templates, objects and a specifically designed rule base language. The creation and implementation of inference networks reduced the number of rules the system needs to consider when using a specific rulebase. Base rules within each rulebase are used as the roots with which forward chaining commences. Antecedents that cannot be proved through forward chaining are then used as the goal for backward chaining throughout the associated inference network. The Knowledge Based Front End for PHOENICS improved the accuracy and consistency of the prepared data file. The system synthesises the user entered data and inferred data into appropriate PHOENICS commands to fully describe a computational analysis of fluid flow. A knowledge domain for jet impingement was used as a vehicle to demonstrate the concepts incorporated within the system.

The program architecture was carefully designed to enable future extensibility. Replacement, or extension, of the existing database and knowledge bases with new assertion templates, objects and rules, which would be inferred upon by the same inference engine is feasible. This potential for extensibility allows the system to be applied to different knowledge domains.

An important aspect of Computational Fluid Dynamics is the correct specification of the meshed geometry. Aspect ratios within the grid can have disastrous effects on the convergence of the solution and the accuracy of the results, and are therefore of paramount importance. A novel method of aspect ratio dependent finite volume grid generation is presented which utilises a generalised Fourier Series profile function. This technique ensures that given an arbitrary, one dimensional, region, its overall height, the minimum cell size, and the maximum allowed cell aspect ratio, the region can be meshed using grid clustering near a wall or within a duct. Meshing each axis as a one dimensional region enables a complete mesh to be obtained by superimposing the axes together. Within the final domain, the cell aspect ratio will not exceed the predefined maximum. Feasibility studies into the monitoring and control of the PHOENICS solution algorithm and results analysis through post processing grid optimisation, were performed. The potential for the latter two areas to be integrated into the KBFE looks promising.

ACKNOWLEDGEMENTS

To my Director of Studies, Dr K Jambunathan, I would like to express my sincere gratitude for his tireless enthusiasm throughout this project. I would also like to extend the same appreciation to Dr Eugene Lai, my first supervisor. Furthermore, their continuous moral and technical support was always there in times of need. I am also grateful to my second supervisor and Head of Department, Eur Ing Professor B L Button, for providing the resources and facilities.

Special thanks are directed towards the Polytechnics and Colleges Funding Council (PCFC) who have financially supported this project.

Researchers are always in need of a first class library service for access to references and publications in whatever medium. I am extremely grateful to the engineering library staff, especially the librarian, Mr J Corlett, for providing an excellent information retrieval service which, in my opinion, is second to none. Thanks are also due to staff in the Computer Services department for their extreme patience, in particular Dr RHA Eade and Mr R Gage.

On a more personal note, I would like to express the value of my friendship with my colleagues Mr Shabir Kapasi and Mrs Shirley Ashforth-Frost. Their patience and continual support, was very welcome, especially when I needed someone to discuss ideas with. One further colleague who needs a special mention is Mr Karl Wyer, for his expert help with various computer hardware and software difficulties experienced throughout this research.

Finally, my years of higher education and the pursuit of further, formal, qualifications is coming to a close. It is here where I would like to mention those people closest to me that have not only had to suffer my intolerable attitude, in times of despair, but have also shared my good times. Furthermore, they have always had confidence in me, something that I often lack. Firstly: my wife Suzanne, for her confidence, endless patience and understanding; and secondly my parents, Gloria and Dennis, who have always believed in me.

PUBLICATIONS

1. Jambunathan, K., Lai, E., Hartle, S. L., and Button, B. L., (1991a). Development of an Intelligent Front End for a Computational Fluid Dynamics Package. *Artificial Intelligence in Engineering, Volume 6, Number 1, January 1991, 27-35.*

2. Jambunathan, K., Lai, E., Hartle, S. L., and Button, B. L., (1991b). Development of an Intelligent Front End: An Experience. *Engineering Applications of Artificial Intelligence in Engineering, Volume 4, Number 5, 385-392.*

3. Jambunathan, K., Lai, E., Hartle, S. L., and Button, B. L., (1992). Development of an Intelligent Front End using LISP. *Proceedings of the Seventh International Conference on Artificial Intelligence in Engineering, 14/16 July 1992, University of Waterloo, Ontario, Canada, 229-243.*

4. Hartle, S. L., Li, H., Lai, E., Jambunathan, K., and Button, B. L., 1993. A Knowledge Based Approach to Data File Checking for Numerical Simulation Packages using an Expert System Shell. *Submitted for publication in Engineering Applications of Artificial Intelligence, July 1993.*

5. Hartle, S. L., Jambunathan, K., Lai, E., and Button, B.L. Aspect Ratio dependent finite volume grid generation. *Submitted for publication in the International Journal for Numerical Methods in Engineering, August 1993.*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Chapter 3

AR	Aspect Ratio
h	Height of arbitrary one dimensional region
L	Smallest cell size
m_j	Gradients of the regional profile function, j = 1, 2, 3
N	Number of cells within the region
y	Down stream cell face distance from a datum

Subscripts

i	Cell number

Greek

α	Grid clustering parameter
α_j	Abscissa profile function parameters, j = 1, 2, 3, 4
β	$f(2\pi\alpha_3)$
ϕ	Dependent variable
λ	Cell aspect ratio
θ	$2\pi y$
ψ	Maximum allowed $f(2\pi y_i)$
ψ_N	$(\alpha_2\ h)\ /\ (\pi\ AR\ L)$

ABBREVIATIONS

AI	Artificial Intelligence
AIP	Advanced Information Processing
BEM	Back End Manager
CFD	Computational Fluid Dynamics
CIM	Computer Integrated Manufacture
DoI	Department of trade and Industry
ES	Expert System
ESPRIT	European Strategic Programme of Research and development in Information Technology
FE	Front End
IFE	Intelligent Front End
IKBS	Intelligent Knowledge Based System
IT	Information Technology
KB	Knowledge Base
KBFE	Knowledge Based Front End
KBS	Knowledge Based System
MIS	Management Information Systems
MIT	Massachusetts Institute of Technology
MMI	Man Machine Interfaces
MRP	Materials Requirements Planning
PHOENICS	Parabolic Or Elliptic Numerical Integration Code Series
PIL	PHOENICS Input Language
SERC	Science and Engineering Research Council
SRS	System Rule Syntax
URS	User Rule Syntax
VLSI	Very Large Scale Integration

CHAPTER 1

INTRODUCTION

1.1 Background

Software packages come in many forms ranging from simple teaching programs, through database packages to extremely complex programs that solve problems based on the fundamental laws of physics. Simple teaching packages usually require interactive input from the user which might be a simple yes or no answer to a question. However, complex and versatile programs conventionally rely on auxiliary input files to feed the main executable code with data. These type of systems employ a specific language designed for the package which are idiosyncratic because they contain encoded data. This data is formulated from commands that the software can understand but appears unintelligible to users who are unfamiliar with the command language. This is typical of software developers who try to simplify concepts so much that they become too engrossed to realise that other people do not appreciate the significance of specific commands. Each package usually has its own specific command language used for entering data which describes the problem to be analysed. There usually exists a bottleneck in the use of the software, for novice users, which can be overcome by appropriate training. Progression through the initial stages of the learning curve are prerequisites to becoming proficient with any package. A similar situation is experienced when a new computer language is being learnt. Furthermore, packages of a similar nature exhibit an overlap of concepts, as with computer languages. This allows a proficient user of one particular package, say for stress analysis, to understand the command language, of another stress analysis package, relatively easily when compared with a novice. This is because a problem needs specific data before it can be solved, occasionally in a predetermined order. Knowing what data is required, and when, contributes to the problem of data entry.

Being able to communicate with a computer in English through an interactive session whereby data relating to the execution of a particular package could be entered, would be beneficial. Benefits would include reducing the training required, shortening the learning curve and increasing the number of potential users. In order to allow communication of this nature it would be necessary to insert, between the user and the target package, secondary software that would act as an intermediary. This secondary software, often referred to as the Front End (FE) would take the information given by the user and transform, or synthesise, it into the commands required by the target program.

1.2 Application

The processes of heat/mass transfer, chemical reactions and fluid flow pervade all aspects of human life. These processes can be observed in engineering: combustion engines, aircraft, rockets, heat exchangers, air conditioning plants, the natural environment: pollution, storms, floods, fires and in the human body: blood flow, temperature control via heat and mass transfer. As a consequence of the enormous influence the processes have on human life it is essential to be able to predict the behaviour in order to deal with them effectively. Extensive research throughout the world, over many years, has yielded many powerful numerical simulation packages.

Within the engineering industry the use of numerical simulation packages play an extremely important role in computer aided design. Powerful microcomputers provide relatively small companies access to comprehensive Computational Fluid Dynamics (CFD) packages. CFD modelling of physical situations can be an extremely complex procedure and it usually requires specialist expertise and familiarity with the package to establish a working model. The generation of an input data file to a CFD package can be cumbersome and simple modifications usually require extensive alterations to the format. These modifications can be very susceptible to catastrophic failure due to the enormous potential for human errors in typing or a momentary lack of concentration. This risk increases directly with the size of a data file which is usually large in a realistic problem. The data files contain information relating to the geometry, boundary conditions, properties and the solution parameters associated with the package. In common with other numerical schemes most CFD packages tend to be of a generic nature thus allowing numerous permutations of analyses to be performed. For example a CFD package might be able to consider laminar/turbulent flows, heat/mass transfer and chemical reaction processes. The availability of a number of options for the user to choose increases the number of commands he may have to enter, each of which informs the main executable code to either include or omit a particular option from the analysis, thus limiting the number of variables the program has to solve. Clearly, the marketability of the software package relates to its versatility to model a variety of different classes of problems. Even though the availability of CFD packages is increasing, their popularity and potential market is yet to be fully realised, especially by small companies. This is mainly because of the costs involved in releasing engineers to attend the necessary training courses to become proficient with the package, and the need for these engineers to have at least a

basic understanding of the processes involved in order to get the full benefit from the courses. The time to become familiar with a numerical stress analysis package, MARC, is anything up to one year depending on the ability of the user, Bennet and Englemore (1979). This timescale is typical for most software packages and experience has shown this to be so for PHOENICS (versions 1.4, 1.5 and 1.6).

PHOENICS is a general purpose finite volume package designed for the simulation of fluid flow, heat/mass transfer and chemical reaction processes. It achieves the simulation by solving the associated governing differential equations of fluid motion, heat transfer and the conservation of chemical species, Patankar (1980). PHOENICS solves the governing equations in a discretised form, employing specifically designed solution algorithms which are hidden from the user. Problems are formulated and then described to PHOENICS with the PHOENICS Input Language (PIL). PIL is used for entering data such as the geometry, fluid properties, boundary conditions and solution parameters. PIL can be entered interactively, or into a data file which is read and interpreted by PHOENICS. The latter method is possibly the most commonly used technique, which requires a data file named Q1.DAT. For a beginner, the task of learning how to specify a problem correctly using PIL is a very slow process. Some PIL commands require background CFD knowledge, and as such an understanding of the techniques used to solve CFD problems is usually advisable. Experience has shown that the learning curve for PHOENICS can easily extend beyond twelve months: this would be required to become familiar with the most commonly used commands, and to be made aware of the advanced facilities within PHOENICS. Indeed experienced users still find commands or functions available that they are not aware of.

Front Ends to software packages are usually designed to improve the data entry process through the use of menus, interface screens and help facilities. Techniques such as these are used to provide fixed interfaces, and as such could demand considerable time to modify. Knowledge Based System (KBS) techniques rely on an inference engine to use a set of rules contained within a Knowledge Base (KB) to extract the information from the user and to perform the necessary presentation. The term Intelligent Front Ends (IFEs) is used by some to describe user interfaces employing knowledge based techniques. The term Knowledge Based Front Ends (KBFEs) is also used as an alternative, and is one which is preferred because it describes the techniques used for the development of the front end. However, the terms IFE and KBFE will be used synonymously.

Over the last decade two initiatives were embarked upon for Information Technology (IT) in Britain and Europe: ALVEY and ESPRIT. ALVEY was a British initiative which commenced in 1983 for a five year period. There existed numerous research projects under this initiative, one of which was directed towards IFEs. As a contrast to ALVEY, ESPRIT was the European programme which established international collaboration from industry and academia towards IT research. The infrastructure of ESPRIT was similar to ALVEY in so much as they categorised the research interests. The area of Knowledge Based Front Ends was the counterpart of the ALVEY IFE research theme. ALVEY and ESPRIT are briefly discussed in Chapter 4.

An IFE, as defined by the SERC/DoI in their final report on Intelligent Knowledge Based Systems (IKBS) architectures is as follows,

"[A] front end to [an] existing software package for example a finite element package, or mathematical modelling system, [which] provides a user friendly interface (a "human window") to packages which without it, are too complex and/or technically incomprehensible to be accessible to many potential users. An intelligent front end builds a model of the user's problem through user-oriented dialogue mechanisms based on menus or quasi-natural language, which is then used to generate suitably coded instructions for the package.", Bundy et al. (1984).

The definition was taken by Bundy (1984a) and condensed into a more succinct statement, "An intelligent front end (IFE) is a kind of expert system. It is a user friendly interface to a complex software package which would otherwise be incomprehensible and/or too complex to be accessible to many potential users."

1.3 Aims and Objectives

1.3.1 Aims

To establish a knowledge based methodology to prepare data for complex computer programs. This was to be achieved through the development of a Knowledge Based Front End for a commercial Computational Fluid Dynamics package, PHOENICS.

PHOENICS is such a complex package that the inclusion of all commands within the KBFE and their interrelationships would not be feasible under this project. In order to demonstrate the KBFE concepts, and how they should be implemented, a knowledge domain was chosen which considered two-dimensional jet impingement.

Using this knowledge domain, the initial aim of the research was ...

- To assess, through the development of a prototype front end, how an expert system shell would perform with respect to knowledge representation, data storage and inferencing.

This, as will be shown, proved to be problematic, in so much as the chosen shell was inadequate for this application in terms of knowledge representation and data storage facilities. However, a useful introduction into the techniques of knowledge based systems and the terminologies used was provided through the development of the prototype. The experience gained using the shell contributed in the decision to move towards the use of a traditional Artificial Intelligence language, LISP. This then lead to a modified research aim consisting of ...

- Using LISP, develop specialised knowledge representation formalisms, inference techniques and a method of storing information for CFD purposes.

1.3.2 KBFE System Objectives

The objectives were to consider the requirements of CFD, and how the process of formulating a problem and heuristically entering / controlling the analysis is performed. To this end the following was addressed ...

- Establish methods of storing PHOENICS variables and data relating to boundary conditions.

- Generate a rule based language for use within the knowledge bases. Achieve knowledge categorisation through the use of multiple rule bases.

- Implement forward and backward chaining techniques on the rule bases through the use of inference networks.

- Establish and implement techniques to recognise when the KBFE needs data entry from the user.

- Devise data synthesis rules to generate coded instructions for PHOENICS.

1.3.3 CFD Heuristic Objectives

- Establish techniques to mimic the heuristic processes performed manually by users of PHOENICS for grid generation.

- Carry out a feasibility study into controlling the PHOENICS solution algorithm, by monitoring the residuals and field values to sense when relaxation factor modification is required.

- Investigate grid optimisation post processing procedures that have the possibility, through incorporating code, to aid the analysis of results.

1.4 Chapter contents

A literature review is provided in chapter 2 which examines previous work in the field of Artificial Intelligence (AI) and its application to CFD. This is preceded by a description of what an expert system is, why they are used, and introduces criteria to be employed in assessing the need for an expert system. An insight into Intelligent Front Ends (IFEs) is given, which is a consequence of the Alvey project, and its ESPRIT counterpart Knowledge Based Front Ends (KBFEs). There is very little difference between an IFE and a KBFE, essentially in architecture structure, however the term KBFE is preferred but both terms will be used synonymously. Application areas of KBFEs, and their associated research, are presented which illustrates the wide use of such systems. Areas covered include, but are not restricted to, program generation, databases, statistical packages, and general engineering. General engineering encompasses KBFEs for finite element stress analysis packages, mesh generation, building design and building energy simulation packages. Finally, the interaction of AI and CFD is reviewed.

Due to historical reasons, PHOENICS was chosen as the package with which to interface the developed KBFE. PHOENICS is a Computational Fluid Dynamics package, and as such chapter 3 concentrates on the architecture of PHOENICS and introduces the various data entry techniques. This is preceded by briefly mentioning the basics of CFD with respect to the discretisation and the governing differential equations that are solved in order to numerically predict fluid flow. An example geometry and data file is given for unconfined jet impingement in order to illustrate the PHOENICS Input Language. The reasoning behind the integration of AI and a commercial CFD package is addressed which is then extended to cover the type of knowledge required in order to formulate and specify a problem using PHOENICS terminology. PHOENICS uses the finite volume integration technique, and as such requires that the defined mesh for a specific geometry adheres to certain guidelines in order to ensure representative flow results. One parameter that must be considered is the cell aspect ratio when refining a mesh near a wall, in order to capture the viscosity effects of near wall flows. A technique for heuristic finite volume mesh generation is presented that utilises a region height, minimum cell size and a maximum allowed aspect ratio. These parameters, when used with the developed equations, create a smoothly varying mesh which does not exceed the maximum allowable aspect ratio.

A closer look at the historical background of Intelligent Front Ends and Knowledge Based Front Ends is presented in chapter 4, through the résumé of the ALVEY and ESPRIT projects. A schematic representation is given which indicates the locality and role of a KBFE. This is then expanded onto a more detailed scale whereby the architecture and individual components of a KBFE are described. Expert system shells are mentioned with respect to their role in the development of KBFEs.

A prototype KBFE was developed using the expert system shell LEONARDO, versions 3.17, 3.18 and 3.20. This was the first approach to be taken in the research, as illustrated by the initial aim of the project. The LEONARDO KBFE is described in chapter 5, whereby the architecture is presented, and problems that were encountered are highlighted. It was these problems which prevented a useful system being developed and contributed to the decision to modify the approach to use LISP. The prototype consisted of two independent facilities: a data file checker, Hartle et al. (1993), and a data file generator. Possibly the most important deficiency was LEONARDO's use of pseudo-lists, these are described and compared with doubly linked lists. The need for mathematical parsing within KBFEs is presented.

LISP was used to develop specialised knowledge representation formalisms, inference techniques and a method of storing information for CFD purposes. This was a consequence of the experience gained with LEONARDO, hence the modified research aim. Chapter 6 presents the architecture of the KBFE, and describes the various specific LISP functions. A brief discussion on symbolic manipulation precedes a detailed examination of the KBFEs database and data storage through the use of LISP structures. Assertions, usually referred to as facts, are used to compliment the data storage techniques. A rulebase language is presented which has been developed specifically for the project and incorporates traditional **IF ... THEN** and **list quantification rules**. A detailed description of the inferencing techniques is given, and inference networks are examined with respect to the implementation of forward and backward chaining.

Solution monitoring and control, along with adaptive grid optimisation, are tentatively examined in chapter 7. The feasibility of incorporating the heuristic monitoring and control of the PHOENICS solution algorithm is addressed through the partial implementation of the technique. Adaptive grid optimisation is very briefly looked at through the consideration of a grid iteration technique which endeavours to optimise the convergence onto grid independent solutions.

The current study is concluded in chapter 8 whereby the observations and experiences acquired through the research are presented. The pertinent areas of work are highlighted in order to consolidate each of the chapter contents. Extensions of the project are suggested in the recommendations for further work which includes the investigation of the use of Neural Networks to aid in grid generation.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Expert systems have been given an enormous amount of attention during the last decade, and were initially the first commercial by-product from the research directed towards Artificial Intelligence (AI). More specifically, the research that is of current interest is related to the application of expert systems, either using the classical languages such as PROLOG or LISP, or through the implementation of any one of the numerous expert system shells available. For rapid prototyping and a quick introduction to the techniques employed by expert systems, implementation of shells are recommended due to the ease with which one can become familiar with the specific semantics and syntax that they employ. However, at the early stages of developing a system one must not totally isolate the possibility of using a different tool or even a development language. The reasoning behind this will become clearer in due course. The use of AI techniques, related to integrating systems with other software, has been a constant area of research.

The main thrust of an expert system is to reduce the load placed upon an expert, in a highly specialised area, by removing the necessity for him to perform relatively mundane and tedious tasks. To this end, a system should be there purely as a slave and not as an "expert". This is manifest since the system will only perform the tasks that it has been "told" to, through the use of encoded rules, and it will not be able to apply new information until the appropriate modification of the knowledge base has been completed.

The necessity for an expert system can be questioned by the sceptics who believe that a computer can never replace a human. This can never be argued because all computer programs, no matter how extensive the validation process, invariably contain hidden "bugs". Their existence is only to aid the expert and to try and disseminate relatively low level expertise. Criteria for assessing the need for an expert system can be based upon the time required for an "expert" to perform a specific task. Kathawala et al. (1989) reports on a checklist devised at the Massachusetts Institute of Technology (MIT) which establishes whether a field is worthy of integrating an expert system. These conditions are:

1. There must be an expert in the field;
2. The task must require a few minutes to a few hours to complete, anything less

should be done manually;

3. The task must be of the sort typically taught to novices;
4. There must be a high payoff, in terms of money saved;
5. No common-sense must be required.

Furthermore, Morley and Taylor (1986) suggest, under point 2 above, an expert system should not be used if it takes less than ten minutes to solve a problem. Under these circumstances Aseo (1988) claims that such an application is not cost effective, and any implementation exceeding a few hours is beyond the state of the art.

In this chapter, areas within KBFE design and implementation will be discussed. Applications will be mentioned with particular attention to CFD and its associated facets. The research undertaken by others in the field of KBFE applications will be reviewed and pertinent points drawn from their publications. An assessment of general trends in hardware and software environments will accompany information related to the techniques used.

2.2 Intelligent front ends

The definition of an IFE, Bundy et al. (1984) and Bundy (1984a), indicates that the implementation of developed systems are directed at improving the usability of existing software packages which are technically incomprehensible and/or too complex, for the novice user, to use. All packages that have IFEs developed for them have one feature in common; they all have enormous potential for use. This is provided that the initial learning obstacles can be overcome, which may take anything up to one year, Bennett and Englemore (1979). The type of package that IFEs can service is virtually limitless.

Edmonds and McDaid (1990) report on an architecture for Knowledge-Based Front Ends (KBFEs) which originated from the FOCUS project (ESPRIT2 project 2620: Front Ends for Open and Closed User Systems). FOCUS is a collaborative project between industrial and academic pa[illegible]G, Imperial College, LUTCHI, METEK, Phillips, Solvay, University of Barcelona and the University of Muenster, and required fifty six man years of work over a four year period, Brouwer-Janse (1990). The project sees KBFEs as enhancing the usability of target software packages by providing improved interfaces and knowledge-based support to the user. The main thrust of the project was to develop

generic tools for constructing and maintaining KBFEs. The KBFE concept is similar to that proposed through the ALVEY project, Bundy et al. (1984), but isolates a unit within the KBFE that separates the user, application package and knowledge-bases. This unit, called the Harness, effectively contains the inference engine along with data presentation routines. The Harness controls the dialogue and presentation of data and information with respect to the formulation of a problem. The synthesis of data to commands required by the application package is controlled by the Back End Manager (BEM), Edmonds and McDaid (1990) and Prat et al. (1990). FOCUS allows similar application packages to be utilised from the same KBFE through the generation of separate data synthesis knowledge bases.

2.3 KBFE applications

There exists many application areas for KBFE development. A summary of some areas will be given with a sample of some relevant publications. The areas to be considered includes: Management Information Systems (MIS), Program generation, Databases, Process plant, Simulation, Statistical packages, general engineering and CFD.

2.3.1 Management information systems

Drechsler and Peppard, (1988) describes the application of IFE technology to Management Information Systems (MIS) and Computer Integrated Manufacturing systems (CIM). The work was carried out under the ESPRIT project 319 relating to data transfer between CIM and MIS sub-systems. The expert system, LSM (Lot-Sizing Module), was designed to supply an answer to the lot-sizing problem in the context of Material Requirements Planning (MRP). Given relevant data, the system chooses between available lot-sizing techniques in order to provide data to the information system package. LSM was developed using the ES/P ADVISOR expert system shell.

Kurstedt et al. (1988) discuss the development of a responsive system, MLSTRAIN, as an IFE for computer-based management application programs. A conceptual background to the responsive system is provided by the Management System Model (MSM), which details the relationships between the manager, what is managed and the tools used to perform the management. MLSTRAIN is designed to aid managers plan a training schedule for their workers in order to become computer literate, through professional development.

2.3.2 Program generation

Barstow et al. (1982) report on the development of an automatic programming system ΦNIX, which was written using INTERLISP-D and runs on a Xerox 1100 processor. The system was created for petroleum scientists who may not be knowledgeable about computers but would benefit from the generation of computer models, using natural concepts, to quantitatively interpret well-logs. The resulting model specifications can be implemented in any of several different target languages, such as LISP or FORTRAN.

Engquist and Smedsaas (1980) describe the development of a system which generates FORTRAN code for the numerical solution of systems of hyperbolic and parabolic differential equations, DCG (Differential equation and Code Generator). The user describes the system he wishes to model by interacting with the computer using a Problem Description Language (PDL). DCG is divided into two separate systems: the analyser, written in SIMULA, and the synthesiser, written in FORTRAN, which utilises a code library. The analyser handles the user communication and symbolically processes the problem through syntax and semantic analysis. The analyser generates an output file containing instructions for the synthesiser in the form of patches of code to be used in the final FORTRAN program, and flags indicating the characteristics of the problem. The synthesiser utilises the information within the file and generates a program from a "preprogram" in a code library according to the related instructions.

Fang et al. (1988) present an IFE, AUTOP-GPSS/C (AUTO Programming in GPSS/C), that is used to aid the generation of GPSS-C programs for Discrete-Event Simulation Systems (DESS). Turbo-PROLOG was used to develop the IFE on an IBM-PC/XT(AT). The system incorporates three fundamental components: the man-machine interface, module selection and program generation. The machine interface utilises natural language dialogue controlled by rules contained within the rulebase. Every question asked relates to certain DESS concepts and the information gathered is stored in a global database called the blackboard. Module selection is performed via an inferencing process on the knowledge base, containing more than one hundred rules, and utilises the data and information stored on the blackboard. The system incorporates forward and backward chaining. Ninety of the rules contained within the rulebase, when used with backward chaining, create the resulting GPSS/C code.

Uschold et al. (1984) report on the development of an IFE for Ecological Modelling, ECO, which aids an ecologist user build a customised FORTRAN program to simulate the processes in an ecosystem. The user creates a system dynamics model through interacting with the IFE using free-form dialogue. The system knowledge base contains information relating to ecological modelling data and is subdivided into three modules: Module library, Entities and the Process library. The Module library contains approximately forty modules which are mathematical functions, each with an associated ecological context. Entities are objects that can be linked together with processes by the user and IFE to create a system dynamics model. The Process library includes all of the ecological processes that the user can incorporate into the model. The dialogue subsystem essentially consists of six command statements, such that when combined with objects and processes produces an English like sentence that is interpreted by the IFE to create the model. The program generation is created with the aid of a well defined program template.

2.3.3 Databases

Banwell (1989) describes the empirical knowledge elicitation approach taken for the development of an IFE to a library database of information. The emphasis of the initial research was to create a methodology whereby an individual user model would be built up interactively during dialogue sessions. This would then be coded into the working system. A stereotyping paradigm is advocated which utilises a model that should be upgraded to suit the particular user. This model could then be recalled when the user re-starts a new interactive session.

Cornali (1990) presents four adaptive strategies for KBFEs incorporated in a prototype system, TELOS, an adaptive front-end for a NAG FORTRAN library, and which is encompassed by the FOCUS initiative. These strategies are (1) User Stereotyping, (2) Adaptive Terminology, (3) Objective Qualifications, and (4) Context-Sensitive Examples. User stereotyping is considered to be valuable for adaptive front ends provided that (a) the attributes predict the user's ability, (b) stereotyping is only a first approximation, and (c) attributes are upgraded as the user progresses. An important feature of an IFE is its ability to adapt the required terminology to suit the user. Objective hierarchies are used within TELOS to ascertain the needs of the user through a question and answer session. This leads to the generation of a goal which is used in the search for the best method to

accomplish the request. Context sensitivity relates to the ability of the system to provide explanations in the context of the current user's situation. This removes a level of abstraction from the user and allows him to relate to the explanation

Ford et al. (1989) address the problems experienced by users of the NAG (Numerical Algorithms Group) numerical and statistical software libraries. Such problems include having to learn new programming languages and being able to find their way through reference manuals containing thousands of pages in order to successfully implement a chosen library routine. The paper describes the role NAG have adopted with respect to the KBFE projects they have undertaken. The KBFEs are NAXPERT, KASTLE, FOCUS, GLIMPSE and SISP. NAXPERT is a decision support system written in PROLOG to aid the selection of numerical routines from a FORTRAN numerical library. KASTLE (Knowledge Assisted Selection Tool for Library Environments) is a KBFE for the NAG FORTRAN library written in PROLOG, Whitmore (1991). NAG was the coordinating partner of the ESPRIT2 FOCUS project which has been previously discussed. GLIMPSE utilises PROLOG as its symbolic language to interface a FORTRAN statistical package, GLIM. Wolstenholme and Nelder (1986) report in detail the front end for GLIM. SISP, reported by Reid (1990), is a front-end which operates on a PC to provide the user with an interface to systems with which he wishes to interact. The systems can be operating systems or software packages. SISP allows a non-computer literate to communicate and use a variety of facilities contained on a PC.

Khabaza et al. (1988) suggest that it is extremely difficult to provide an IFE to a complex database, such as an online 'help' system for sophisticated software. The system is an intelligent help-file-finder (HFF) for the POPLOG programming environment. To qualify the difficulties, various factors that make finding online information troublesome are presented. To alleviate the problem it was proposed to utilise the human brain as an inference engine, due to its inherent power, thereby allowing the user to take decisions and make inferences. A pre-requisite to this is that the database should be very well organised and include considerable quantities of meta-information, i.e. information about the contained information. The conclusions drawn from the work suggested that the task of designing a truly intelligent HFF was beyond the state-of-the-art, but progress was made by combining human intelligence with a well designed database.

Mao (1988) describes a model of a knowledge oriented human-computer interface that takes an expert system as an IFE to a database package written in Turbo-PROLOG. Mao expresses an opinion that the IFE should ideally transfer control of the dialogue back and forth from the user to the system. This is because a menu system, which is computer controlled, restricts the user and convicts him to progress through verbose questioning. Whereas a user controlled dialogue session is reported to place the system requirement so high, the need for natural language processing, that the generality of the user interface design is lost. The interface model or IFE consists of five modules and a knowledge-base. The Menu Module provides multiple windows whereby data entry can take place and forms the user interface. The Input Analysing Module divides an English sentence into nouns, verbs, relatives, determinatives and relevant phrases. Classification of the words into a tree structure precedes keyword matching with the knowledge base. The Clustering Module sorts and merges the result produced by the Input Analysing Module. The Learning Module consists of knowledge acquisition and allows the renewing and modification of the knowledge base. The Explaining Module describes the system infrastructure to the user and explains, for example, the knowledge structure in the knowledge base and the schema of the database.

Tou et al. (1982) developed RABBIT which is an intelligent database assistant that aids the user to formulate a query. The retrieval paradigm is based on a psychological theory of human remembering: retrieval by reformulation. The system has been developed specifically for those users that are either casual users or who approach the database with only a vague idea of what they want. The latter rely on the system to guide them through a (re)formulation of their queries.

2.3.4 Process plant

Emmett (1987) discusses real time data acquisition from multiple plant sources, including operator input from the console and signals from various transducers and instruments. The associated problems, which include cable costs, plant noise and communications, are discussed and appropriate solutions presented. Communications noise, in various forms, is addressed through I/O processors. The IFE, as it is referred to, consists of a I/O processor card fitted in the plant interface unit. This simulates parallel processing, through each unit receiving and processing data before passing it to the controlling processor for use with data from other interfaces which is captured in a similar manner.

PICON, Process Intelligent CONtrol, is an IFE designed to serve an existing distributed control system. Moore (1985), describes PICON's application to assisting operators in dealing with the numerous alarms that can result from an interruption in the process industry. PICON does not control any of the distributed system, but only diagnoses the alarm systems and advises operators how to deal with them. The system was designed to operate in the lambda/PLUS LISP machine, a parallel processing computer with a dedicated LISP processor for symbolic processing tasks, and an MC68010 processor for fast data acquisition.

2.3.5 Simulation

Elmaghraby and Jagannathan (1985) describe the design and implementation of an expert system to assist simulationists in selecting a simulation language matched to their model and computer resources. The expert system and relational database have been built using a dialect of LISP (IQLISP) on an IBM personal computer. The rules are primarily conjunctive with each clause being made up of disjunctive sub-clauses. A backward chaining inference engine utilises the data contained within the database whilst trying to fire the rules.

Guariso et al. (1989) describes the conceptual design and a prototype implementation of a knowledge based interactive generator of simulation models. The system comprises three major components: model base; knowledge base; and the data base. The model base contains the numeric models in the form of executable simulation programs, written in languages such as FORTRAN, which run on a 8088 processor (MS-DOS). Data for each model is provided and if several models are linked together then supplementary information needs to be provided. Management of the model base is performed by the system manager and the knowledge base. Knowledge relating to how models can be connected is coded into the knowledge base using INTERLISP-D. The database contains the necessary data for correctly executing different component models using a frame based system. The system manager consists of three components: a dialogue component; a control unit; and a simulator. The dialogue component forms the user interface, whereas the control unit performs various tasks including the maintenance of the different bases and model consistency checking. The simulator monitors the execution of the simulation models, calls the numeric programs from the model base and provides the necessary data from the database.

Michelsen et al. (1988) describe the development of SEAT (Strategic Engagement Analysis Tool) written in Common Windows, Common LISP and KEE (Knowledge Engineering Environment). Most weapons simulations or battle simulations are historically written in FORTRAN. SEAT consists of two primary components: a generic IFE and time-driven simulation capabilities. The IFE enables the user to specify scenario values for different objects within the simulation, and it ensures that the values are both meaningful and consistent with one another. The development languages for the IFE were the Symbolics Window system and Common LISP. KEE was used to create the simulation environment that contained the knowledge bases and the supporting graphics routines. Seven knowledge bases existed within the prototype that contained information relating to the objects (actors) and the rules for each subsystem.

O'Keefe (1986) addresses the similarity between expert systems and simulation. Furthermore, it is suggested that one of the most important application areas for knowledge based methods is IFEs. Dialogue handling, user models and a model of the target package are identified as useful intelligence attributes to be included in potential IFEs. It is also mentioned that allowing a system to obtain and analyse the results of a simulation and appropriately altering the supplied data would be beneficial. If a KBS could be produced to perform such tasks easily then "the resulting system would be a very sophisticated IFE".

Strandhagen (1989) gives an overview of the different uses of expert systems in manufacturing simulation, including simulation and expert systems as a teacher, expert systems for scheduling, and expert systems as analysts of simulation results. The main goal of the SIMMEK research programme was to supply Norwegian industry with a simulation tool that can be used at all levels of Production Management. SIMMEK is an object oriented package that requires no programming experience to use the system and the input data includes economical factors. The three main modules of SIMMEK consist of: the Modeller, the Simulation Kernel and the Analyst. The Modeller is the IFE, and essentially creates a model by joining together objects within a window environment. Two sub-models constitute a complete model: the layout model and the product flow model. A model, stored in the database, is checked for consistency and then transformed to program code readable to the simulation kernel.

Tangen and Wretling (1986) discuss the application of Intelligent Front Ends (IFEs) to numerical simulation programs. The paper attempts to generalise the functional and knowledge aspects of IFEs. The proposed functional aspects include the need for a graphical interface, flexible presentation of simulation results and easily extendable knowledge-bases. The flexible presentation of simulation results can be omitted if the numerical simulation programs are provided with adequate post-processing facilities. However, advice on analysing the results should be included within the system. Knowledge aspects of IFEs presented give a flavour of the information required to establish a working IFE, such as production rules, templates, parameter constraints and typical parameter values. An IFE architecture is presented that relates to the described system KIPS (Knowledge Interface to Process Simulation) which interfaces a process plant simulation program. The paper, although addressing IFE requirements, only focuses on simulation programs that rely on the user building models by interlinking objects from a library of predefined components. This approach is slightly different to that required from CFD in so much as the latter does not use libraries of predefined components to create a model.

Xuesi and Zhengzhong (1988) describe the system architecture, contained knowledge, control strategies, and explanation facilities of the Simulation Integration Algorithms Selecting Expert System (SIASES). SIASES was written using the Simulation Algorithms Expert System development Tool (SASEST), using Turbo-PROLOG, and forms the basis of an IFE to the simulation software SL1. SIASES provides knowledge on mathematical properties, advice on algorithm selection, and assistance on interpreting mathematical expressions. The knowledge encoded into SIASES consists of declarative knowledge, process knowledge and control knowledge which is represented using ten forms of selected predicate symbol expressions. Control is performed through forward chaining on the rules.

2.3.6 Statistical packages

Wittkowski (1991) [illegible] PANOS, a human-computer interface to a statistical package. PANOS is described as a front-end graphical interface to statistical database management and analysis systems, for biomedical research applications, written in TURBO-PASCAL 5.5 for MS-DOS. Data is entered into the system through screen forms using a specifically designed problem formulation language.

Wolstenholme and Nelder (1986) describes the progress made in the development of a knowledge based front end for a statistics package, GLIM. GLIM was designed for the analysis of generalised linear models (GLMs). The system runs on a VAX 11/750 and employs logic programming through the use of sigma-PROLOG and its associated programming environment APES (Augmented Prolog for Expert Systems). The main features of APES highlighted as being useful for KBFE development were declarative dialogue handling and inherent explanation facilities. Five requirements were formulated for the system, these included (a) advice should be given to the user, based on broad principles, indicating what actions are available; (b) the system should explain any advice given; (c) several user levels; and (d) clarification of questions or terms used. The conceptual structure of the system comprises three facets: an abstract statistician; a translator; and the statistics package (GLIM). The abstract statistician enables basic computations to be performed and can store and display data graphically or in tabular form. This allows common features between similar packages to be performed, using high level language possibly incorporating natural language understanding, but remains independent of the target software. The translator synthesises the information into GLIM commands. Depending on the rules used within the back-end manager, multiple packages could be serviced, in this case GLIM.

2.3.7 General engineering

The area of general engineering has been so classified to encompass areas such as building design, solid mechanics, control system design, and dynamical system analysis, among many others.

Ambroziak and Kleiber (1990, 1991) discuss a blackboard consultation system for constitutive modelling in solid mechanics. The system, CONMOD, utilises a central communications facility for data storage. Even though the system is not exactly an IFE it advises the user on what type of equation set should be employed to model a solid mechanics problem. Furthermore, it is concluded that the work presented would be important to those interested in materials data bases and front-ends to numeric packages.

Bennett and Englemore (1979) utilised EMYCIN, an extension of MYCIN, Shortliffe (1975), to develop an IFE, SACON (Structural Analysis CONsultant), which advises engineers in the use of a finite element structural analysis package, MARC. At no time

during the development of SACON did the authors find the representation formalism (rule-based) of EMYCIN to be a hinderance relative to the formulation of knowledge or its eventual implementation. Furthermore, for an engineering application it was found that the inherent confidence factor mechanism was not implemented. This concluded that the omission of uncertainties within the generation of data and information was not required. Wager (1984) discusses the architecture of SACON in greater detail concentrating areas such as the analysis strategy, knowledge base and the knowledge manager.

Building design and building energy simulations have been performed using existing Computer Aided Building Design (CABD) packages as discussed by Clarke (1990). MacRandal (1987), Clarke et al. (1988), and Clarke and MacRandal (1991) describe the form and content of an IFE for building design incorporating building energy simulation. Clarke (1990) presents the evolution of energy models and indicates that the fourth generation which includes Intelligent Knowledge Based Systems (IKBSs), the IFE, will emerge in the early to late 1990s. The architecture of the IFE includes various co-operating software modules that includes human-computer interaction, data entry validation and intelligent defaulting. The software modules are organised around a central communications module, the 'blackboard'. This structure enables each software module to access and write information onto the blackboard, thus sharing relevant data. The 'blackboard' architecture is a problem-solving framework and was developed for the HEARSAY-II speech-understanding system, Erman et al. (1980), and has been used as an architecture for control, Hayes-Roth (1984, 1985). Reddy and O'Hare (1991) outlined the three main components of the model: the blackboard attributes, knowledge source attributes, and the general system attributes. These attributes are described and are used to survey, as then, current applications.

Fink et al. (1987) describes ATHENA AIDE, an expert system used for the preparation of input models for the ATHENA (Advanced Hydraulics Energy Network Analyser) thermal hydraulics package. It is a menu driven graphics interface utilising rule-based and object-oriented programming techniques and is executed on single-user Xerox AI work-stations. ATHENA models physical systems using objects such as pipes, pumps and valves from libraries similar to the process plant simulation codes described by Tangen and Wretling (1986). Knowledge representation is performed using an object-oriented language in conjunction with Interlisp-D. Two user models are introduced through the automatic and

passive modes of operating ATHENA AIDE. Automatic operation is intended for a novice whereby the system guides the user through the data entry sequences, whereas passive operation is intended for experienced system users who know when and what data to enter.

Pang (1988) has taken the definition of an IFE given by Bundy (1984a) and has developed an IFE for a control system design and analysis package SFPACK, which is similar to MATLAB. The IFE is essentially an enhanced command interpreter with several fixed user models to introduce system flexibility. The enhancement is produced by allowing system adaptability to various users, guidance for the use of the application package, command recoverability for errors in user input and to act as a tutoring tool. A multi-level approach has been adopted to introduce three user levels: Expert level (IFE as a caretaker); Intermediate level (IFE as an assistant); Novice level (IFE as a tutor). The IFE was developed using C as opposed to using a traditional AI language or an expert system shell.

Ramirez and Belytschko (1989) describe an IFE, ETUDES (Expert Time integration control Using Deep and Surface Knowledge System), which is used for setting time steps in dynamic finite element programs written in FORTRAN. It was concluded that the only suitable knowledge representation techniques to be used consisted of production rules and frames. Furthermore, it was necessary to represent deep knowledge using mathematical models. In order to facilitate these requirements and to correctly balance flexibility and control, the system was developed using OPS5, a production rule system. The fundamental inferencing process within OPS5 is forward-chaining, although backward-chaining can be emulated.

Thomas et al. (1990) gives a brief description of an expert system interface (IFE) to a suite of rotor dynamics programs. A more comprehensive report is provided by Thomas et al. (1988). The overview states the requirement of the IFE to be (1) the generation of an input data file for rotor dynamic simulation, (2) to control the execution of the suite of FORTRAN programs, (3) extract the required data from the simulation output, and (4) iteratively generate new test cases until a solution has been found. The ES interface was developed using POP11 which resides in POPLOG and implements a forward chaining inference technique on a knowledge base that contains approximately thirty large rules. Exposing the system to engineers has resulted, in their opinion, that the system produces

the same, or better, results as they would by hand. The time taken to obtain a result makes the system attractive, half a days work can be performed in approximately ten minutes.

Weiss et al. (1982) developed an expert system, ELAS, which was integrated into existing software of the petro-chemical company Amoco for monitoring, controlling and interpreting the results from a well-log analysis program, INLAN written in FORTRAN. ELAS is a production rule advice model using the EXPERT production rule system, which is also written in FORTRAN, Weiss and Kulikowski (1979).

Wong et al. (1988) addresses the coupling of an expert system written in Common LISP and several ancillary engineering programs, written in FORTRAN. The expert system is a Seismic Risk Advisor (SRA) used to evaluate the seismic risk of a particular building according to data and expertise expressed in rules. Fuzzy sets are used to represent uncertainties associated with the data and hence with the overall risk assessment. The rules encoded into the system have a complex LISP structure which is difficult to decipher without explanation. Data transfer between the FORTRAN and LISP codes is performed by writing and reading the necessary information onto/from disk, this is the same approach taken by Tong (1985).

Yu et al. (1988) proposed an IFE using rule-based techniques for Intelligent Computer-Aided Control System Design (ICACSD) using a recently developed software system called CADCSC. The expertise that was captured for the IFE consisted of problem-describing, control-theory, problem-solving, algorithms, and executable knowledge. The primary goals highlighted for the ICACSD included the development of knowledge representation techniques for control systems design, and the development of a knowledge base and data base for use with CADCSC. The architecture for the ICACSD is described and attention is directed towards the main functions of the system: Supervisory Expert; Modelling Expert; Analysis Expert; Design Expert; and the Simulation Expert. The preceding "Experts" are individual units within the system each of which contains a knowledge base, data base, inference engine and a user interface. Standard production rules are used in the rulebases and both forward and backward chaining are implemented.

2.4 Integrating artificial intelligence and computational fluid dynamics

Abbott et al. (1988) performed a feasibility study for using a knowledge-based system to aid the user of a sophisticated CFD program, FLUENT. The work performed has been shown to be representative of the experiences throughout the familiarity with PHOENICS. Experiments were performed with novices and a CFD expert to assess their performance when presented with a problem. Three general conclusions were drawn from the study: (1) Sufficient expertise exists for the effective use of CFD code with respect to the accessibility, efficiency and quality of results; (2) Categorisation of CFD expertise as physical, numerical and technical; (3) Automated assistance via AI techniques is possible. CFD problem solving procedure involves setting up the problem, solving the problem and analysing the results. CFD expertise categorisation is discussed and presented in a tabular format. Important aspects that agree with the findings of Abbot et al. are incorporated in this project.

Uzel et al. (1988) performed a feasibility study for using Intelligent Knowledge Based Systems (IKBSs) in Computational Fluid Mechanics (CFM). Their study concentrated on the development of an IKBS for PHOENICS whereby the CFD preprocessor accessed the inference engine and knowledge bases. The method of performing this integration of the IKBS into PHOENICS appeared to require the embedding of the chosen shell, EX-TRAN 7, into the CFD package. EX-TRAN 7 is written in FORTRAN 77. The attributes for using a shell approach were hypothesised and they indicated that they could only speculate on the facilities that should be obtainable. It is claimed that the system developed with EX-TRAN 7 should be able to link the development environment into itself and create an executable IKBS program.

Mehta and Kutler (1984) and Mehta (1986) are variations of the same publication detailing the integration AI with aerodynamics with their idealised system AERODYNAMICIST. When introducing expert systems they take on board the fundamental definitions and reiterate the versatility of expert systems in symbolic processing as opposed to purely numeric processing. Furthermore, they suggest that having expert systems within a company shifts the normal distribution of expertise to a situation whereby the distribution becomes heavily biased towards a higher level of expertise. Seven levels of expert system development are introduced which includes conception, feasibility demonstration, prototype construction, extended use in a prototype

environment and the acceptance of the performance of the prototype system. The final two stages relate to commercially extending and releasing a system.

The characteristics of expert systems differ with respect to the characteristics of conventional programs, Waterman (1986). Mehta (1986) indicates that some of the features such as programming tools, programming style, program architecture and the type of data allow expert systems to grow progressively, and as a result are easy to modify. This is in contrast to conventional programs that grow by revision, and are difficult to modify. A hierarchy of solution methods is presented which establishes a relationship of program power against generality for various programming techniques. Expert systems that possess both symbolic and algorithmic procedures are reported to have a greater generality than their purely algorithmic counter-parts.

Mehta and Kutler (1984) state that CFD involves ten areas where expertise should reside. For the development of an expert system to work with an existing CFD package then some of these facets can be omitted. This reduction leaves the following facets for consideration:

1. Problem definition and input;
2. Selection of necessary available turbulence models;
3. Grid generation;
4. Assignment of boundary conditions;
5. Assignment of initial conditions;
6. Assessment of the resulting flow fields;
7. Presentation of results.

Kutler et al. (1985) state that the most likely area for developing an expert system for CFD application would be within the field of grid generation. This implies that the designed system should purely act as a pre-processor and not as a post-processor. Clearly, if the development of an expert system purely for grid generation is to be attempted, then the ultimate goal would be to produce a system that could not only establish an initial mesh but also adaptively refine it to optimise the grid in order to obtain grid independent results.

Vogel (1989) has developed EZGrid which is a knowledge-based system for automated flow field zoning and mesh generation for CFD. The zoning procedure produces zonal regions whose individual meshes are discontinuous at the interface boundaries, this kind of grid generation procedure is not ideal for finite volume formulation. EZGrid was developed using three programming languages: C, Franz Lisp (a dialect of Common Lisp) and MRS (Meta-level Representation System). The application area lies within the field of grid generation around aerodynamic bodies.

Blacker et al. (1988a) and Blacker et al. (1988b) describe AMEKS (Automated MEshing Knowledge System) which is a two dimensional automated quadrilateral mesh generating system which uses a knowledge based approach. AMEKS was designed to create a meshed domain for finite element modelling. The system architecture and procedures that AMEKS implements for the decomposition of complex shapes are described. FASTQ, a parametric-mapping mesh generator, is used to develop the resulting grid. Heuristics used to produce the final mesh are presented, such as vertex and regional classifications. A comparison of complex mesh generations with those created by analysts shows that AMEKS can perform to the standard of human counterparts.

Dannenhoffer and Baron (1987) discuss the development of a tightly coupled hybrid expert system for complex, local compressible flow analysis using CFD. The system, MITOSIS, combines expert system control through the use of traditional inferencing techniques with a rulebase and the inherent computational power of conventional programming. An important concept is that all procedures share data contained within a central data pool. The implication is that as long as the input and output structure for each procedure are fixed, solution algorithms can be developed and tested independently from other procedures. Rules within the rulebases essentially form the control of the system through inferencing and are used to execute the required procedures.

Knight and Petridis (1992), discuss the experience gained in the design and implementation of an experimental CFD software package incorporating an intelligent knowledge-based component, FLOWES. FLOWES was constructed with the CFD source code available within which to directly integrate the IKBS component. Clearly, this is advantageous because the system could interact with the numerical code. The example described illustrates a two dimensional heat transfer problem.

Tong (1985) introduces the concept of coupling AI/ES and CFD techniques for the design of aerodynamic bodies. The system architecture that is presented distinguishes the AI and CFD modules as being separate entities within the Expert Design System (EDS). The AI module essentially acts as a front end to the CFD programs and data transfer is performed through a system message utility (file passing). The front end is written in Franz LISP, implements the production rule system OPS5 and interfaces the FORTRAN analysis code. The AI modules are put in hibernation while the analysis code is being executed.

2.5 Conclusions

The generation of Knowledge Based Front Ends has increased significantly over the last fifteen years. The various application areas covers management information systems, program generation, database and statistical package interfaces, process plant intelligent control, and simulation packages. The general field of KBFE research covers many aspects from fundamental inference techniques, through data storage to the design and implementation of different user models. The concept of a KBFE is to increase the accessibility of various packages or programming languages to potential users. The progress in the development of universal KBFE concepts is slow, but it is being addressed through various ESPRIT projects. The increased availability of expert system shells gave impetus in the number of developed systems. However, a comparable number of systems have also been developed using traditional AI languages such as LISP or PROLOG, in various dialects. Improved computing power, and the increased number of packages has stimulated the research into KBFEs.

It is evident from the reviewed publications that there is no one preferred method of developing KBFEs: either with an expert system shell or an AI language. The complexity of the application package, and the intricacies of the required data preparation, are governing factors as to which approach to take. The application might be able to sustain the restrictions of an expert system shell, but the KBFE could easily expand such that the development has to be written around the shell's limitations. One possible solution to this dilemma is to advance further the ESPRIT research programme to increase universal KBFE concepts. This could be expedited through active dissemination of research results, and an amalgamation of the remnants of the ALVEY and ESPRIT IFE/KBFE groups. New research initiatives in the area of KBFE development would be extremely beneficial.

Commercial CFD packages have, until recently, often lacked an easy way of preparing, creating and entering data, particularly for users who infrequently run a package. There are many different aspects of defining a CFD problem, each of which requires expert tuition and guidance. KBFEs for CFD packages have enormous advantages for users who do not frequently use them, in that they become more accessible. The very nature of KBFEs is to remove the ambiguities associated with computer packages. Conversing with the user in their own language enables data to be extracted, and synthesised into package commands through the use of symbolic processing. Symbolic processing is a major component in expert systems, and as such is the only logical way with which to effectively create knowledge base systems. Furthermore, the flexibility and extensibility of increasing the knowledge base size, without affecting the reasoning process is a major advantage.

The continual research into the areas of natural language understanding, graphical user interfaces, user models, and symbolic computing techniques are all going to contribute to the advancement of universal KBFE concepts. A centrally coordinated committee of KBFE research projects would consolidate existing results and provide a forum through which adequate dissemination could occur. Dedicated journals relating to KBFE research would advantageous.

In conclusion, KBFE research has grown significantly over the past 15 years. The field has spawned new areas of research such as user modelling and natural language understanding. Progress towards universal KBFE concepts has been initiated through the ESPRIT project, which should be further encouraged.

CHAPTER 3

COMPUTATIONAL FLUID DYNAMICS

3.1 Introduction

This chapter aims to introduce the application area of the KBS development, namely Computational Fluid Dynamics. It briefly discusses why CFD is required and mentions two of the packages that are commercially available which implement different integration techniques. The selected target package for the development, PHOENICS, is then covered in more detail whereby the finite volume technique is briefly examined and the PHOENICS environment is presented. The data file structure is introduced, and a simple example for unconfined jet impingement is given. This is used to illustrate the PHOENICS Input Language (PIL). The reasoning behind the integration of Artificial Intelligence and a commercial CFD package is addressed, this is then extended to cover the type of knowledge required in order to formulate and specify a problem using CFD terminology. Finally, it is important to be able to establish a grid that can reliably predict the viscosity effects of near wall flows, Roberts (1971), which is governed predominantly by the mathematical models used, and still do not contravene the maximum allowable aspect ratio within the integration domain. A technique for one dimensional finite volume grid generation is presented that utilises the length of a region within the subdivided domain, a predefined minimum cell size and the maximum allowed aspect ratio.

3.2 The need for computational fluid dynamics

The processes of heat/mass transfer, chemical reactions and fluid flow pervade all aspects of human life. These processes can be observed in engineering equipment, in the natural environment and in living organisms. Engineering equipment and power production systems involve heat and fluid flow processes, as do heating and air conditioning plants. Combustion engines, aircraft, rockets, reactors, furnaces, heat exchangers, all involve chemical reaction and thermofluid processes. Heat transfer is important within the design and manufacture of electrical circuits as overheating has catastrophic effects. Pollution, storms, floods and fires are affected by fluid flow and heat/mass transfer. The human body resorts to temperature control through heat and mass transfer via perspiration and complex non-Newtonian blood flow occurs through important organs such as the heart. As a consequence of the enormous influence the processes have on human life it is essential to be able to predict the behaviour in order to deal with them effectively. There

are two methods of predicting heat transfer and fluid flow behaviour: experimental investigation and numerical calculation.

Experimental investigation involves the physical measurement of velocities and temperatures within a flow region. Various techniques for velocity measurements are available which include hot wire and laser doppler anemometry, both are capable of measuring the fluctuation velocities and Reynolds stresses associated with turbulent flow. Temperature measurement is also possible through the use of various techniques depending on the application. Available methods can be divided into discrete or whole field measurement. Discrete measurement can be obtained by strategically placing direct contact instruments such as thermocouples and platinum resistance thermometers, whereas non-intrusive techniques includes the use of pyrometers. Whole field temperature measurement techniques includes the use of holography, which exploits the change of fluid density with temperature, and liquid crystal thermography. To aid the analysis of liquid crystal thermography, in order to obtain surface heat transfer coefficients, it is possible to apply image processing techniques, Ashforth-Frost et al. (1992).

Numerical calculations for heat transfer, fluid flow and chemical reaction involve the discretisation of the governing differential equations: Navier-Stokes equations; energy equation and the conservation of chemical species equation, into a form appropriate to the integration technique being utilised. Finite volume, Patankar (1980), and finite element, Taylor and Hughes (1981), techniques are the most popular numerical methods used for fluid flow. Extensive research throughout the world, over many years, has yielded many powerful numerical simulation packages. CFD involves the numerical study of fluid flow within predefined geometries based on the solution of discretised governing differential equations. It is important not to solely rely on numerical techniques to establish a full field solution to a problem. Initial corroboration of numerical and experimental results generates a degree of confidence for the numerical solution. Attaining correlation of initial experimental and numerical results allows the solution of different configurations to be used with a greater degree of confidence.

3.3 Computational fluid dynamics packages

Established research institutions / industrial organisations develop and modify their own CFD code to suit the situations that are under analysis. To allow industrial engineers the

facility to analyse fluid flow, several commercial CFD packages have been made available, such as PHOENICS (Parabolic Hyperbolic Or Elliptic Numerical Integration Code Series) and FLOTRAN. PHOENICS utilises the finite volume discretisation method, whereas FLOTRAN employs the finite element technique. FLOTRAN has been developed with the integration into existing pre- and post-processing software as a major specification, whereas CHAM, the developers of PHOENICS, appear to have followed a more independent approach. File conversion programs could be written for PHOENICS to enable compatibility, only if the structure of the resulting data files were known. Indeed, FEMVIEW limited have recently been working with CHAM to develop independent pre- and post-processing facilities for PHOENICS. Collaboration with CHAM has provided FEMVIEW with the necessary information describing the structure of the resulting data files, so that data conversion programs could be written.

For historical reasons, PHOENICS (Version 1.4, 1.5 and 1.6) was the preferred CFD tool within the Department of Mechanical Engineering at The Nottingham Trent University, and as such was an ideal target package for the development of a Knowledge Based Front End (KBFE).

3.4 PHOENICS environment

PHOENICS is a general purpose finite volume CFD package that solves the governing differential equations of motion, conservation of chemical species and heat transfer. The finite volume technique discretises the governing differential equations using various interpolation schemes between adjacent cells within a complete integration domain. An introduction into numerical heat transfer and fluid flow is given by Patankar (1980). The PHOENICS environment is shown in Figure 3.1 which consists of a pre-processor (SATELLITE), the processor (EARTH) and the post-processor (PHOTON).

3.4.1 SATELLITE

SATELLITE is an interpreter which receives an instruction stack, or data file (Q1.DAT) and translates the contents into a subsequent data file, EARDAT.DAT, which is directly read by the main processor, EARTH, upon executing the PHOENICS solution algorithm. The translation that takes place is essentially performed by FORTRAN code that recognises the format of valid PIL commands which describes the overall analysis to be

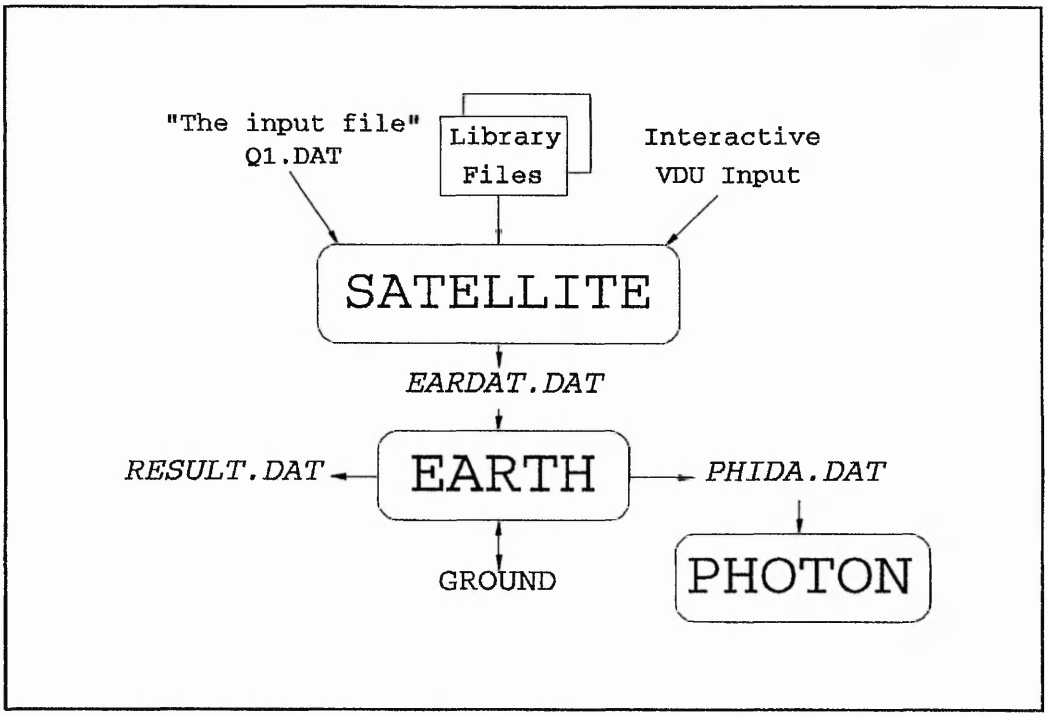


Figure 3.1: The PHOENICS environment

performed. These commands are then synthesised into a highly structured numeric file, EARDAT.DAT, which is read by EARTH.

3.4.2 EARTH

EARTH is the main processor of PHOENICS and contains the main solution algorithm which solves the complete set of discretised linear algebraic equations associated with the analysis. The results of the analysis are written to two files, RESULT.DAT and PHIDA.DAT. RESULT.DAT is used to display the results in a cell by cell, slab by slab manner so that the user can have easy access to the whole field values. PHIDA.DAT is accessed by PHOTON in order to display graphically the results in the form of contours and velocity vectors. PHIDA.DAT is also used for restarts when the user feels that the results obtained from the previous run did not converge.

3.4.3 PHOTON

PHOTON (**PH**OENICS **O**u**T**put **O**ptio**N**) is the post processor that is used to represent the resulting flow fields and scalar fields with vectors and contours respectively. PHOTON uses different data entry commands to those defined in PIL.

3.5 Description of physical phenomena using PHOENICS

3.5.1 Dependent variables

Throughout the integration domain PHOENICS can solve for pressure, velocities, temperatures and concentrations for one or two phase flows. These properties are known as dependent variables and PHOENICS can solve for a default number of fifty such variables. For a single phase analysis the available dependent variables are ...

P1 Pressure
U1 the x-direction velocity
V1 the y-direction velocity
W1 the z-direction velocity
R1 the volume fraction
KE the turbulence kinetic energy
EP the dissipation rate of turbulence kinetic energy
H1 the specific enthalpy
C1 concentration variable
C3 another concentration variable
.
.
.
C35

The velocities, volume fraction, enthalpy and concentration variables have corresponding second phase dependent variables U2, V2, W2, R2, H2, C2, C4, C6, ..., C34.

Independent variables within PHOENICS consists of time and the three space dimensions. Time is measured in the early-to-late direction with the variable "T". The three space

dimensions x, y and z are identified using the cardinals WEST to EAST, SOUTH to NORTH and LOW to HIGH respectively, Figure 3.2.

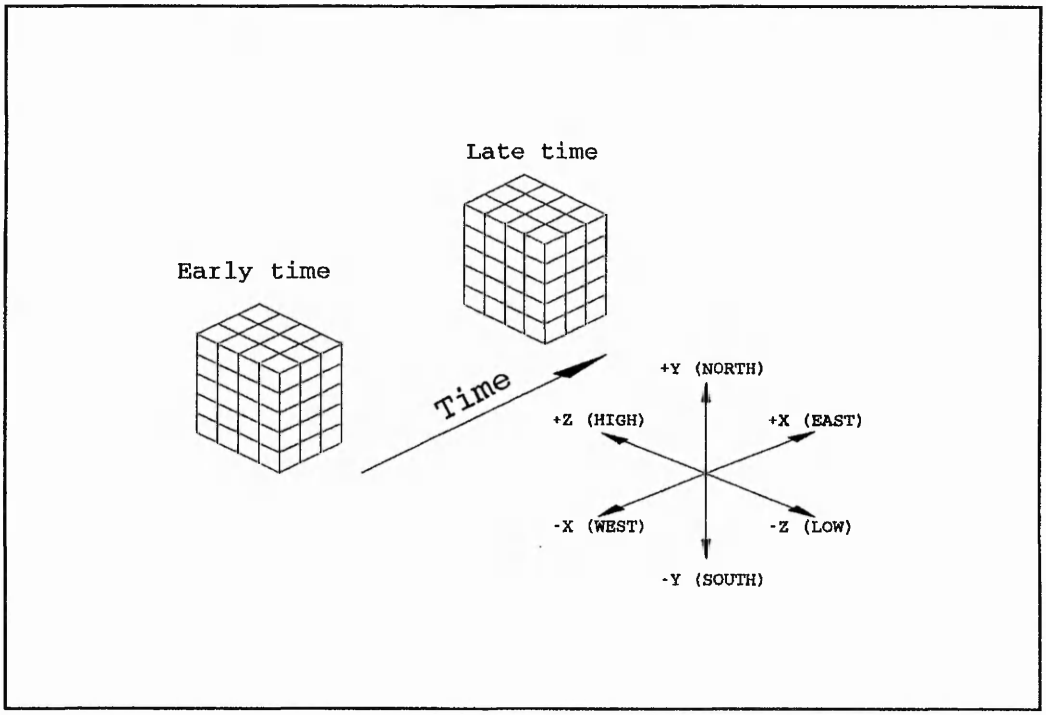


Figure 3.2: PHOENICS cardinal notation

3.5.2 Discretisation of the continua

Pressure and other scalar dependent variables are computed at the centre of a set of finite cells that are linked together to form a complete geometry. Velocities are calculated at the cell faces on a staggered grid in order to eliminate pressure checker-boarding and wavy velocity fields through implementing a piece-wise linear interpolation scheme for the pressure and continuity equations, Patankar (1980). The relative positions of the computed scalar dependent variables and velocities for a two dimensional grid in x and y is shown in Figure 3.3.

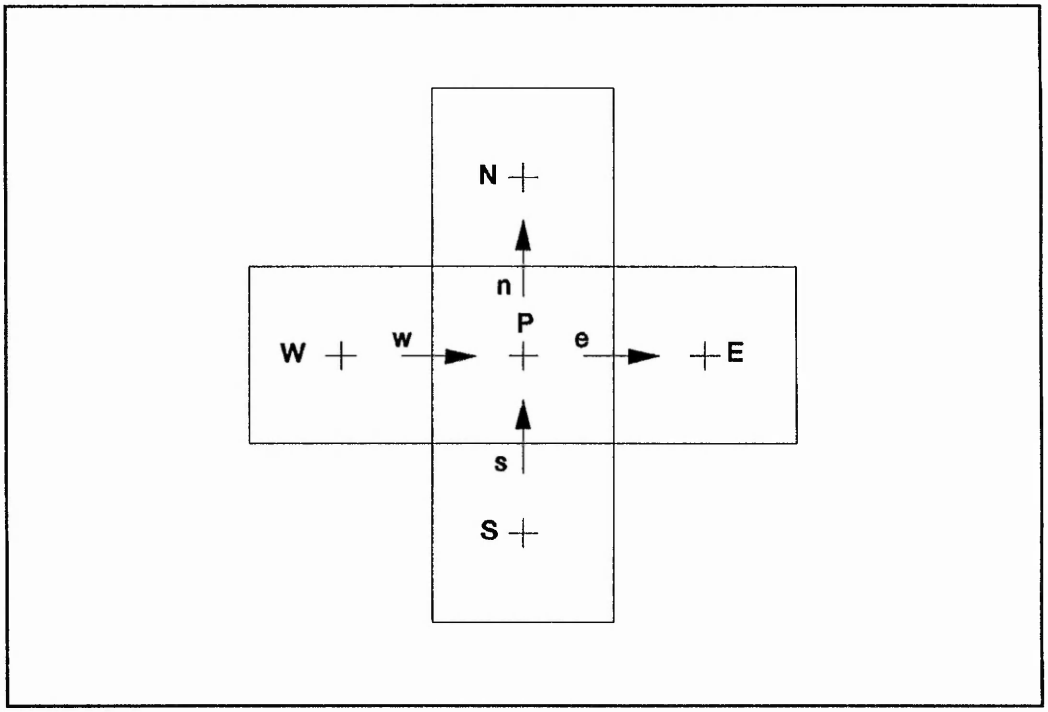


Figure 3.3: PHOENICS staggered grid

3.5.3 General differential equation solved by PHOENICS

The Navier-Stokes equations (momentum equations), energy equation and the conservation of chemical species equation can all be represented using the general differential equation given by equation (3.1).

$$\underbrace{\frac{\partial}{\partial t}(\rho\ \phi)}_{\textit{Transient term}} + \underbrace{div(\rho\ U\ \phi)}_{\textit{Convection term}} - \underbrace{div(\Gamma\ grad\ \phi)}_{\textit{Diffusion term}} + \underbrace{S}_{\textit{Source term}} \qquad (3.1)$$

3.5.4 Linear algebraic discretisation equations

The finite volume discretisation for equation (3.1), applied over an entire integration domain, results in the creation of a set of linear algebraic equations. Each dependent

variable has an associated algebraic equation at each cell which takes on the general form shown in equation (3.2).

$$\phi_P = \frac{a_E\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S + a_H\phi_H + a_L\phi_L + a_T\phi_T + S}{a_E + a_W + a_N + a_S + a_H + a_L + a_T} \qquad (3.2)$$

where ϕ is the dependent variable under consideration and the subscripts P, E, W, N, H, and L correspond to the locations at which the variable is computed, in accordance with the North, South, East, West, High and Low cardinal convention illustrated in Figure 3.2. Furthermore, T relates to the previous time step. The coefficients in equation (3.2), given as $a_{E, W, N, S, H, L, T}$, are functions of the discretisation geometry (grid size), in-cell Peclet number and the physical properties of the fluid. The full discretisation equations and their associated coefficients for one, two and three dimensions are given, along with their derivations, in Patankar (1980).

3.5.5 Solution to the algebraic equations

The algebraic equations given by equation (3.2) for different dependent variables are often strongly coupled. This is particularly true for velocity and pressure and their associated correction equations. This strong linkage dictates that an iterative technique be implemented for the solution of the complete set of algebraic equations for a given integration domain. PHOENICS utilises several iteration schemes which sweep through the domain updating the coefficients and necessary terms after the current iteration cycle is complete. The modifications are performed using the newly computed values of the dependent variables.

3.5.6 Problem specification

The popular method of entering data to fully describe a problem, using PHOENICS, is to create a data file, Q1.DAT, which contains the complete definition in command language form. To simplify the understanding of the PHOENICS command structure, the data file has been split into twenty four sections or groups (excluding a relatively new group used for debugging purposes), this is a common division throughout the PHOENICS environment. The twenty four groups are ...

GROUP 1. Run title and other preliminaries.
GROUP 2. Time dependence and related parameters.
GROUP 3. x-direction grid specification.
GROUP 4. y-direction grid specification.
GROUP 5. z-direction grid specification.
GROUP 6. Body-fitting and other grid distortions.
GROUP 7. Variables (including porosities) named, stored and solved.
GROUP 8. Terms (in differential equations) and devices.
GROUP 9. Properties of the medium (or media).
GROUP 10. Interphase-transfer processes and properties.
GROUP 11. Initialisation of fields of variables, porosity etc.
GROUP 12. Unused
GROUP 13. Boundary and internal conditions and special sources.
GROUP 14. Downstream pressure (for free parabolic flow).
GROUP 15. Termination criteria for sweeps and outer iterations.
GROUP 16. Termination criteria for inner iterations.
GROUP 17. Under-relaxation and related devices.
GROUP 18. Limits on variables or increments to them.
GROUP 19. Data communicated by SATELLITE to GROUND
GROUP 20. Preliminary print-out
GROUP 21. Frequency and extent of field print-out.
GROUP 22. Location of spot-value and frequency of residual print-out.
GROUP 23. Variable-by-variable field print-out and plot and/or tabulation of spot-values and residuals.
GROUP 24. Preparations for continuation runs.
GROUP 25. DEBUG group.

The command language consists of three templates that effectively cover all of the commands available within PIL. These templates are ...

Variable = value

Command(argument 1, argument 2, ..., argument n)

Command(ϕ) = value

where

Variable : This can be a system or a user declared variable. User declared variables are initially given a specific type with the commands ARRAY, BOOLEAN, CHAR, INTEGER or REAL. System variables such as RHO1 and ENUL are provided with default values which can be superseded by user values.

Value : This is an appropriate assignment with which to instantiate the variable with.

Command : This is a PIL command that is used to define the problem or to specify solution control.

Arguments : The arguments are related to the specific command that is being used to define the problem. Arguments can consist of, but are not restricted to, "Y" (yes), "N" (no), ϕ's, variables and numeric values.

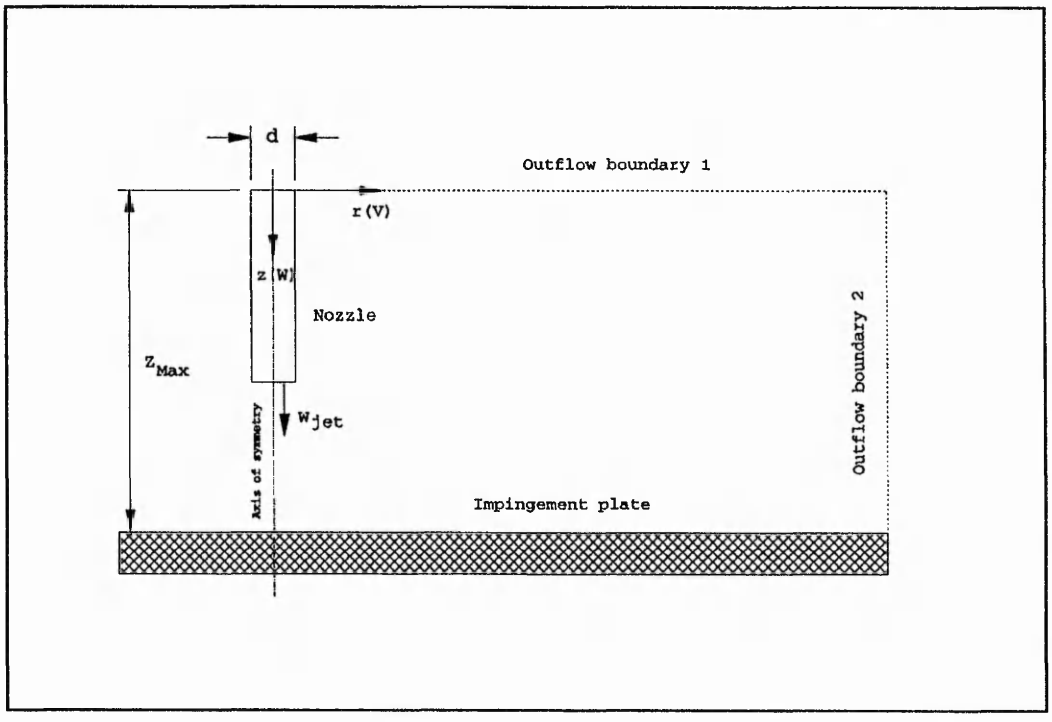


Figure 3.4: Unconfined jet impingement

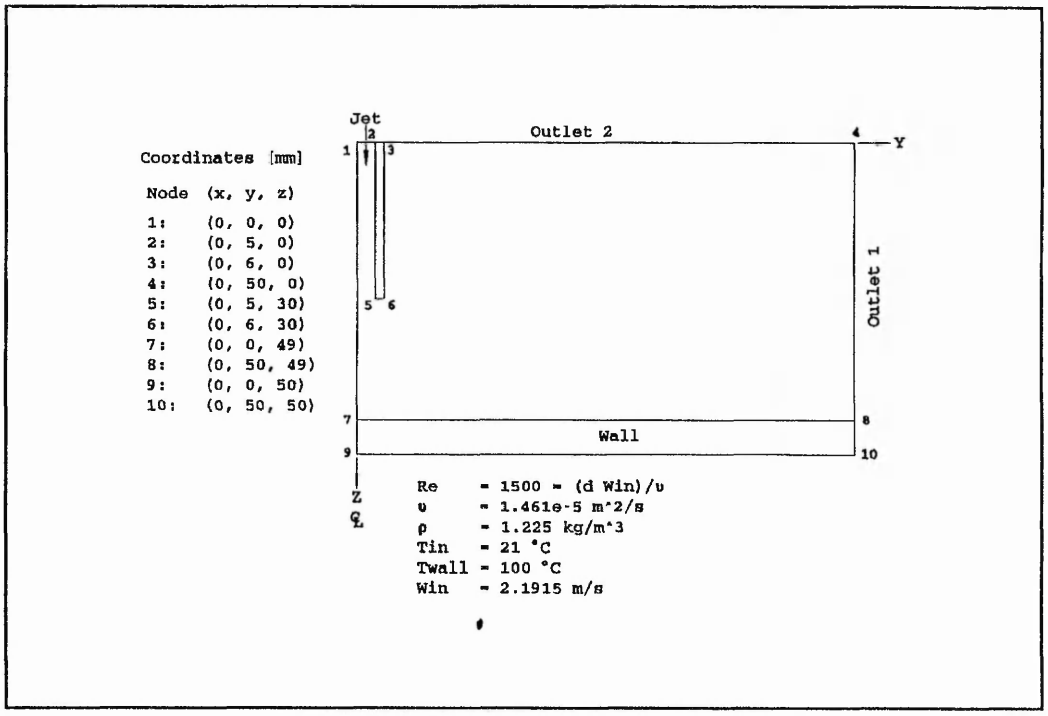


Figure 3.5: Geometry and Specification associated with the two dimensional unconfined thermal jet impingement

Figure 3.4 and Figure 3.5 show a typical jet impingement geometry which can be modelled using the PHOENICS data file shown in Figure 3.6. Figure 3.4 indicates the physical situation, and Figure 3.5 illustrates the geometry and the information required to be entered into PHOENICS. The data file shown in Figure 3.6 is simplistic in that the analysis is two dimensional, isothermal, incompressible and laminar. This serves only to illustrate how a beginner might feel when initially confronted with such a bemusing collection of commands and declarations. The fundamental commands that are most frequently used are those that specify the geometry and define the grid, specify the boundary/initial conditions, define the fluid properties and control and monitor the convergence of the solution.

3.6 The need for artificial intelligence interaction

CFD packages tend to be generic in so much as the developers try to create code that is as versatile as possible. This is clearly the case for PHOENICS, and because of its

```
Talk=f; Run(1, 1);vdu=tty
Text(2D Unconfined impinging round jet)
Real(win,Re,D)
Re=2000; D=10.0e-3
win=Re*enul/D
```
Preliminary Information

```
Cartes=F
subgrd(y,1,11,d/2,1.0)
subgrd(y,12,12,1.0e-3,1.0)
subgrd(y,13,49,0.369,1.0)
subgrd(z,1,18,0.03,1.0)
subgrd(z,19,37,0.02,1.0)
```
Grid definition

```
Solutn(p1,y,y,y,n,n,n)
Solutn(v1,y,y,n,n,n,n)
Solutn(w1,y,y,n,n,n,n)
```
Dependent variables

```
enul=1.461e-5; rho1=1.2250
```
Fluid properties

```
conpor(0.0,cell,1,1,-12,-12,1,-18)
```
Grid definition

```
fiinit(v1)=win
fiinit(w1)=win
```
Initial values

```
patch(plate,hwall,1,1,1,ny,nz,nz,1,1)
coval(plate,v1,1.0,0.0)
coval(plate,w1,fixval,0.0)
patch(inlet,low,1,1,1,11,1,1,1,1)
coval(inlet,p1,fixflu,rho1*win)
coval(inlet,v1,onlyms,0.0)
coval(inlet,w1,onlyms,win)
patch(outlet1,low,1,1,13,ny,1,1,1,1)
coval(outlet1,p1,fixval,0.0)
patch(outlet2,north,1,1,ny,ny,1,nz,1,1)
coval(outlet2,p1,fixval,0.0)
```
Boundary conditions

```
lsweep=300
resref(p1)=1.0e-8
resref(v1)=1.0e-8
resref(w1)=1.0e-8
relax(p1,linrlx,0.8)
relax(v1,falsdt,0.5)
relax(w1,falsdt,0.5)
output(p1,y,y,y,y,y,y)
output(v1,y,y,y,y,y,y)
output(w1,y,y,y,y,y,y)
iymon=14; izmon=33
nplt=1
stop→
```
Solution algorithm control parameters and result presentation commands

Figure 3.6: PHOENICS Q1.DAT data file after Figure 3.5

tremendous versatility it has been widely used as a research tool by academics and industrialists. However, increased versatility causes difficulties when initial exposure to the software is experienced, this directly affects the rate at which proficiency with the code is attained. Experience has shown that in order to be able to use PHOENICS effectively an understanding of the fundamentals associated with numerical heat transfer and fluid flow is required, as well as becoming proficient with the specific command language provided for data entry. As experience develops it is possible to interact with the solution algorithms to carry out various tasks such as introducing non-standard fluid property models.

Initially there are three methods which can be used to become familiar with any computer package. Firstly, by attending the necessary training courses and being taught how to use the software. This is undoubtedly the easiest approach to take because expert advice and guidance is constantly at hand. Although training courses can be costly. Secondly, by capitalising on the in-house experience with the software. Thirdly, if no in-house experience exists then embark on a self learning programme, utilising supplied training files and appropriate literature. The latter method is not a preferred option because the rate of learning significantly reduces due to the overwhelming quantity of information that needs to be considered to locate the detailed data that might resolve immediate problems. Asking for information considerably reduces the time spent looking through manuals.

Mehta and Kutler (1984) highlighted ten areas within CFD where expertise should reside, these included the construction and analysis of numerical methods for solving the governing differential equations through appropriate algorithm development. With the ability to use commercial CFD code such requirements automatically become redundant. Thus leaving the need to capture expertise in areas such as problem definition and input, selection of appropriate turbulence models, grid generation, and boundary / initial condition assignments. Such areas of expertise are characteristic of those required for the specification of a problem using commands within a data file for a CFD package.

The command languages provided by software developers are usually individualistic and occasionally appear very ambiguous for the novice user or beginner. This situation could be easily improved if a front end to a software package could be developed that converses with the user in his native language and translates his requirements into the necessary commands to fully describe the analysis. To introduce such a front end would potentially

allow the availability and accessibility of the software to increase several fold because of the relative ease with which a user could enter problem specifications. The use of AI techniques and languages are ideally suited to such a problem because of the level of symbolic processing involved. The necessary numerical processing could easily be provided by the integration of traditional numerical code provided by FORTRAN or C. The twenty four groupings, previously mentioned, within PHOENICS have provided an ideal categorisation of the knowledge bases in the final KBFE for the isolation of rules relating to the generation of the final data file.

3.7 Computational fluid dynamics knowledge elicitation

Throughout the duration of using PHOENICS a self learning programme was utilised to become familiar with the intricacies of the software. This was the primary method of knowledge acquisition in conjunction with conversing with other experienced users of PHOENICS. Information required for the formulation and specification of a flow analysis consists of fundamental parameters such as fluid properties, boundary conditions and initial field values in conjunction with the grid data. Possibly the most important area of simulating fluid flow with a CFD package is the generation of a suitable mesh that would capture the flow characteristics of the geometry under consideration. Abbot et al. (1988) evaluated the existence and value of "expert knowledge" in the use of CFD, and analysed the nature of the knowledge for the possible development of a knowledge based assistant. Implementing experimental procedures on predefined problems with two novice users and a CFD expert resulted in the highlighting of a potential need for various expert assistants. The proposed expert assistants consisted of: Engineering Assistant, Grid Generator, Convergence Expert, Flow Analyst, Assumption Maker, Data Display Expert and Application-Specific Experts.

3.8 Aspect ratio dependent finite volume grid generation

3.8.1 Introduction

The discretisation process associated with the finite volume technique requires that the integration domain is made up of quadrilateral or cuboid orthogonal cells for a two or three dimensional analysis respectively. In order to accommodate this, PHOENICS provides five methods of specifying the geometry and corresponding cell discretisation.

The methods are governed by the PIL commands SUBGRD, GRDPWR, FRACtions (XFRAC, YFRAC, ZFRAC, TFRAC) and Body-Fitted Coordinates (BFCs). Specifying the grid using FRACtions can be performed using either the "method of pairs" or the "direct method". Each method, except BFCs, specify the cell face positions, on the appropriate axis, which extend through the entire domain, see Figure 3.7. BFCs allow the user to define a distorted grid that follows the geometrical body contour of the required flow region.

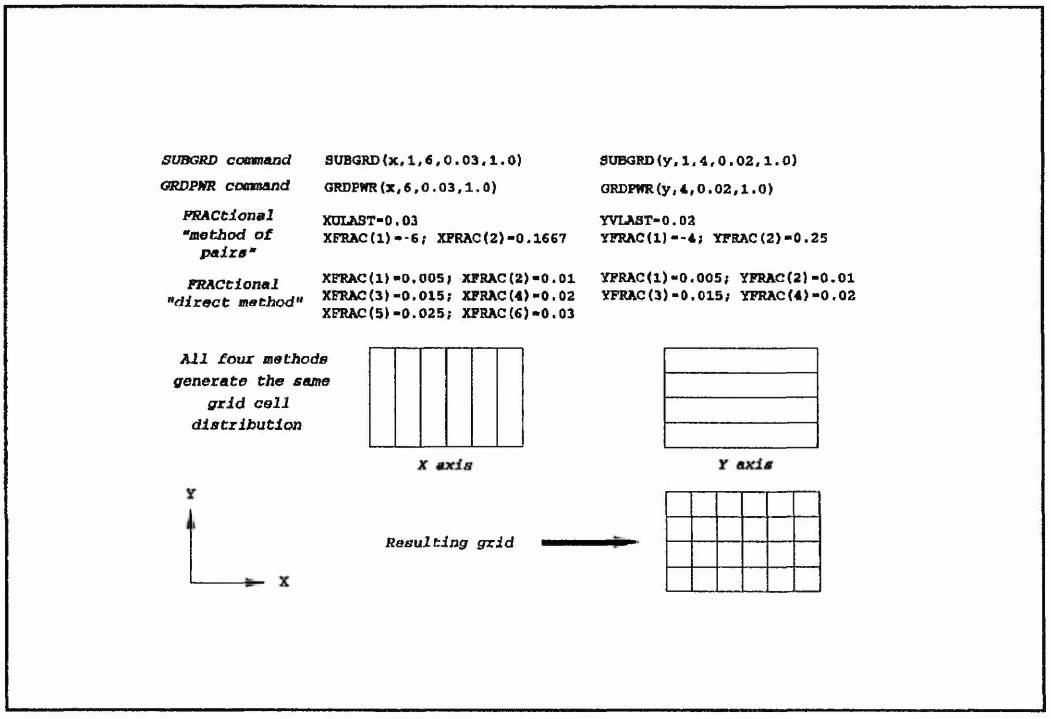


Figure 3.7: PHOENICS grid generation commands

Grid generation methods have been, and are continually being researched in order to automate meshing procedures, Vogel (1989) and Blacker et al. (1988a and 1988b). Most physical problems inherently require the use of BFCs which have to be as orthogonal as possible, for finite volume discretisation, and can be promoted by applying mapping procedures similar to that introduced by Ryskin and Leal (1983 and 1984), Gilding (1988) and Wang and Georgiadis (1989). The orthogonal mapping technique essentially transforms a two dimensional cartesian system (x,y) of coordinates onto an orthogonal boundary-fitted system (ξ,η), Figure 3.8, and is defined by the covariant Laplace equation,

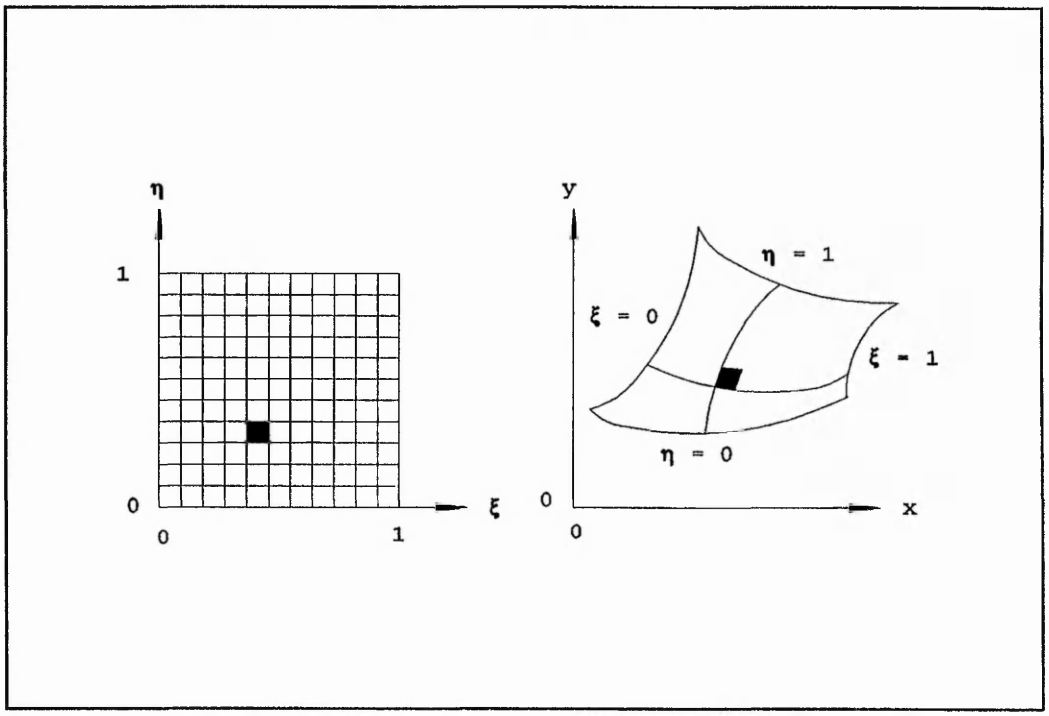


Figure 3.8: Orthogonal mapping. Two dimensional cartesian onto a two dimensional boundary fitted system

Ryskin and Leal (1983). The generation of the cartesian set of coordinates is a relatively simple task for a uniform grid and should abstractly represent the shape of the physical domain, Gilding (1988). For CFD it is desirable to capture near wall viscous effects, such as boundary layer growth, by refining the grid next to wall boundaries and gradually increasing the size of the cells as the distance from the wall increases. This enables increased computational efficiency without requiring a fine uniform mesh within the domain. Anderson et al. (1984) presents several transformations for cell distributions with continuously varying cell sizes. The transformation equations cover grid clustering near a wall, in a duct, and near an interior point of a computational grid. The presented transformation equations are derived from the work performed by Roberts (1971) which considers the generation of computational meshes for boundary layer problems. Roberts established a cell distribution profile between two walls for a given number of cells, Figure 3.9, and is given by equations **(3.3)** to **(3.7)**.

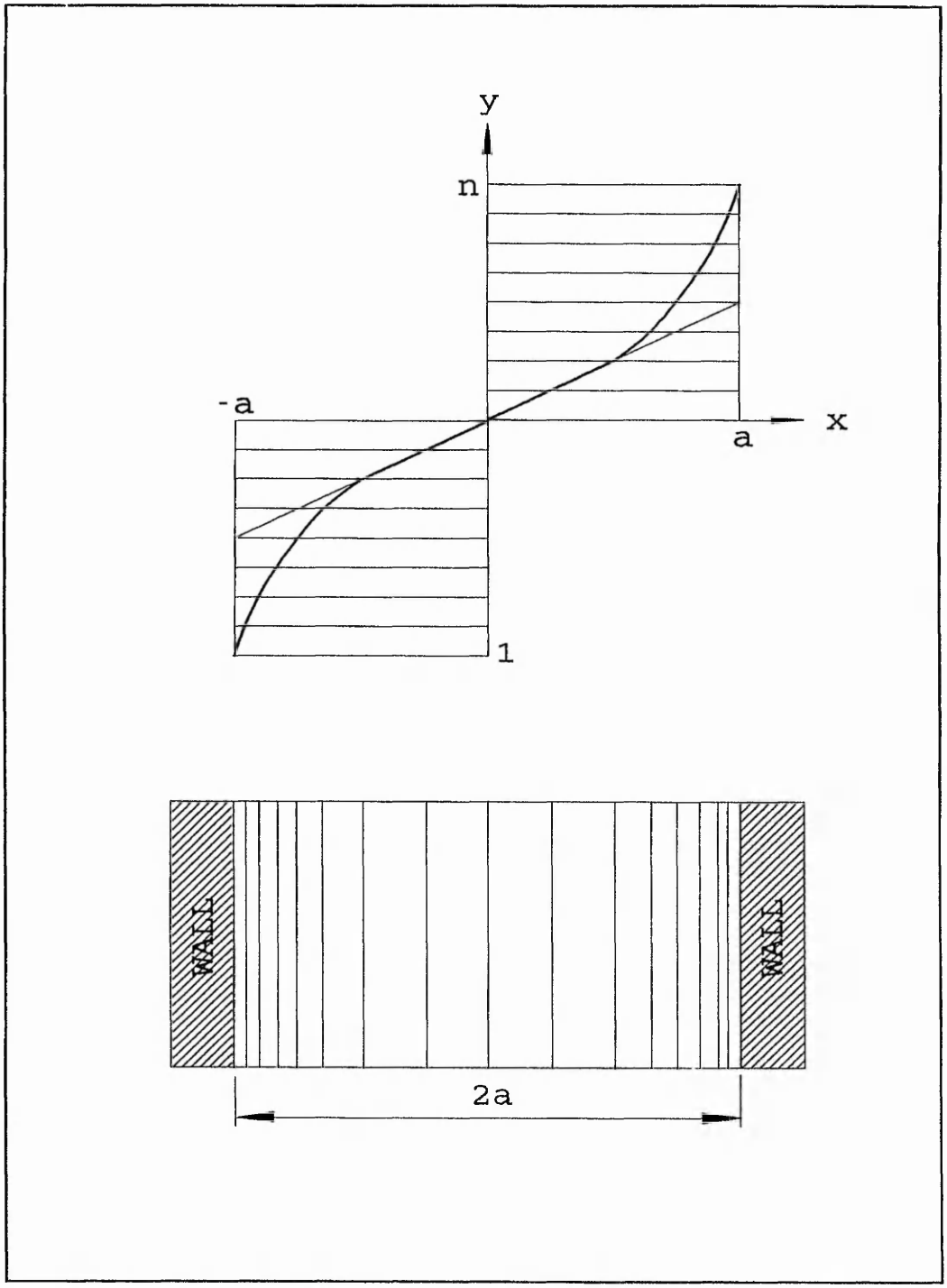


Figure 3.9: Roberts' (1971) cell distribution profile between two walls for a given number of cells

$$y = \frac{n+1}{2} + \frac{n-1}{2}\left\{\frac{(1-\alpha)x}{a} + \alpha\left(\frac{f\left(\frac{a-x}{d}\right) - f\left(\frac{a+x}{d}\right)}{f(0) - f\left(\frac{2a}{d}\right)}\right)\right\} \tag{3.3}$$

where

$$f(z) = e^{-z} \tag{3.4}$$

or

$$f(z) = (1+z)^{-1} \tag{3.5}$$

or

$$f(z) = (1+\log(1+z))^{-1} \tag{3.6}$$

or

$$f(z) = (1+\log\{1+\log(1+z)\})^{-1} \tag{3.7}$$

where α is the proportion of the mesh points which represent the boundary layer ($0 < \alpha < 1$) and n is the number of cells within the region to be meshed which is of length $2a$. The function, given by equation (3.3) varies linearly in the interior of the region and much more rapidly, on a length scale of order d, in the boundary layers. When applying the transformations to each axis and superimposing one on top of the other a mesh can be obtained. However, using this method it is not possible to limit the resulting aspect ratios within the grid, an important factor that can seriously affect both the convergence of a solution and the results, Abbott et al. (1988). Indeed, in the experiments that Abbot et al. performed, one of the novices experienced difficulty with obtaining solution convergence for a particular analysis. Eventually, it was discovered that the cell aspect ratios at the inlet were excessive at 40:1 and after considerable reduction convergence was obtained. It is generally accepted that the aspect ratio for CFD should not exceed 10:1, when using finite volume code, a limit advised by CHAM through private communications. Furthermore, the cell density should be sufficient to obtain a grid independent solution. Grid independency is usually a result of an iterative analysis procedure whereby the mesh density progressively increases. Grid independency is achieved when the predictions from

the previous analysis do not significantly vary with the present results. The transformations proposed by Roberts (1971) rely on the fact that the number of cells required is known *a priori*.

As Abbott et al. illustrated, aspect ratios are important and the absence of this factor with the technique presented by Roberts (1971) is a clear deficiency. To remove this deficiency, a technique has been developed that establishes a cell distribution within a one dimensional cartesian space, similar to Roberts, utilising the minimum cells size, maximum aspect ratio, and the height/length of a region. This distribution is governed by the smallest cell residing adjacent to a wall, and extending through the entire domain as shown in Figure 3.7. The approach uses a dynamically generated function to establish successive cell aspect ratios, taking the base metric as being the smallest cell size, which varies from unity up to the predefined maximum. The cell aspect ratios are used to establish the subsequent co-ordinate relative to the previous and the minimum cell size.

3.8.2 Symmetric formulation, grid clustering in a duct

Figure 3.10 shows an arbitrary region enclosed between two parallel plates. The smallest cell size, **L**, the maximum aspect ratio, **AR**, and the height of the region **h**, are all pre-defined. It is required to obtain the number of cells and their associated co-ordinates whereby the cell aspect ratio, λ, continuously varies within the range $1 \leq \lambda \leq AR$. Furthermore, the cell distribution should be symmetric about the centre line.

For the required distribution successive non-dimensional co-ordinates are given by :-

$$y_{i+1} = y_i + \lambda_i \left(\frac{L}{h}\right) \tag{3.8}$$

where

$$i = 0, 1, 2, 3, ..., N$$

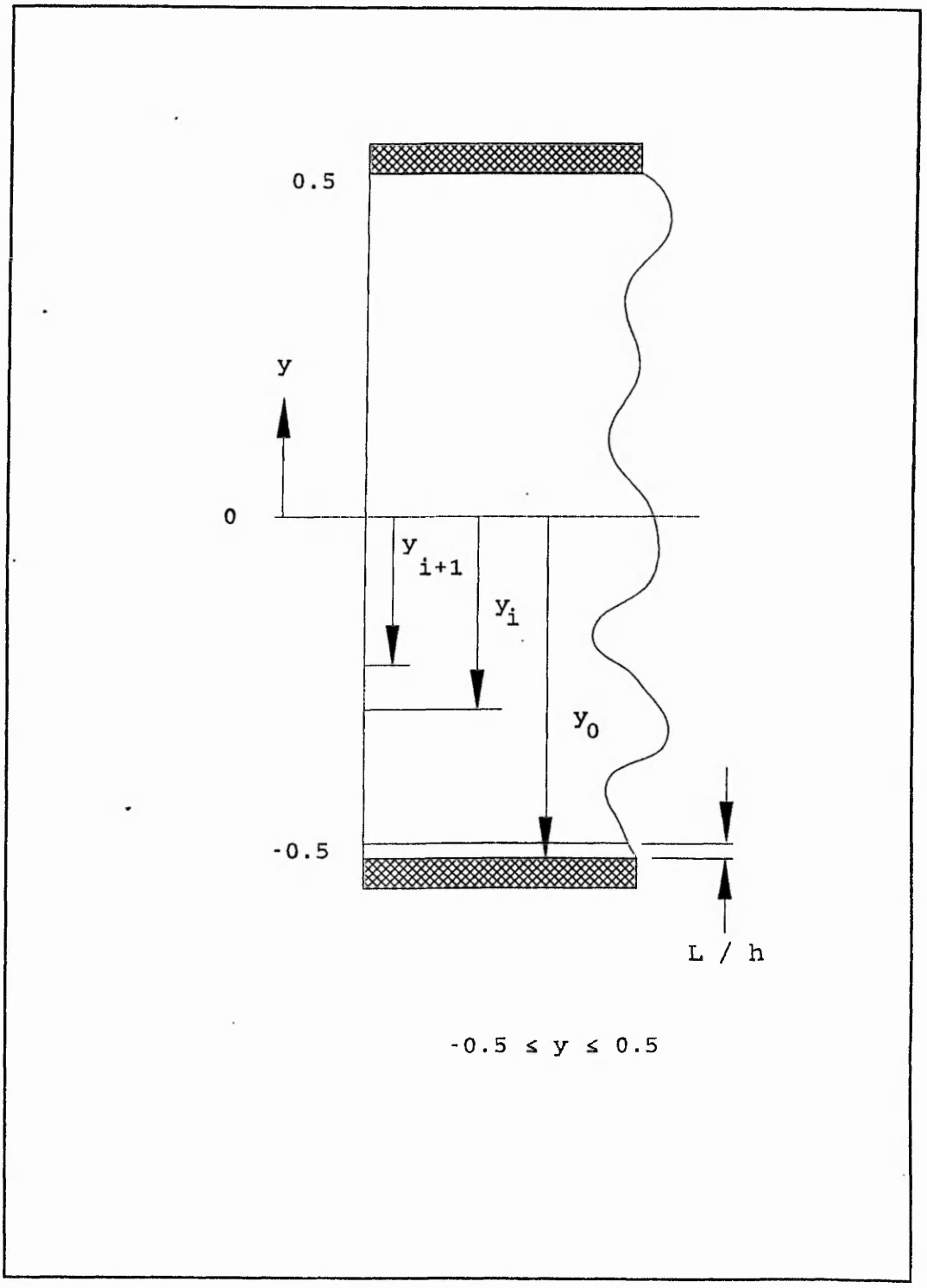


Figure 3.10: Arbitrary one dimensional region between two parallel plates

$$\lambda_i = 10^{f(2\pi y_i)\log_{10}(AR)} \tag{3.9}$$

$$\lambda_i = 10^{f(2\pi y_i)\log_{10}(AR)} \quad 0.0 \le f(2\pi y_i) \le 1.0 \tag{3.10}$$

$$y_0 = -0.5 \tag{3.11}$$

$$y_N = 0.5 \tag{3.12}$$

N = calculated number of cells within the region

Therefore, from equation **(3.9)**, it can be seen that the cell aspect ratio, λ, can vary within the range 1 to AR, this is provided that the limits of the function $f(2\pi y_i)$ are maintained, as given by **(3.11)**.

Development of the profile function, $f(2\pi y_i)$

Equation **(3.8)** indicates that for the cell size to continuously vary then the cell aspect ratio also needs to continuously vary. This leads to an initial triangular profile being assessed for the correct characteristics to satisfy the limitations placed on the profile function, Figure 3.11. In order to model the profile a Fourier series was developed to correlate the relationship. This was chosen because of the ability of a Fourier series to model discontinuous relationships. The standard Fourier series is given by ...

$$f(\theta) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(n\theta) + b_n \sin(n\theta)] \tag{3.13}$$

$$a_0 = \pi^{-1} \int_{-\pi}^{\pi} f(\theta) d\theta \tag{3.14}$$

$$a_n = \pi^{-1} \int_{-\pi}^{\pi} f(\theta)\cos(n\theta) d\theta \tag{3.15}$$

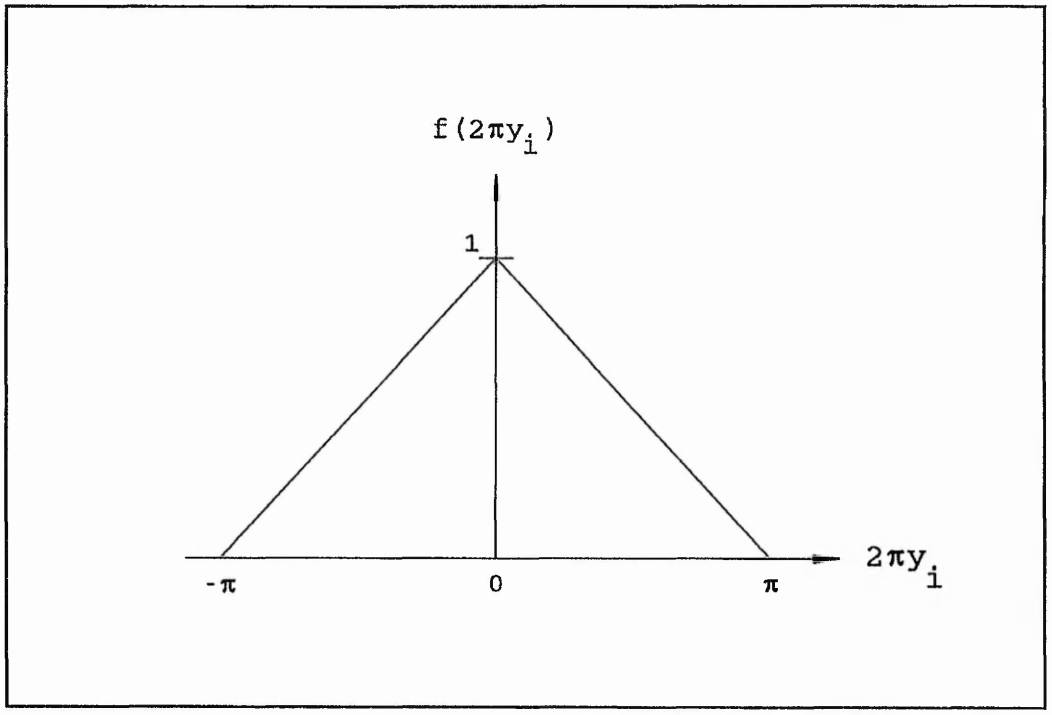


Figure 3.11: Initial triangular profile function

$$b_n = \pi^{-1} \int_{-\pi}^{\pi} f(\theta) \sin(n\theta) d\theta \qquad (3.16)$$

... where the function $f(2\pi y_i)$ is regionally defined. The triangular profile proved to be inadequate because of the inability to force the centre cell to be located directly on, or evenly distributed about the apex, f(0). The profile was then modified in order to account for the ill-positioning of the mid-point cell, Figure 3.12. The profile characteristics α_1, α_2, and Ψ were initially pre-defined to force the profile to be triangular, as shown in Figure 3.11. As the local aspect ratios are calculated, $(\alpha_1/2\pi)$ is instantiated as the absolute value of the penultimate negative value of y. Furthermore, $(\alpha_2/2\pi)$ is instantiated as the absolute value of y just preceding the penultimate negative value. Having obtained α_1 and α_2, Ψ is given by equation **(3.17)**, with Ψ_N rounded up to the next **ODD** integer.

where

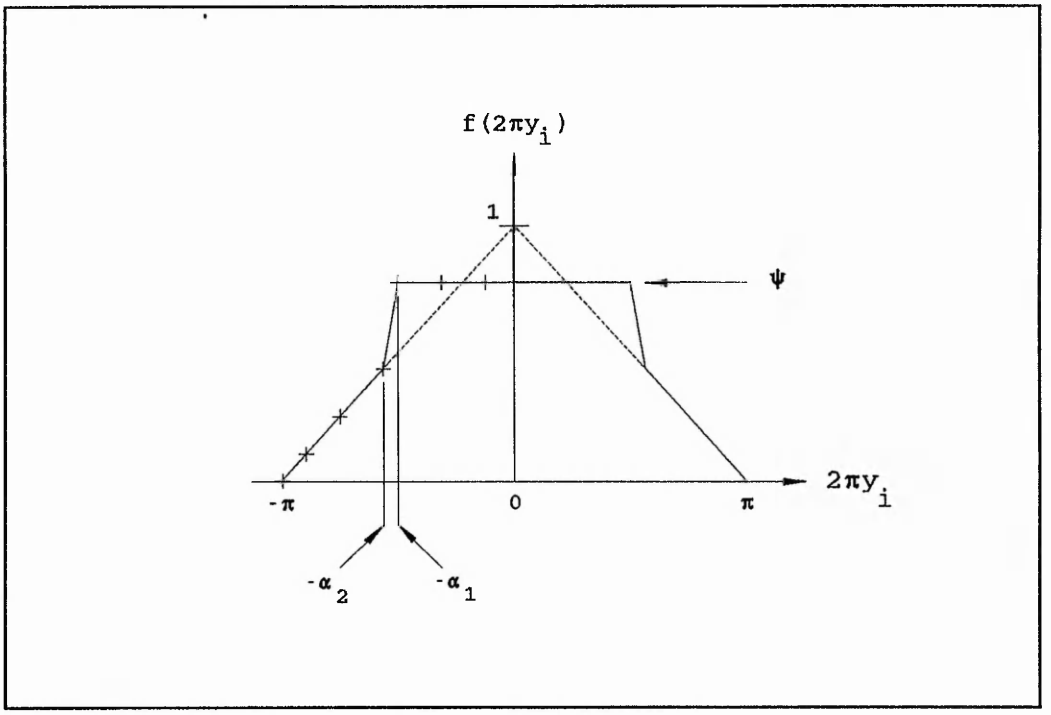


Figure 3.12: Modified triangular profile function

$$\psi = \frac{\log_{10}[\alpha_1 h(\pi \psi_N L)^{-1}]}{\log_{10}[AR]} \tag{3.17}$$

$$\psi_N = \frac{\alpha_1 h}{\pi\ AR\ L} \tag{3.18}$$

However, the modified profile, although resolving the problem related to the ill-positioning of the mid-point cells through the use of Ψ, given by equation **(3.17)**, created a further problem that can only be illustrated through an example. Establishing the Fourier series coefficients for Figure 3.12, and fixing n to be 100 gives the results shown in Table 3.1.

Figure 3.13 and Figure 3.14 can be used to illustrate the problem associated with the modified triangular profile. For all y_i, where i = 1 to 9, the resulting cell distributions are calculated correctly using equation **(3.8)**. However, from points 10 to 14, the corresponding y_i values do not mirror those values associated with points 0 to 5, as shown

Maximum aspect ratio, **AR = 10**
Minimum cell size, **L = 1.7333 mm**
Height, **h = 60 mm**

i	y_i	$2\pi y_i$	$f(2\pi y_i)$	λ_i	y_{i+1}
0	-0.5	$-\pi$	0	1	-0.471
1	-0.471	-2.96	0.058	1.143	-0.438
2	-0.438	-2.752	0.1237	1.3296	-0.4
3	-0.4	-2.5133	0.201	1.5871	-0.354
4	-0.354	-2.224	0.2923	1.96	-0.293
5	-0.293	-1.841	0.4055	2.544	-0.224
6	-0.224	-1.4074	0.5526	3.5694	-0.121
7	-0.121	-0.7603	0.7588	5.7388	2.711

Assigning values to α_1, α_2 and Ψ gives

$$\alpha_1 = 1.4074,\ \alpha_2 = 1.841,\ \Psi = 0.7129$$

Altering the Fourier coefficients and recommencing the calculations yields :-

6	-0.224	1.4074	0.7108	5.1375	0.075
7	-0.075	0.4712	0.7129	5.1625	0.074
8	0.074	0.465	0.7129	5.1625	0.223
9	0.223	1.4012	0.7121	5.1529	0.372
10	0.372	2.3373	0.2563	1.8043	0.424
11	0.424	2.6641	0.1521	1.4195	0.465
12	0.465	2.9217	0.0701	1.1752	0.499
13	0.498	3.1347	0.0024	1.0056	0.528

Table 3.1: Tabulated progression through the Fourier Series for the modified triangular profile function

in Figure 3.14. The reasoning behind this lies with the assignment of the cell aspect ratio for y_9. For a comparable mirror image of the grid distribution about the centre line, y_{10} should be located roughly at the absolute value of y_5, thus:-

$$y_{10} = y_9 + \lambda_9\,(L/h)$$

and from Table 3.1

$$0.293 = 0.223 + \lambda_9\,(1.7333/60)$$

$$\therefore \lambda_9 = 2.4231$$

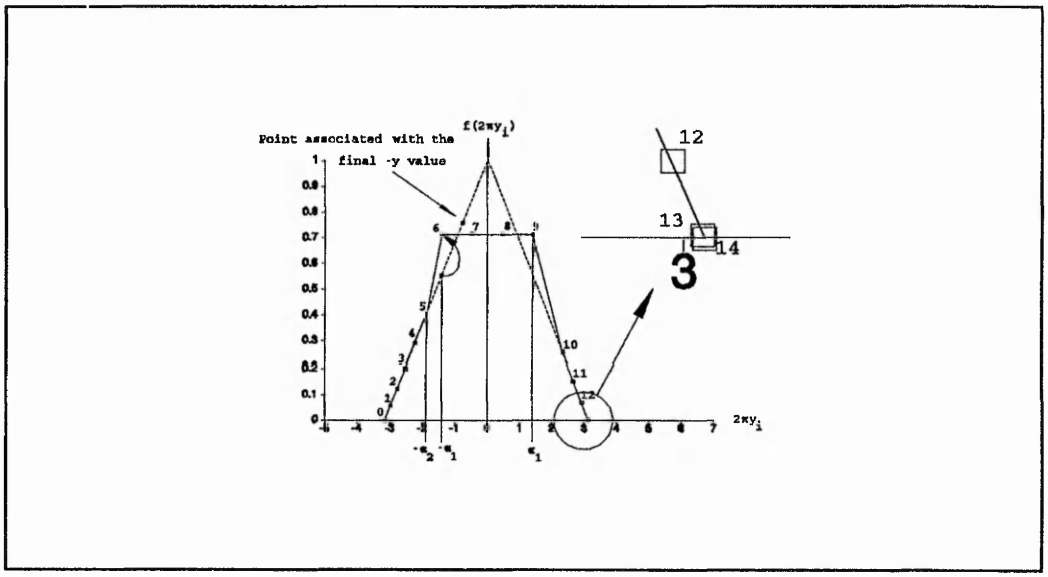


Figure 3.13: Problematic profile function

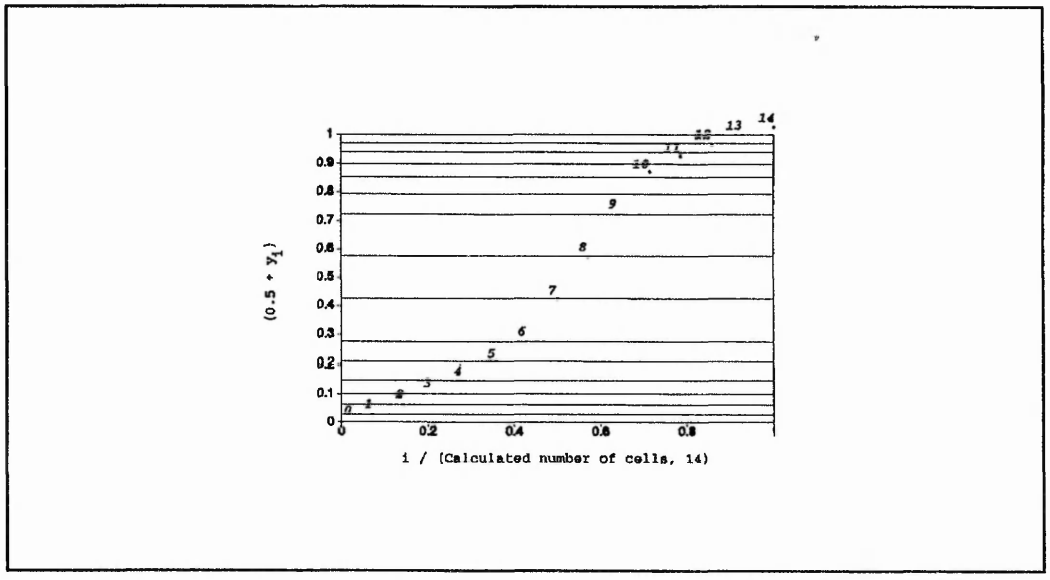


Figure 3.14: Resulting grid distribution after Figure 3.13

Now, equation **(3.9)** gives

$$f(2\pi y_9) = 0.3844$$

The most comparable value of $f(2\pi y_i)$ with $f(2\pi y_9)$ in Table 3.1 occurs at i = 5. A similar process takes place with cells 13 and 14 whereby the height of cell 14 should correspond with the smallest cell size. This implies that $f(2\pi y_{13})$ should be zero in order to force λ_{13} to be unity.

Resulting profile and equations for grid clustering in a duct

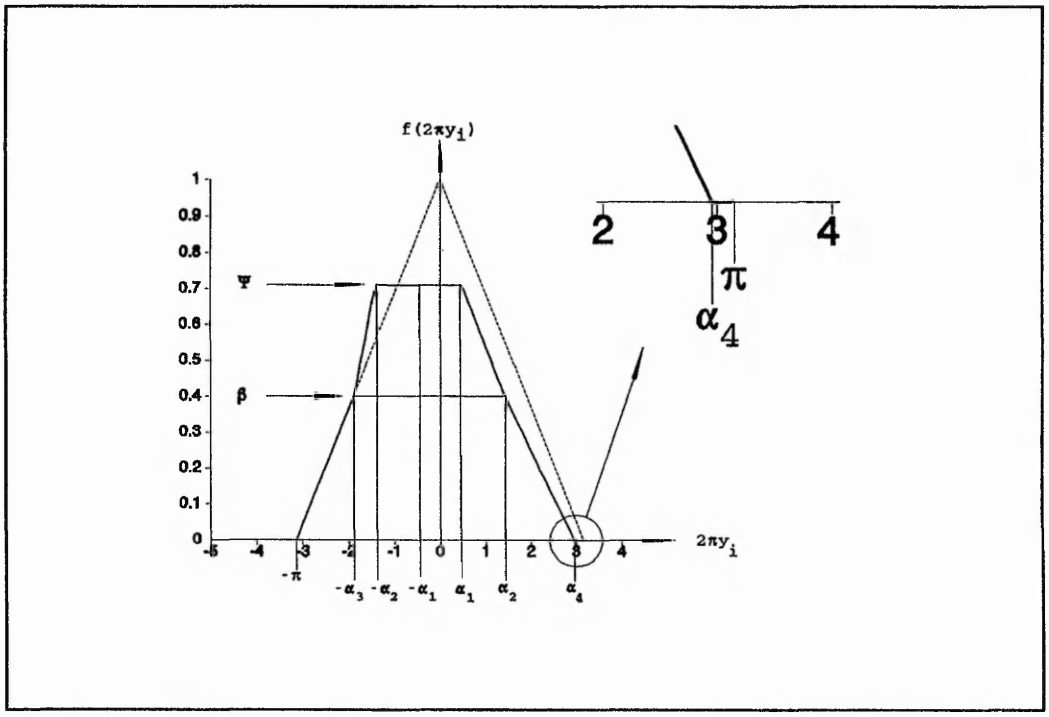


Figure 3.15: Profile function for grid clustering in a duct

The resulting profile function is shown in Figure 3.15, and is defined with equations **(3.19)**.

$$
\begin{aligned}
f(\theta) &= 1 + \frac{\theta}{\pi} && -\pi \le \theta \le -\alpha_3 \\
f(\theta) &= m_1(\theta + \alpha_2) + \psi && -\alpha_3 \le \theta \le -\alpha_2 \\
f(\theta) &= \psi && -\alpha_2 \le \theta \le \alpha_1 \\
f(\theta) &= m_2(\theta - \alpha_1) + \psi && \alpha_1 \le \theta \le \alpha_2 \\
f(\theta) &= m_3(\theta - \alpha_4) && \alpha_2 \le \theta \le \alpha_4 \\
f(\theta) &= 0 && \alpha_4 \le \theta \le \pi
\end{aligned}
\tag{3.19}
$$

$$\theta = 2\pi y_i \tag{3.20}$$

$$m_1 = \frac{\psi - \beta}{\alpha_3 - \alpha_2} \tag{3.21}$$

$$m_2 = \frac{\psi - \beta}{\alpha_1 - \alpha_2} \tag{3.22}$$

$$m_3 = \frac{\beta}{\alpha_2 - \alpha_4} \tag{3.23}$$

$$\alpha_1 = \left| 2\pi y_{I+1} \right| \tag{3.24}$$

$$\alpha_2 = \left| 2\pi y_I \right| \tag{3.25}$$

$$\alpha_3 = \left| 2\pi y_{I-1} \right| \tag{3.26}$$

$$\alpha_4 = \pi - \frac{2\pi L}{h} \tag{3.27}$$

$$\beta = f(2\pi y_{I-1}) \tag{3.28}$$

$$\psi = \frac{\log_{10}\left\{\alpha_2 h\,[\pi \psi_N L]^{-1}\right\}}{\log_{10}(AR)} \tag{3.29}$$

$$\psi_N = \frac{\alpha_2 h}{\pi\, AR\, L} \tag{3.30}$$

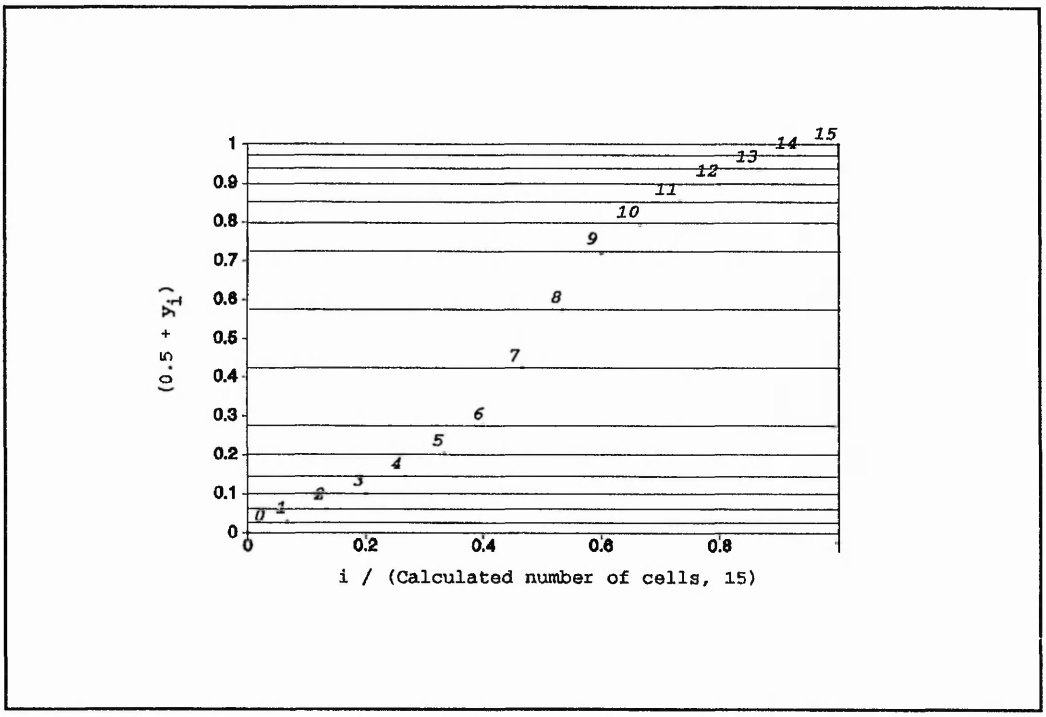


Figure 3.16: Resulting cell distribution after Figure 3.15

Appendix A gives the values for the Fourier series coefficients a_0, a_n, and b_n, as defined by equations **(3.14)** to **(3.16)**, using the profile function definitions given by **(3.19)**. In order to reduce the severity of the transition to/from the linear distribution, generated by the plateau on the profile function, from/to the continuously varying sections, it has been found that Ψ_N should be rounded up to the next **ODD** integer. Figure 3.16 illustrates the improvement on the symmetry of the grid distribution for the parameters shown in Table 3.1 (h = 60 mm, [illegible]3 mm and AR = 10). It must be noted, however, that the modified function increases the number of cells by one.

Evaluation of the profile parameters α_1, α_2, α_3, α_4, β and Ψ.

Initially the gradients m_1, m_2, and m_3 are preset to π^{-1}, $-\pi^{-1}$, and π^{-1} respectively. These are required to initially force the profile function to be triangular. Furthermore, the initial values of the profile constants are:-

$$\alpha_1 = \alpha_2 = 0, \text{ and } \alpha_3 = \alpha_4 = \pi$$

also

$$\beta = 0.0 \text{ and } \Psi = 1.0$$

Generation of the cell distribution commences from the initial value of $y_0 = -0.5$. Calculation of the associated cell aspect ratio from equation **(3.9)** and utilising equation **(3.8)** yields the next cell co-ordinate. This procedure is then repeated until y_i becomes positive, only then can α_2, α_3, β and Ψ be determined. Table 3.2 fully illustrates this procedure.

3.8.3 Generic formulation

Thus far the profile for determining the cell aspect ratios for successive cells has been presented which enables the generation of a grid distribution within a duct. The introduction of a clustering parameter, α, enables a generalisation of the procedure in order to obtain grid clustering near a wall as well as within a duct, as shown in Figure 3.17 and Figure 3.18. Introducing the clustering parameter, α, requires the modification of equations **(3.11)**, **(3.20)**, **(3.25)** and **(3.26)**, as follows:-

$$y_0 = \alpha - 1 \tag{3.31}$$

$$\theta_i = \pi y_i \, (1 - \alpha)^{-1} \tag{3.32}$$

$$\alpha_2 = \left| \theta_I \, (1 - \alpha)^{-1} \right| \tag{3.33}$$

$$\alpha_3 = \left| \theta_{I-1} \, (1 - \alpha)^{-1} \right| \tag{3.34}$$

Maximum aspect ratio, **AR = 10**
Minimum cell size, **L = 1.7333 mm**
Height, **h = 60 mm**

i	y_i	θ	$f(2\pi y_i)$	λ_i	y_{i+1}
0	-0.5	$-\pi$	0.002	1.0047	-0.471
1	-0.471	-2.9592	0.0581	1.1432	-0.438
2	-0.483	-2.7517	0.124	1.3306	-0.4
3	-0.4	-2.5102	0.201	1.5884	-0.354
4	-0.354	-2.2219	0.2928	1.9623	-0.297
5	-0.297	-1.8657	0.4061	1.5474	-0.223
6	-0.223	-1.4033	0.5533	1.5754	-0.12
7	-0.12	-0.7544	0.7599	5.7528	0.046

Now, y_{i+1} has become positive and **I** is classified as the index of the penultimate negative value of y_i

$$\therefore I = 6$$

Thus, using equations (3.25), (3.26) and (3.28), we have

$$\alpha_2 = 1.4033, \alpha_3 = 1.8657, \beta = 0.4061$$

Also, from (3.30) and (3.29), we have ...

$$\Psi_N = 1.5459 = 3 \text{ (rounded up to the next ODD integer)}$$

$$\therefore \Psi = 0.7121$$

Thus, re-commencing from i = I

6	-0.223	-1.4033	0.7104	5.1337	-0.075
7	-0.075	-0.4715	0.7127	5.1608	0.074

Increasing i to I+1, (3.24) and ? yield ...

$$\alpha_1 = 0.4712, \alpha_4 = 2.9601$$

and re-commencing

8	0.074	0.4652	0.7119	5.1516	0.223
9	0.223	1.4003	0.4072	2.5539	0.297
10	0.297	1.8639	0.2860	1.9319	0.353
11	0.353	2.2145	0.1945	1.5649	0.398
12	0.398	2.4986	0.1204	1.3194	0.436
13	0.436	2.7381	0.0579	1.1427	0.469
14	0.469	2.9455	0.0035	1.0082	0.5

Table 3.2: Tabulated progression through the final profile function

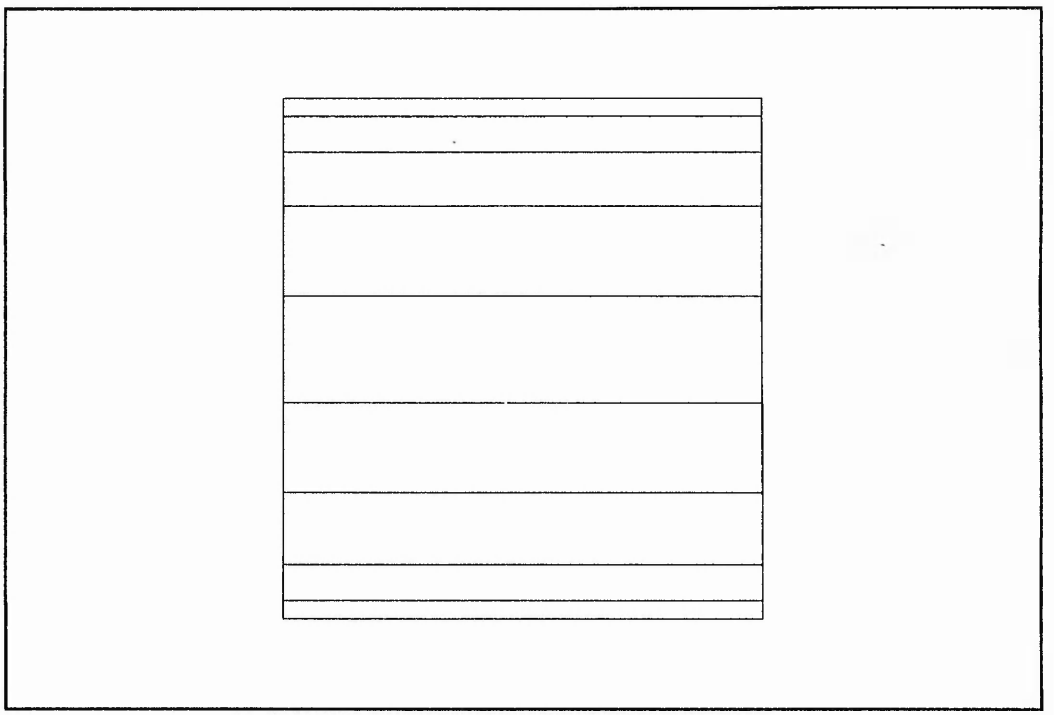


Figure 3.17: Grid clustering in a duct, $\alpha = 0.5$

Furthermore, for grid clustering near a wall, $\alpha = 0$, it has been found that Ψ_N should be rounded up to the next **EVEN** number. This ensures that the last cell is not half the size of the penultimate.

This technique has been successfully coded using the C programming language, Appendix B, which has allowed grid clustering near top and bottom walls to be achieved through the inversion of Figure 3.18. This is performed by recognising that $\alpha = 1$ corresponds with grid clustering near the top wall. Resulting grid distributions for various parameters can be seen in Figure 3.19 and Figure 3.20.

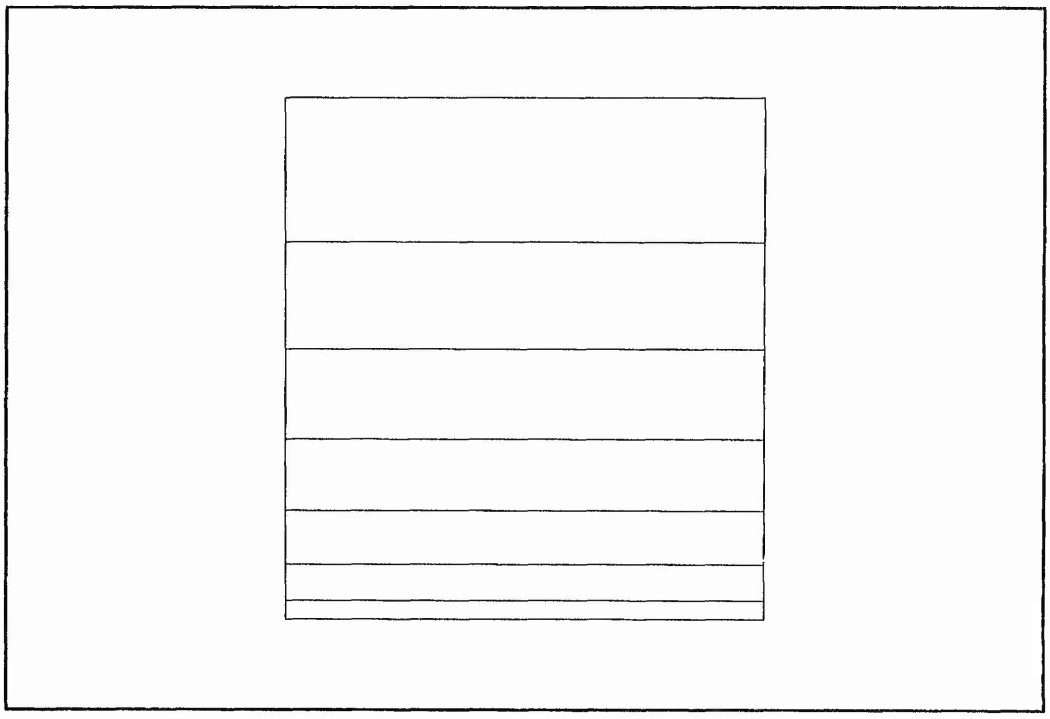


Figure 3.18: Grid clustering near a wall, $\alpha = 0.0$

3.8.4 Example

Considering the geometry shown in Figure 3.5, it is first necessary to regionalise the domain, as shown in Figure 3.21. Following the regionalisation, the alpha values are assigned to characterise the type of grid clustering required, as shown on Figure 3.21. Each region according to the reference axes is meshed separately, as shown in Figure 3.22 and Figure 3.23, given the minimum cell size and maximum allowed cell aspect ratio. Figure 3.24 shows the entire domain having been meshed using this technique.

3.9 Conclusions

A brief introduction has been given to the PHOENICS environment, its infrastructure and the terminology used to define CFD problems. An example has been given consisting of an unconfined jet impingement geometry and the associated specification. The corresponding data file is also presented. This example will be used throughout the thesis to describe some of the pertinent points and to illustrate various techniques related to knowledge representation, data storage and inferencing.

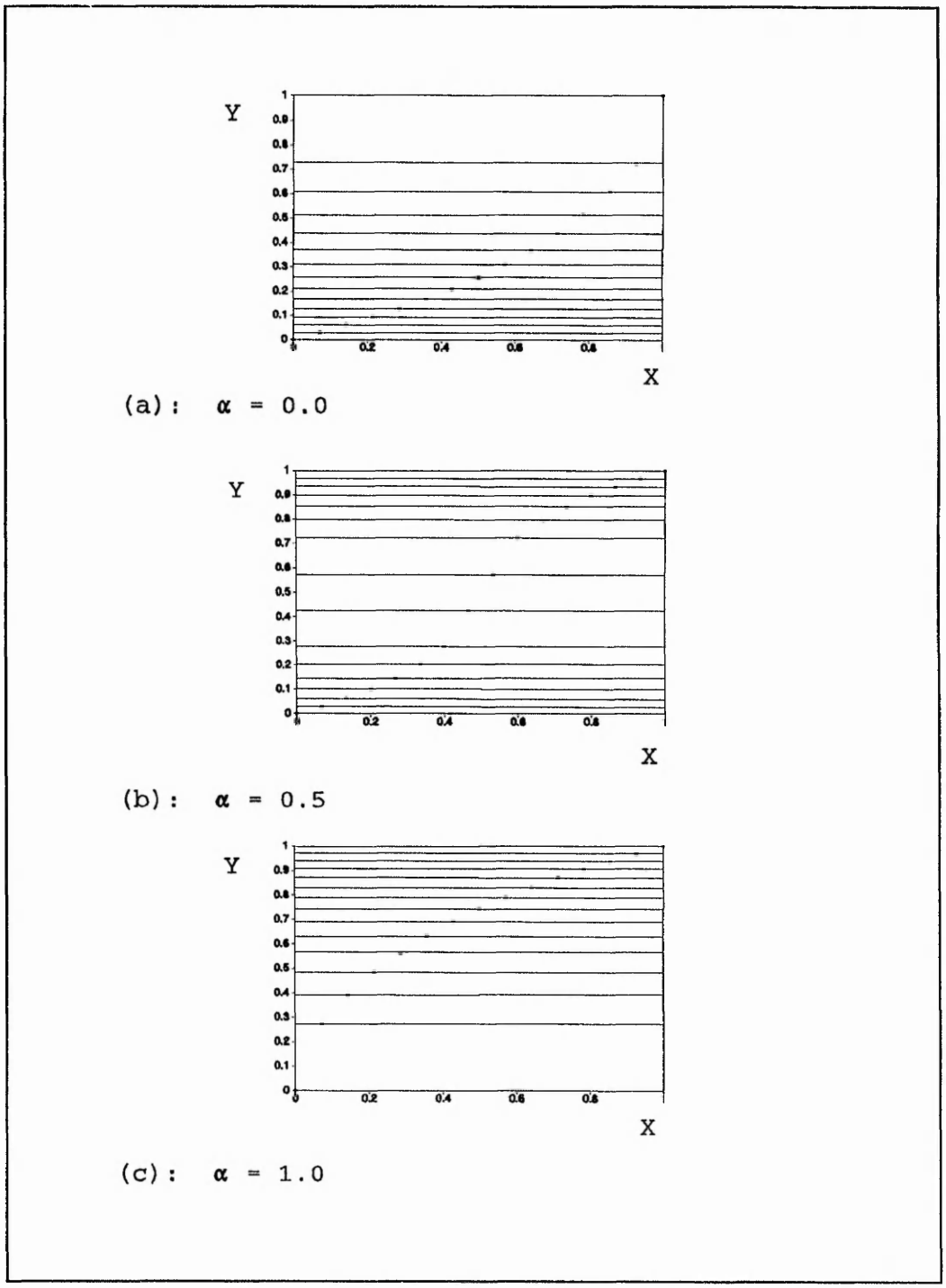


Figure 3.19: Grid clustering: AR = 10, L = 1.733 mm, and h = 60 mm. (a) $\alpha = 0.0$, (b) $\alpha = 0.5$, and (c) $\alpha = 1.0$

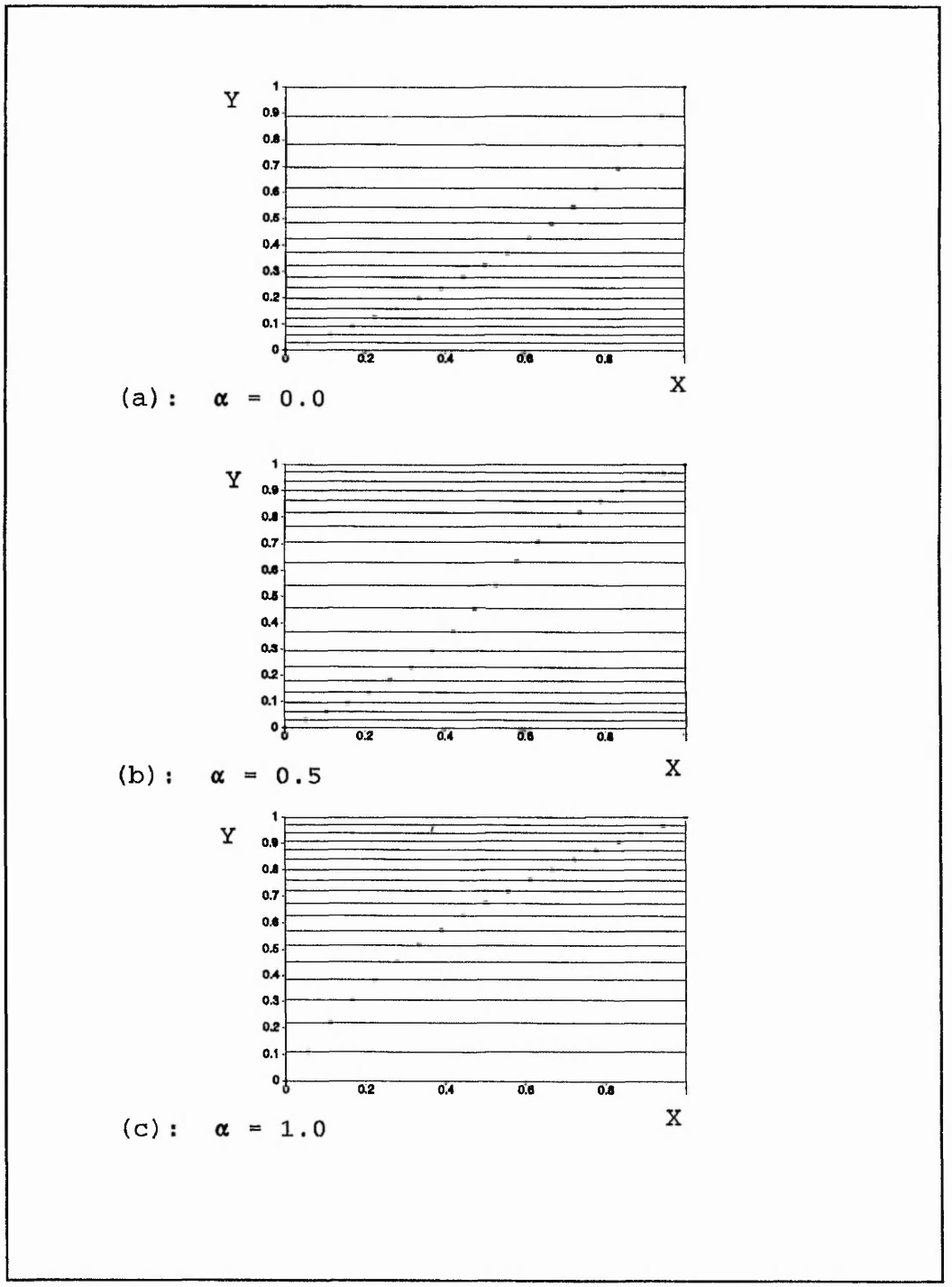


Figure 3.20: Grid clustering: AR = 5, L = 1.733 mm, and h = 0.3 mm. (a) α = 0.0, (b) α = 0.5, and (c) α = 1.0

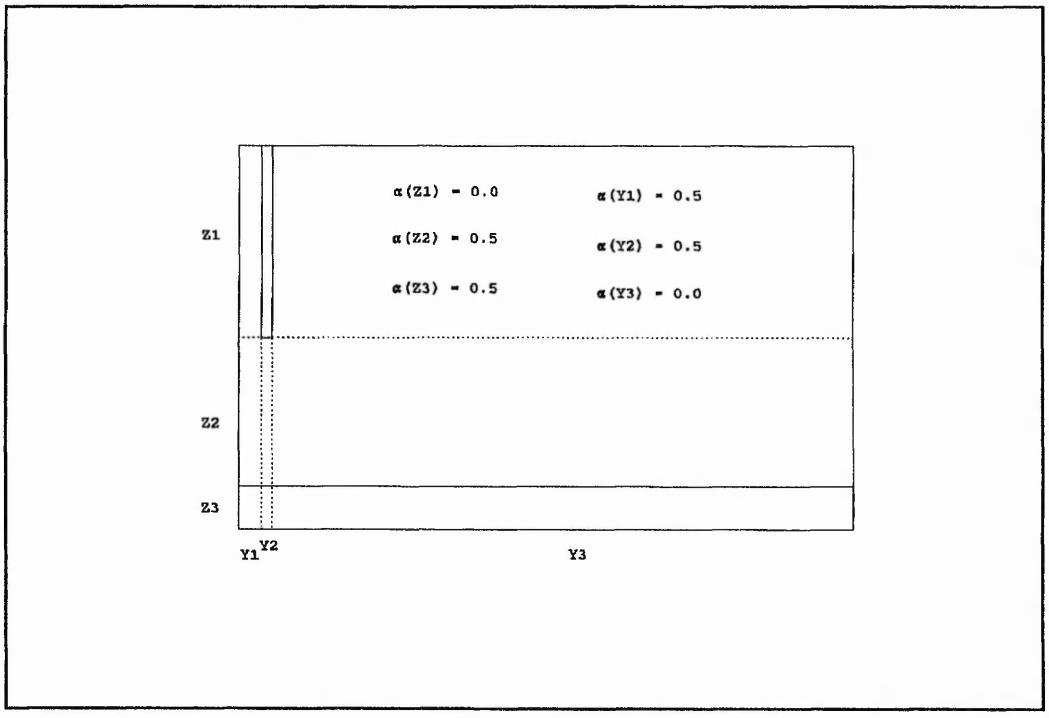


Figure 3.21: Regionalised domain after Figure 3.5

A technique for aspect ratio finite volume grid generation has been presented which was developed because cell aspect ratios can affect the resulting solution, Abbott et al. (1988). The technique ensures that the cell aspect ratios within an integration domain never exceed a predefined maximum. The methodology revolves around the height of a one dimensional region, the minimum cell size and the overall maximum aspect ratio. A one dimensional space is used, therefore allowing each axis to be considered independently, and then superimposed on top of each other to accommodate either two or three dimensional geometries.

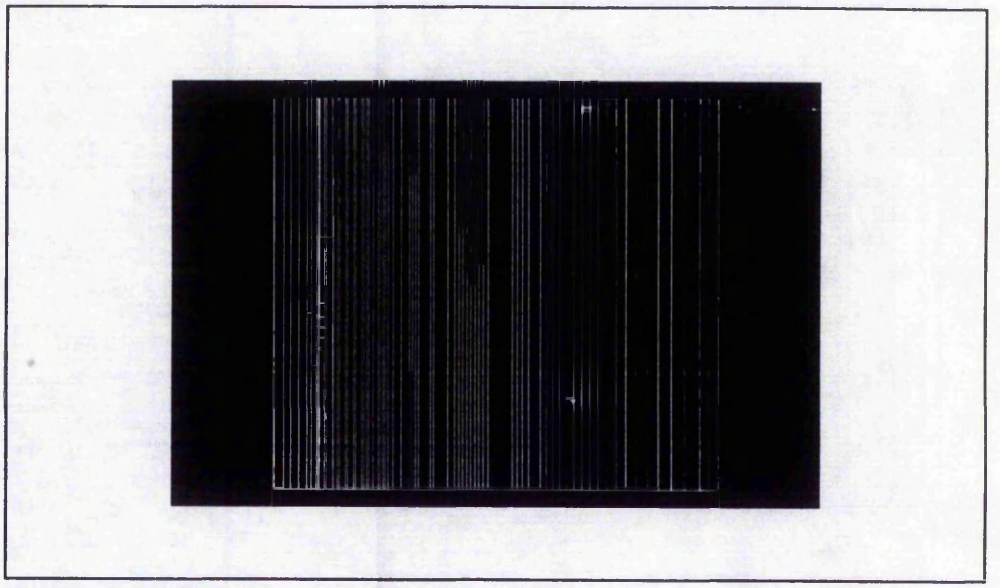


Figure 3.22: Y axis regional meshing

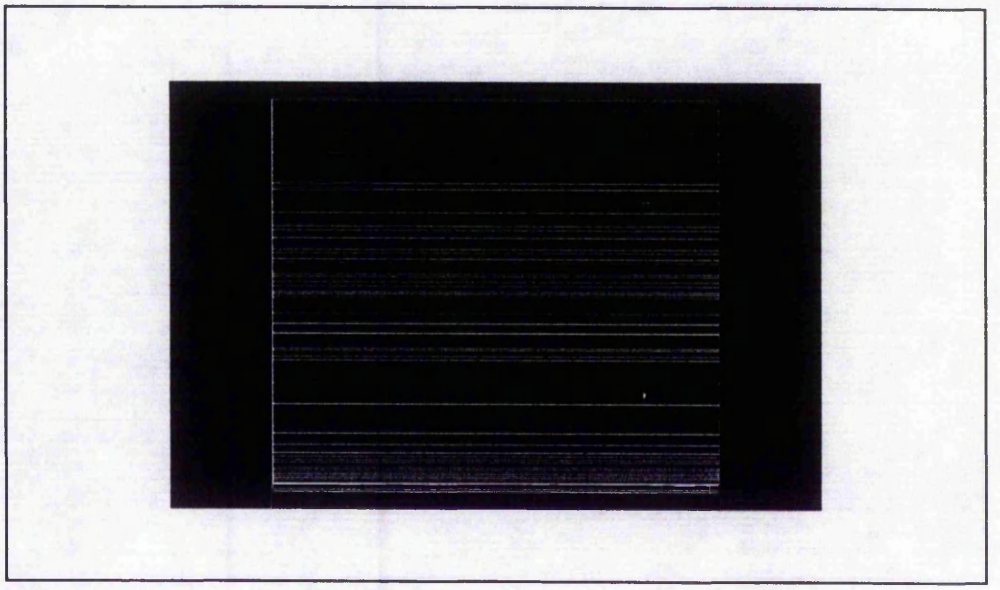


Figure 3.23: Z axis regional meshing

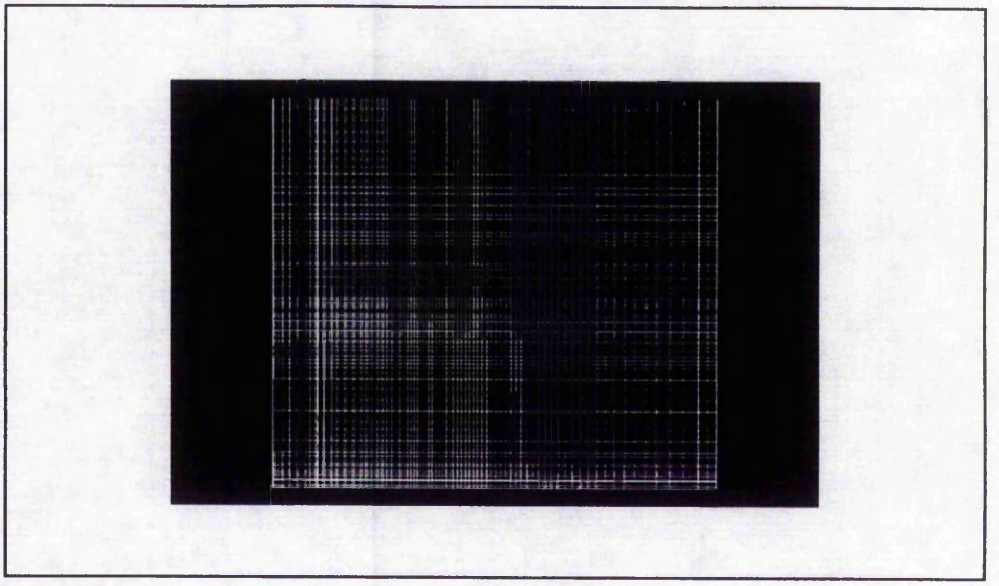


Figure 3.24: Entire meshed domain after Figure 3.5

CHAPTER 4

INTELLIGENT / KNOWLEDGE BASED FRONT ENDS

4.1 Introduction

British and European interest in Information Technology developed with the ALVEY and ESPRIT (European Strategic Programme of Research and development in Information Technology). Both programmes commenced in the early 1980s and have established research into Intelligent Front Ends and Knowledge Based Front Ends respectively. Essentially they endeavour to address the same difficulties experienced with novices using software packages. However, the two main differences as highlighted by Eustace (1985), were that ALVEY concentrated on the collaboration of competing companies, academia and industry, whereas ESPRIT was concerned with the collaboration of different countries. Furthermore, ESPRIT had an element that looked directly at Computer Integrated Manufacture (CIM), where ALVEY divided this into separate parts of the project.

The ALVEY programme of advanced IT research was a joint venture between three UK Government Departments (the Department of Trade and Industry, the Ministry of Defence, and the Department of Education and Science), British industry and academia. The three government departments acted through the Science and Engineering Research Council (SERC). The ALVEY programme was a five year commitment which commenced in 1983 and published its final report in October 1988. The objective was to stimulate British IT research into key technologies of Intelligent Knowledge Based Systems, Man/Machine Interfaces (MMI), Software Engineering, Very Large Scale Integration (VLSI) and Computing Architectures. The programme was named after Mr John Alvey, chairman of the 1982 committee which recommended that such a national programme should be mounted, in response to increasing overseas competition, and in particular to the Japanese Fifth Generation Computer Systems initiative.

Within the ALVEY programme the key technology of IKBS highlighted nine research themes: intelligent front ends; intelligent computer-aided instruction, expert systems, natural language understanding, image interpretation, declarative languages, inference and knowledge representation, parallel architectures and intelligent database systems. In 1983 the first IKBS research theme workshop relating to IFEs was held, Bundy et al. (1984). A second workshop was held a year later, Bundy (1984b).

February 1984 saw the first stage of the ESPRIT programme commence. The programme was to run over a ten year period, consisting of two five year phases, with three main objectives: Firstly, to provide European IT industry with the basic technologies it needed to meet the competitive requirements of the nineties; Secondly, to promote European industrial cooperation in IT; Finally, to contribute to the development of internationally accepted standards. ESPRIT concentrated on five technical areas: Microelectronics; Software technology; Advanced Information Processing (AIP); Office Systems; and Computer Integrated Manufacture. A document prepared by the commission of the European communities reports on the progress and results of the first phase, COM(86) 687 final. The technical area of AIP essentially dealt with the development of Information Processing technologies which addressed the following four key areas: Knowledge Engineering; External Interfaces; Information and Knowledge Storage; and Computer Architectures. The knowledge engineering sub-area is concerned with the construction and use of knowledge-based systems and software tools to help the development process.

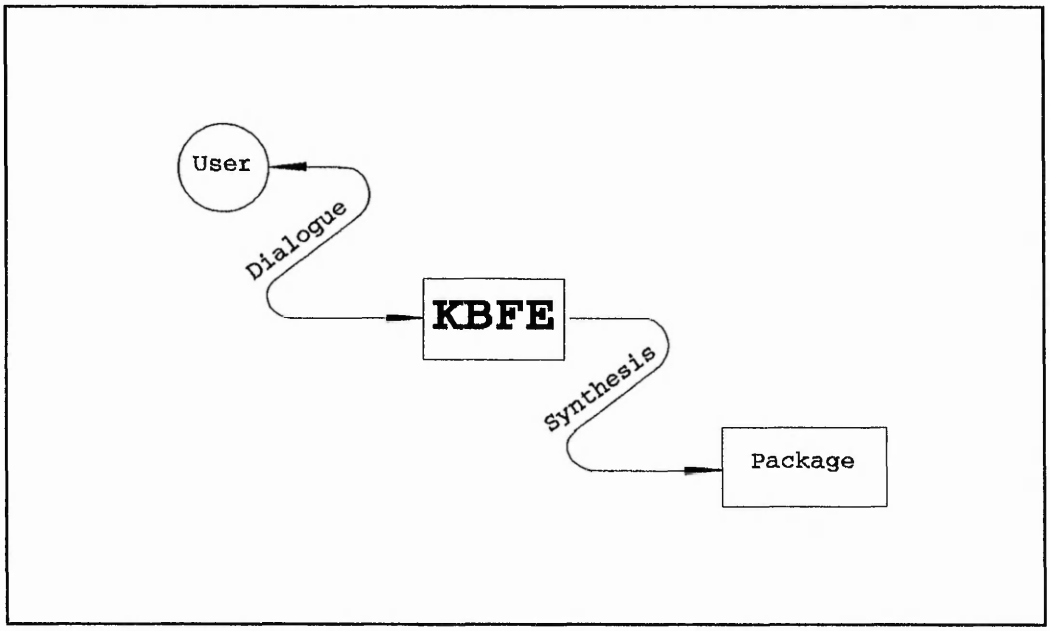


Figure 4.1: Locality and role of a Knowledge Based Front End

A KBFE, as shown in Figure 4.1, is designed to remove the complexities associated with entering a problem specification to a numerical simulation package. Knowledge Based

Front Ends differ significantly from conventional data entry techniques in that they are able to explicitly define a user's problem in the terminology required by the package. This is performed by asking the user questions, structured in English, that allow the IFE to gather the necessary information in order to synthesise the user's problem into the commands required by the target package to fully describe the problem.

The need for KBFEs stems from the fact that conventional software is becoming more and more sophisticated, and as a consequence users are tending to become apprehensive about learning how to use a package. This is essentially what a KBFE endeavours to eliminate. The KBFE concept is such that any user can have access to any package that has a suitable front end to it. KBFEs can be developed for any conventional computer package that requires user input by way of a data file or any other means of using package specific commands, for example: interactive database packages, Mao (1988); interactive control system design and analysis packages, Pang (1988); and engineering packages that are fed with data from auxiliary data files, Thomas et al. (1990).

4.2 Knowledge based front end architectures

Various architectures are presented and described, examples of which are given by Clarke et al. (1988), Tangen and Wretling (1986), Tong (1985), Edmonds and McDaid (1990), and Drechsler et al. (1988). The major objective of all systems is to synthesise the user requirements into package commands. The detailed architecture described by Clarke et al. (1988), for a building energy simulation package, clearly indicates the fundamental requirements of a KBFE which are centred around a BLACKBOARD structure, Hayes-Roth (1983) and Reddy and O'Hare (1991). This orbital representation of modules around a central communications facility allows data transfer between individual modules within the KBFE. Figure 4.2 shows a conceptual architecture that utilises some of the fundamentally important modules proposed by Clarke et al. (1988). The fundamental modules consist of the user model, dialogue handler, package handler, and the knowledge handler. These are supplemented with optional data manipulation routines.

4.2.1 The dialogue handler/user interface

Ramsay (1984) introduces the requirements for dialogue handling in KBFEs. Pertinent points include being able to backtrack when required to amend previously entered

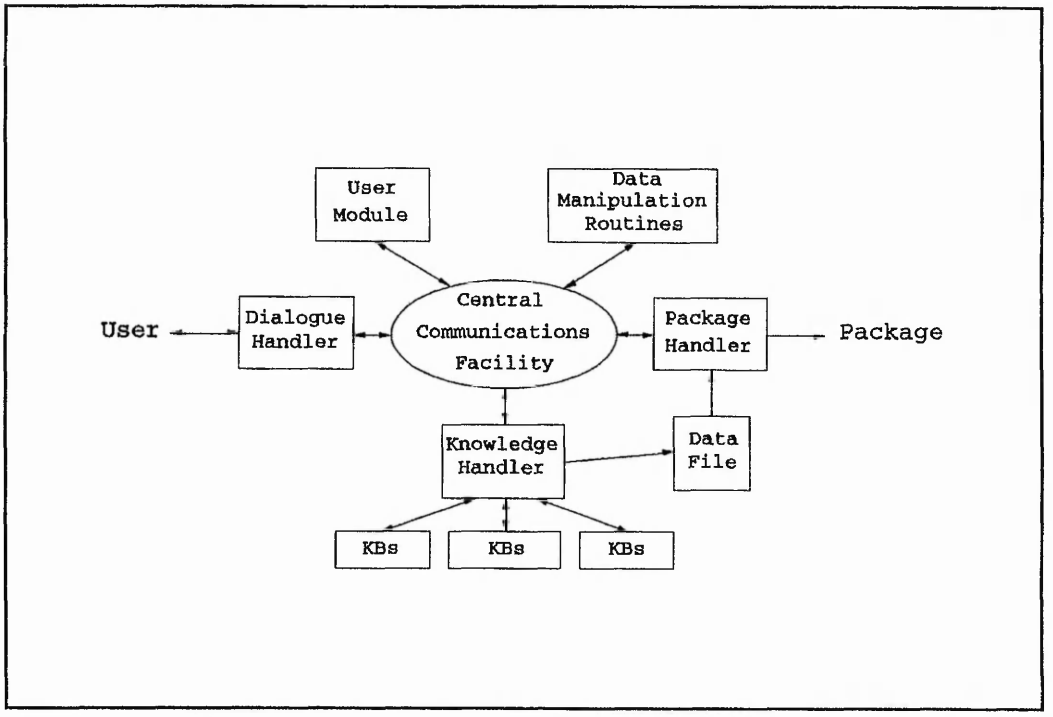


Figure 4.2: Conceptual Knowledge Based Front End architecture

information, being able to distinguish when a user requires superficial help or detailed information relating to a specific task (this should be a function of the user model employed) and appropriate dialogue monitoring.

The dialogue handler would be dependent on the type of user model specified. This is essentially the user interface and controls the sequence of question and answer sessions along with choosing which type of dialogue representation to use, i.e. menus or Natural Language Understanding (NLU). The initial dialogue representation would consist of implementing menus for a KBFE driven system. Research into extending the available dialogue techniques to incorporate NLU would be necessary. ECO, Uschold et al. (1984), used a constrained user driven dialogue handler, i.e. pseudo-NLU. The constraint placed on the system was that predefined command statements were used for the task specification. Even though the commands were English and had specific functions (DOES, USES, SET, UNIFY, SPLIT and DISPLAY) the context still tended to be ambiguous unless their explicit meanings were known. This implied that a manual, or extensive help facilities had to be readily available.

4.2.2 The user model

Benyon (1987) suggests that "We need user models because users are complex systems and we cannot deal with complex systems without models". Examples of these user model attributes include identifiers (name etc.), relative ability to use the package and experience with the software. Jerrams-Smith (1987) extends these requirements of the user model to include behaviour patterns, previous background knowledge, goals and plans.

The overall purpose of the user model is to allow computer systems the ability to maintain a record of the user and to enable him to specify the level at which the dialogue and help routines should be directed. Clarke et al. (1988) allows this module to monitor the interaction sequence taken with the user, thus recording the response speed, number of errors, system default overrides, changes of mind and backtracks.

Ross (1984) indicates that the most common method of producing a user model is to generate 'overlays', or templates, that characterise the behaviour of the user with various postulated models. Information contained within the overlays could relate to the areas identified by Clarke et al. (1988), described above. Jerrams-Smith (1987) also mentions the overlay approach for user models. Pang (1988) simplifies the concept of a user model by constructing three generic categories, (i) the expert level (IFE as a caretaker), (ii) the intermediate level (IFE as an assistant), and (iii) the novice level (IFE as a tutor). Each level reacts to the user differently depending on his self appraisal for using the package, and as such simplifies the developer's need to incorporate extensive routines to consider multiple user models. MacRandal (1987) also simplifies the user model used in the IFE for building design software into two categories: the expert and the novice.

4.2.3 The knowledge handler and knowledge bases

The knowledge handler, usually referred to as the inference engine, operates deductively and selects the relevant knowledge to reach a conclusion by implementing search strategies. The most common search strategies are backward chaining, goal driven, or forward chaining, data driven, with either depth first or breadth first search. Aseo (1988) gives a good diagrammatic representation of the four combinations. The forward chaining mechanism would usually be employed for systems that initially obtain data and then use this to deduce some conclusion. Alternatively, backward chaining would be used for a

diagnostic system whereby a fault is known and the system would reason backwards to identify the cause of the fault. Combining the two strategies has proved to be advantageous for the KBFE to PHOENICS.

Within a knowledge base application the organisation of information into separate knowledge bases is a popular technique because of the ability to categorise the rules. For example, rules for choosing the correct turbulence model, selecting the correct command to specify boundary conditions, etc. Knowledge bases would then be inferred upon by the inference engine in order to deduce appropriate conclusions.

Bundy (1984b) presents a similar architecture for the IFEs which establishes that the 'synthesis' of the task specification, the user definition of the problem, controls all of the modules within the IFE. The synthesis of the task specification is basically the process that transforms the user problem into the coded sequence of commands required by the package. This was described by Bundy as a recursive procedure involving the interaction of preconditions, effects of various methods, task specification and inferencing on information which intelligently bridge any gaps that prevented a user goal to be satisfied. This recursive nature of synthesising a task specification seems to be misleading in that for a given package it is always necessary to specify certain information relating to the task specification. For example, a fluid flow simulation requires that the geometry, fluid properties and boundary conditions should be directly specified by the user. If these parameters are not given then the simulation cannot be performed.

4.3 Knowledge based front ends developed with expert system shells

Prior to commencing the development of any expert system (ES) or KBFE using commercially available ES tools it is common practice to assess the facilities against the specified development requirements. Such requirements might be the use of inheritance, rule-based knowledge representation, deep knowledge representation through the use of mathematical procedures, and the use of frames to store data for use within the rules. It is usually difficult to initially decide on the knowledge representation techniques to use unless prior experience with the specific tool has been attained. Ramirez and Belytschko (1989) describes one of the primary characteristics of expert system development as "incomplete initial program specification". This implies that most expert systems are initially developed as a prototype based on an original estimate of software specifications.

The prototype is then continually revised through a process of incremental development. As the development takes place new problems occur with respect to knowledge refinement which may lead to the decision to implement a different knowledge representation technique. Thus, it is necessary to establish, in the early stages, whether a particular tool will be worth the investment of time and money in the development and improvement of a prototype. Barber (1984) suggests that an expert system shell is an adequate tool for the development of some type of IFEs. However, he states that they "are weak on knowledge representation".

An IFE is defined as "a kind of expert system", consequently it seems reasonable that an expert system shell could be used for its development, in agreement with Barber (1984). This was the approach taken with the development of the prototype KBFE for PHOENICS using the expert system shell LEONARDO.

4.4 Conclusions

It is clear that KBFEs interface the user with some further software package known as the application program. Two initiatives, ALVEY and ESPRIT, essentially address the global issue of Information Technology with the sub areas concerned with IFEs and KBFEs respectively. Conceptually IFEs and KBFEs are the same, and are treated as such throughout this thesis. A KBFE utilises knowledge based techniques to provide a front end to existing software packages and should translate the user's requirements into commands recognised by the application package. Facets used within KBFE architectures include a knowledge handler, dialogue handler, user module and a package handler, as shown in Figure 4.2. Knowledge handlers are usually known as inference engines, that is, they handle the knowledge contained within the knowledge bases, databases, and rules. A dialogue handler controls the communication between the KBFE and the user, in so much as to target the correct level of questioning. This is determined by the information contained within the model created for the user by the KBFE.

The approach taken for the initial development of a KBFE was to implement a commercial expert system shell, LEONARDO.

CHAPTER 5

PROTOTYPE KNOWLEDGE BASED FRONT END USING LEONARDO

5.1 Introduction

This chapter discusses the prototype development of a Knowledge Based Front End using a commercially available expert system shell, LEONARDO. This is preceded by a brief historical look at Expert Systems and how they differ from conventional programming techniques. The chosen shell, LEONARDO, is then reviewed through the knowledge representation formalisms and the problems encountered during its use are presented. Two main features of the KBFE are described: the data file checker and the data file generator. The data file checker is discussed in greater detail primarily because of the techniques that had to be employed in order to develop the software around the limitations of LEONARDO. However, the data file generator is only briefly mentioned because the prototype has been replaced and improved upon through the development using LISP, Chapter 6. The decision to use LISP was taken because of the potential versatility that was experienced with the modular storage of data within the lists provided by LEONARDO.

5.2 Expert systems

Expert Systems (ESs) have emerged from the generic area of Artificial Intelligence (AI). AI is the field of study, associated with computer science, that attempts to duplicate with a machine those activities normally referred to as intelligent when performed by humans. This covers all aspects of human life, as shown in Figure 5.1.

An Expert System is a knowledge based computer program that specialises in performing domain specific professional tasks. These tasks could be classified by novice users as being difficult, whereas the expert would probably regard them as trivial and time consuming. ESs have been shown to be comparable to their human counterparts, Thomas et al. (1988), in terms of accuracy and are generally faster for completing a specific task. The competence exhibited by an ES at performing a given task should always be maintained, but cannot be increased until new knowledge has been acquired and entered into the knowledge bases. It must be emphasised that the system would only work and produce correct results to a problem if the information and heuristics that the knowledge bases contain are themselves correct. The cleché "garbage in, garbage out" equally applies to

ESs as well as conventional computer programs. However, ESs should have the ability of checking the data that is input for validity and accuracy. If there exists any anomalies then the system should automatically indicate that there is a potential error. Conventional programs are just as capable of highlighting errors in data entry procedures by coding in exhaustive checks for the allowed values.

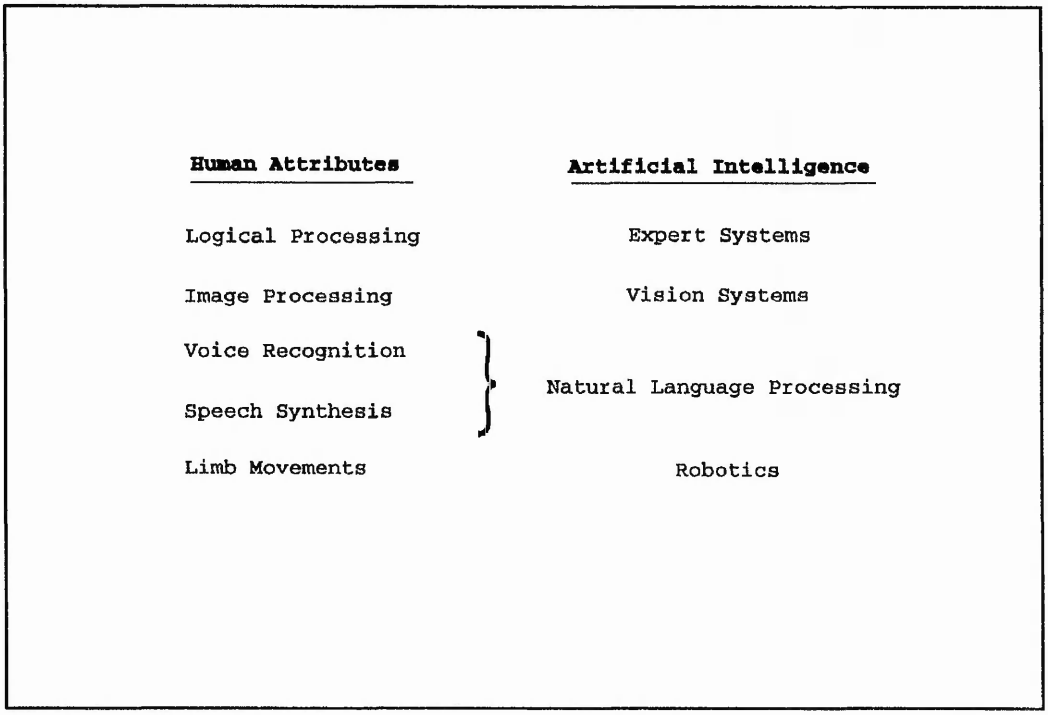


Figure 5.1: Human / Artificial Intelligence attributes

Expert systems are a relatively new method of computer programming. The most immediate distinction between conventional programs and ESs is that the latter is primarily used for symbolic processing, whereas conventional programs are mainly used for numeric processing. Waterman (1986) details how ES differ from conventional programs. The main difference is that an expert system must have expertise, symbolic reasoning, depth and self knowledge. ESs are sometimes known as Knowledge Based Systems (KBSs) and is defined as "a software and hardware system that processes data, information and knowledge", COM(86) 687 final (see references). Knowledge has also been defined as "that part of symbolically described information that is used by competent experts in a particular domain (medical, legal, etc.)".

Symbolic reasoning refers to the ability of the system to manipulate symbols rather than numbers in order to obtain some form of solution. This is again analogous to the expert whereby he chooses symbols to represent the problem concepts and applies rules or heuristics to them. The type of symbolic manipulation employed is directly related to the type of knowledge representation used.

The depth of the knowledge base relates to the extent of the information contained within the system. That is, after development, would the system be able to resolve problems in the real world? For this to be possible the knowledge within the knowledge base should be as exhaustive as is feasibly possible.

Self knowledge relates to the system being able to recall the inferencing process that was performed and to be able to explain its line of reasoning to the user. This is achieved by the generation of inference chains during the interaction with the user and progression through the inference networks. The utilisation of these inference chains within rules relating to the accuracy, consistency and plausibility of its conclusions would have three effects, Waterman (1986): Users would tend to have more faith in the results, hence more confidence in the system; Assumptions made by the system would become explicit to the user as opposed to being implied; and easier to predict and test the effects of various changes within the system. Knowledge used for the assessment of the conclusions is called meta-knowledge, or knowledge about knowledge.

The expertise that a system must have is usually coded in as rules and facts. Skilful implementation of the rules within a robust structure is the secret of a good expert system. This implies that the system should be able to apply its knowledge effectively and efficiently, thus performing as an expert would under real situations.

There exists a growing number of tools available for the development of expert systems. These tools, or shells, provide various knowledge representation formalisms including frames, lists, rules and procedures. The utilisation of a commercial expert system shell, LEONARDO, was the initial approach taken for the development of the prototype for the KBFE to PHOENICS.

5.3 LEONARDO

Expert systems usually contain logical symbolic processing as well as conventional computational techniques. They can be written using standard languages like FORTRAN or PASCAL but are some what cumbersome and contain extra embedded knowledge, Alty and Coombs (1984). This makes subsequent modifications to the control structure difficult without rewriting the code. On the other hand a symbolic reasoning approach which uses an inference engine with either backward or forward chaining automatically considers new rules. Recent development of ES shells have adopted the latter approach and are much easier to use because the KB can be easily modified. Several commercially available ES shells were considered, but the ultimate restrictions of cost, *potential versatility* and availability made LEONARDO the most favourable in this case.

LEONARDO was marketed by Creative Logic as a complete system with all the tools necessary to design, develop, test and deliver expert systems. LEONARDO enabled the developer to create a knowledge base using conventional production rules and quantification rules. Both types of rule create objects which allowed the storage of data within slots in object frames. Production rules are simple **IF condition THEN action** rules, whereas quantification rules are used with class objects to facilitate inheritance. Each object frame provides the basis for all available knowledge representation formalisms. Deep knowledge could be coded in the LEONARDO procedural programming language, within the frame of the appropriate object. Modularised RuleSets could be created by locating the rules for a particular object within the object frame under the slot **RuleSet:**. The inference, by default, was backward chaining with opportunistic forward chaining.

5.3.1 Spurious events within LEONARDO

Given the specification of LEONARDO, a novice to the field of ESs would seem very impressed. Forsyth (1988) reviewed LEONARDO version 3.00, level 3, after its release and "was unable to crash the system in three days' usage, and formed the overall impression that LEONARDO is a robust piece of software." However, after eighteen months' use, through which the shell was subjected to a series of rigorous tests, the initial impressions soon diminished as the software limitations became apparent. Possibly the main problem that became obvious was the number of bugs that were present within the software. These, it can be assumed, only became apparent because of the extent of the

application to which the shell was being applied. Other spurious events that were experienced consisted of unexpected instantiation of objects during the execution of a knowledge base; assignment of nonsensical ASCII characters to real, text and list objects; spontaneous deletion of all objects and creation of large quantities of the same object. Furthermore, the procedures, if too large, spontaneously became corrupt, a similar experience occurred if the lists were allowed to become too long. These were but a few of the problems that were encountered during prototype development.

5.3.2 LEONARDO's use of pseudo-lists

True lists or linked lists, Schildt (1990), have one distinct advantage over arrays: the initial size does not have to be specified, however, as with arrays their size is memory dependent. This allows dynamic creation and deletion of the values within the lists. That is, the actual size can vary during run time, and the memory requirements vary accordingly. The structure of a linked list permits the insertion and deletion of values quickly and easily. This is possible because each element of information carries with it a link to the next data item in the chain. Thus any type of data item can be located in any position within the list. Figure 5.2 conceptually shows the structure of doubly linked lists. The list structure that Creative Logic developed is not strictly a set of true lists, as described above, but can be more accurately described as pseudo-lists. LEONARDO is written in FORTRAN77, Forsyth (1988), and as such cannot implement true lists. To enable LEONARDO to imitate lists Creative Logic utilised CHARACTER*1200 strings.

LEONARDO interprets a knowledge base and for a list object a CHARACTER*1200 variable was assigned to represent the object in compiled FORTRAN77 code. This did not allow numeric values to be stored in the lists, unlike LISP. This essentially confirms Ramirez and Belytschko's opinion that applications have to be written around the tool's limitations. In order to store real or integer values it was necessary to transform the number into a string of characters depicted by the appropriate ASCII numbers and then assign the text to the list. This decomposition of a number was performed by evaluating the power of the number in standard form and then decomposing the number. This essentially produced an exponential form of the number as a string of characters. The following sequence indicates the steps taken for the transformation of the number 3479.7 into its appropriate string:-

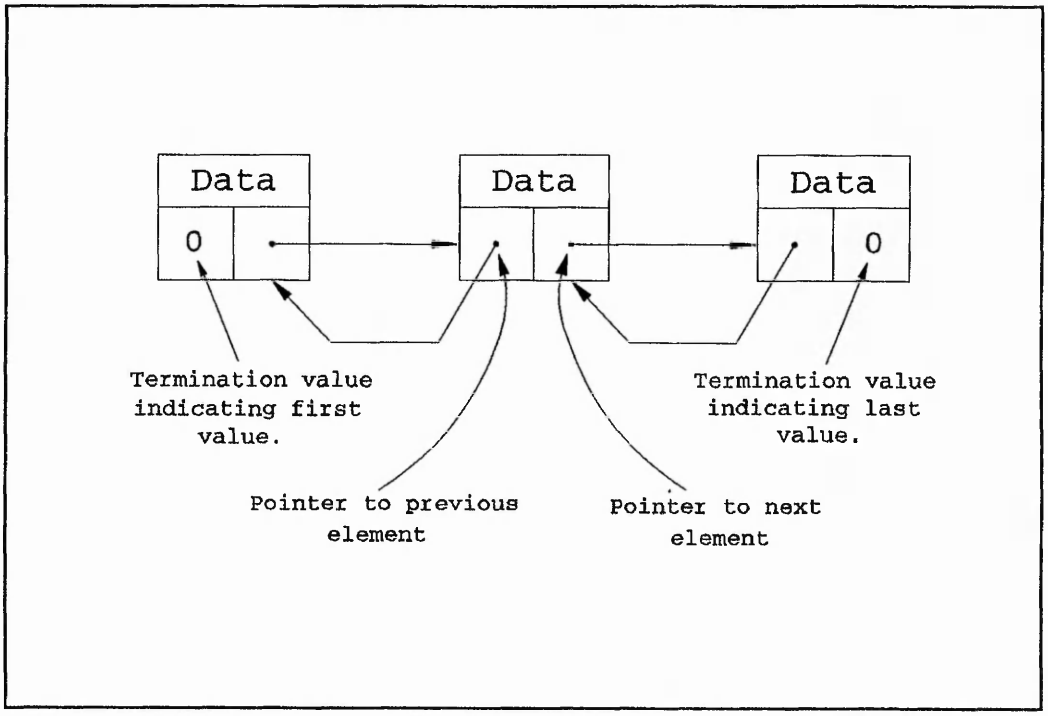


Figure 5.2: Conceptual structure of doubly linked lists

(a) Find the exponential power required to obtain the number in standard form. In this case the power is 3 and the standard form of the number, omitting the exponential part of the expression, is 3.4797

(b) Divide the number given, i.e. 3479.7, by 10^{Power} to obtain a number between 1 and 10. The number becomes 3.4797

(c) Take the integer of the number and transform it into an ASCII number by the addition of 48. This gives the ASCII number of the character required by the string of text. For example the ASCII number for the integer number 3 is obtained by 3+48 = 51, and the ASCII number 51 represents the character '3'.

(d) Concatenate the string of text with the newly obtained character. The original string of text, prior to commencing the process, was ''.

(e) Modify the original number by subtracting from it the value of the integer multiplied by 10^{Power}. For example the original number becomes 479.7 = 3479.7 - 3 * 10^3.

(f) Repeat from steps (b) to (e) until the original number has been reduced to zero.

(g) Insertion of decimal points should be performed as and when necessary. Finally the exponential character 'E' is attached to the string along with the original power determined in (a) above.

After the numeric / text transformation has been completed the example string would be '3.4797E3'. A procedure to perform the transformation was developed so that all numeric values could be inserted into the necessary LEONARDO pseudo-list.

Simple concatenation of text values, or lists, to lists within LEONARDO could be performed using different commands in the rule base and procedural languages. However, insertion of values within lists was not possible, and as such warranted the development of procedures to perform the addition and deletion of values within list objects.

5.3.3 LEONARDO's run time response

When developing an ES of any type, be it for design optimisation or to act as an IFE, then the response is important because a user would not expect to sit in front of a computer that takes forever to perform a specific task. However, it must be emphasised that the response times for various computers differ depending on their configuration. For instance, there would be a significant variation on performing the same task on a PC 386, 100% IBM compatible compared with a workstation. For a networked mainframe computer, the response times would depend upon the load on the computer, i.e. the number of users logged onto the system. The response times on a PC 286 when developing and running the prototype IFE for PHOENICS was found to be a major problem. Excessive disk accessing times by the system increased processing times beyond what was considered to be reasonable. Constant hard disk accessing was required to load each ruleset and procedure into memory as and when it was called. Thus, if a large number of procedures were continually called then constant disk accessing would be required. Procedures within LEONARDO were 'recursive', how this was coded with

FORTRAN77 seems puzzling, however, implementing recursion proved to be tremendously slow and as such it was decided to utilise external procedures to perform complex calculations. Extending the list processing facilities within LEONARDO by introducing procedures to perform certain tasks, as described above, exacerbated the response times. To further compound the problem, external procedures which could be executed from within LEONARDO took considerably more time to produce results compared with DOS execution. For example, the mathematical parsing FORTRAN code, see section 5.4.3, when called from within LEONARDO to calculate the expression **2+(26.47/49)^3** required 15.25 seconds. This is in contrast with an execution time of 1.51 seconds when run directly in DOS, based on the same expression. This shows that the execution from within LEONARDO is approximately ten times slower than that in DOS.

5.3.4 Compilation times and debugging facilities

Within a knowledge base it is always necessary, as with conventional programs, to compile and debug modified code. LEONARDO permitted the compilation of individual rulesets or procedures. This initially proved to be beneficial because there seemed little point in compiling a complete set of rules and procedures if only one had been modified. However, after continual compilation of individual rulesets and/or procedures total corruption of the knowledge base was experienced. In order to retrieve the entire KB a complete compilation was necessary. This proved to be another annoying problem.

A trace facility provided for the user allowed the inferencing process to be followed within the rulesets. However, this facility was not available for procedures, this proved to be highly frustrating and inconvenient. Through private communications with Creative Logic, it was established that there was a procedural debugging facility, but it was not commercially available. Therefore, in order to perform effective debugging of procedures, that were occasionally complex, it was necessary to insert appropriate diagnostics.

5.4 Prototype infrastructure

The prototype IFE was developed using LEONARDO (versions 3.17, 3.18 and 3.20) on an IBM compatible 286 PC AT. The initial stages of development saw rapid progress towards a working system. Unfortunately, the rate of system growth started to rapidly decline as the software limitations became apparent, and the need to develop the system

around these limitations increased.

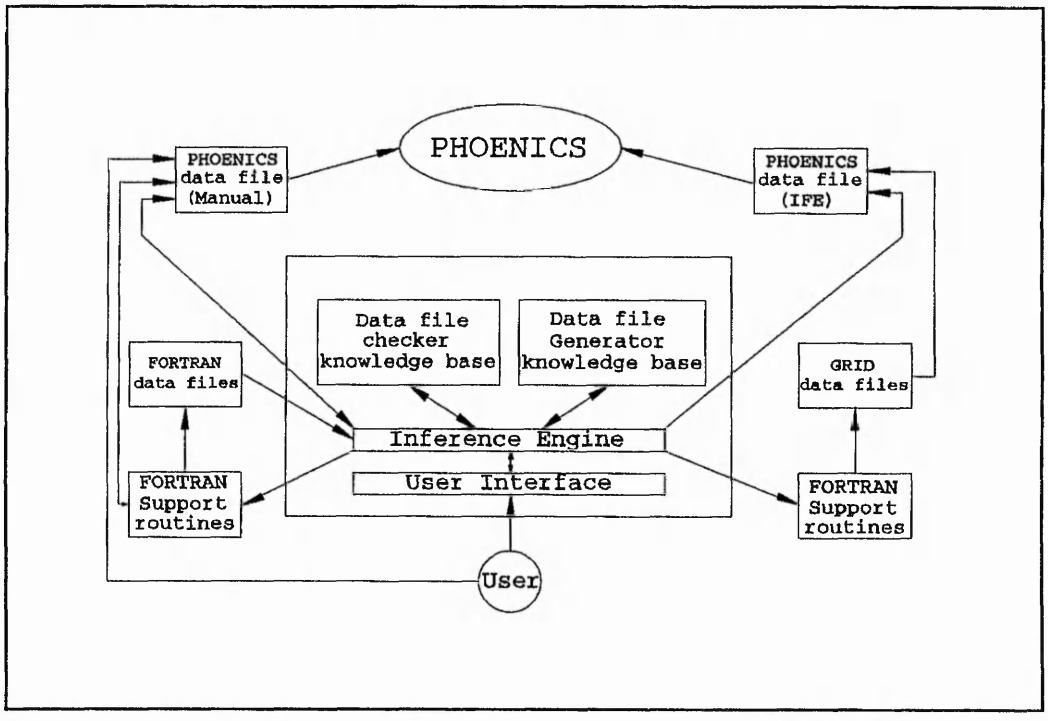


Figure 5.3: Preliminary architecture of the PHOENICS KBFE developed within the LEONARDO shell

Figure 5.3 shows the initial infrastructure on which the development was based. As can be seen there exists external FORTRAN code that is continually accessed to increase the speed and flexibility of the KBFE. It became apparent during the early stages of the development that LEONARDO could not perform rapid complex calculations, that is the calculations could be performed but at the expense of response times. As a consequence of this it was necessary to develop FORTRAN code to perform the grid generation and mathematical parsing.

The primary reason for developing a KBFE was to enable novice users of CFD to become familiar with the techniques employed to model fluid flow problems using a commercial software package. To this end it was important to integrate into the system, knowledge relating to the synthesis of the user's problem definition to appropriate PHOENICS commands. This can be seen to be the fundamental requirements placed upon an KBFE,

and as such would consist of generating a usable data file from an interactive session with the user. This approach was suggested within the feasibility study, Uzel et al. (1988). However, it was thought prudent to also allow partially experienced users the ability to have their manually created data files checked prior to submitting them for analysis. This facility would mimic the process of asking the advice of an expert who would indicate any errors with the data and recommend possible improvements. Certain mistakes, for example the inadvertent transposition of arguments within commands, have been shown to be accepted by PHOENICS, thus indicating an acceptable data file, but have lead to erroneous results. Errors such as these can take hours to find if a large data file has been submitted. In order to eliminate the tedious task of checking the independently generated data file manually, thus reducing the time involved, a prototype system, the data file checker, was developed that would examine the contents of the file and would assess the validity of the commands. This would upgrade the existing facility within the PHOENICS preprocessor, which simply states that an error occurs on one or more lines, to a higher level whereby detailed information regarding the invalid statements would be displayed.

The knowledge for the KBFE was obtained from three different sources. Firstly, practical experience with PHOENICS as a user. The commands that have to be used to correctly model a CFD problem are explicitly defined within the PHOENICS reference manual, TR200 (1989). Experience with CFD concepts, Patankar (1980), is important because PHOENICS appears to assume that the user has knowledge of various techniques used for the discretisation process. This was thought to be possibly the most important method of understanding the operation of PHOENICS, since there is no substitute for experience. The second method was through directly conversing with experienced users, and extracting their knowledge on problem specifications. Finally, by acting as a pseudo-expert when supervising and advising inexperienced users. Knowledge acquired in this manner was transformed into various rules which formed the infrastructure of the KBS.

5.4.1 The data file checker

The data file checker was developed to allow partially experienced users the ability to check manually created files prior to their submission to PHOENICS, Hartle et al. (1993). The structure of the checker can be seen in Figure 5.4. Several problems arose during the early stages, these were :-

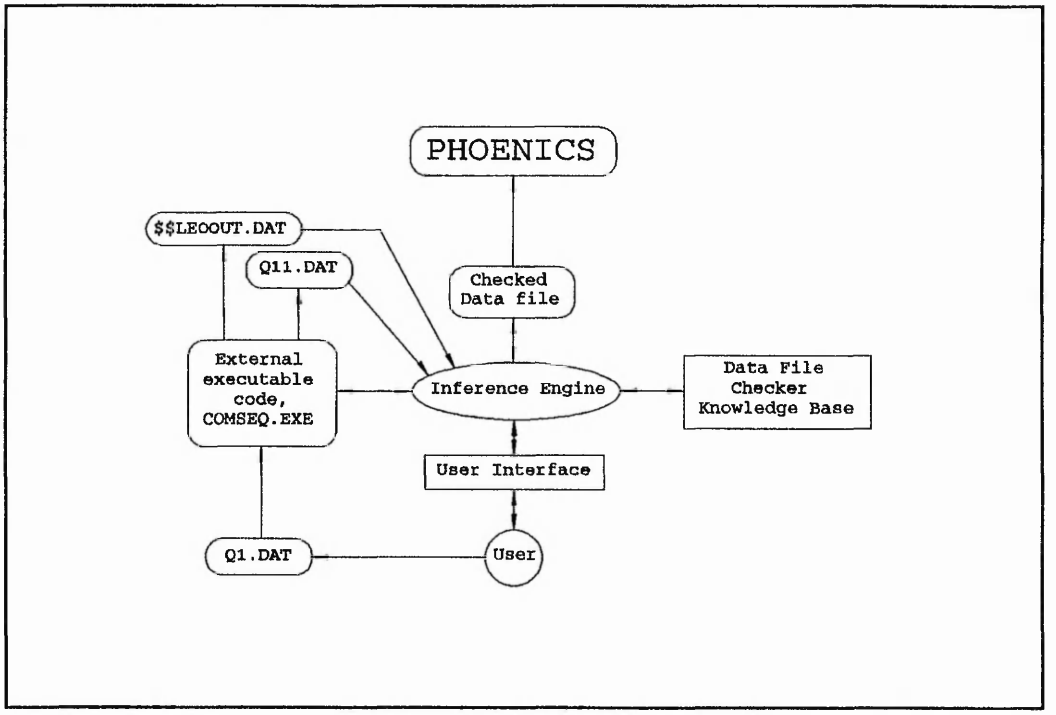


Figure 5.4: Detailed architecture for the data file checker

(1) PHOENICS could generally accept any valid command in any order, with exceptions, which could be referenced to any other command. The position of the referencing command is totally arbitrary. For example, the word "TEMP" (Figure 5.5) appears as an argument within the command statement COVAL but is assigned as the name of H1 on the following line. This would present a problem if sequential checking by the IFE was implemented, because PHOENICS requires an independent variable or an assigned name as the second argument within the COVAL statement. Therefore, the order in which the commands are checked should be predefined prior to activating the KBFE. Even though this layout of the Q1.DAT file could be accepted, there exists a recommended structure in which to define the commands, see section 3.5.6.

(2) Commands within PHOENICS usually necessitate numeric values for, say, defining specific boundary conditions, for example, a wall is t be held at a constant temperature of 100 °C, or an inlet boundary is to have a mass influx of 1.235 kg s^{-1} m^{-2}. However, these boundaries could well be described using expressions involving

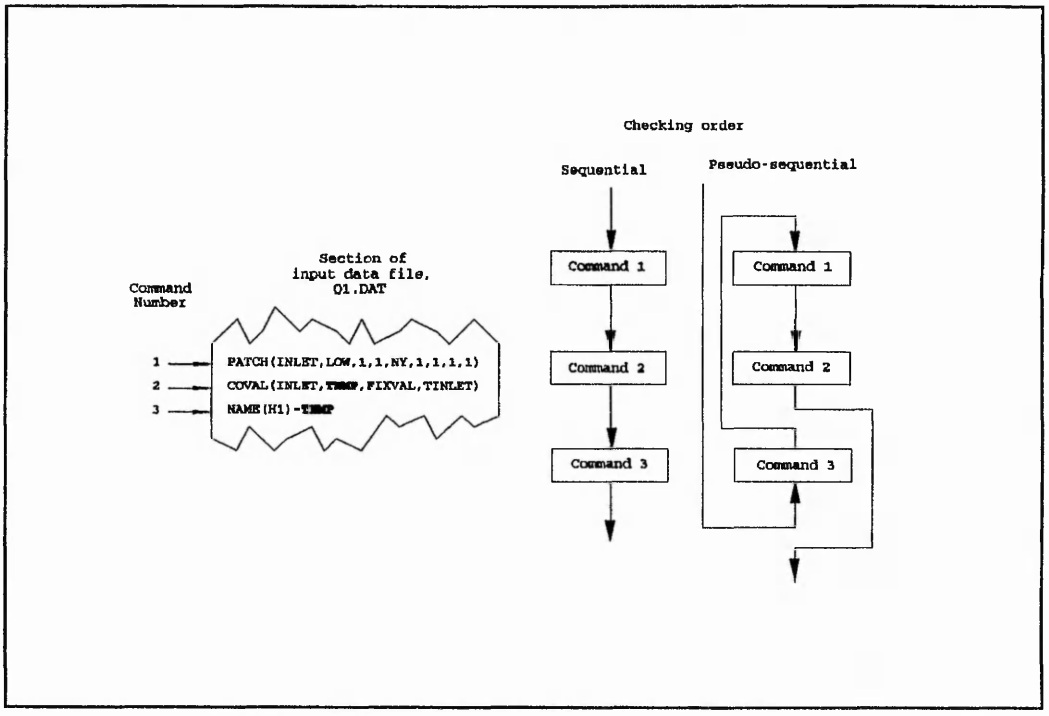


Figure 5.5: Possible order of data entries into the data file with sequential and pseudo-sequential checking orders

previously defined variables, in this case TWALL could be the wall temperature. Similarly, the mass influx could be described as RHO1*WIN, where the values of RHO1 and WIN would have been previously declared. To this end, the program would inevitably fail if, it instead of being given a numerical variable, it was given a character string as an expression in its place.

In order to avoid these problems a pseudo-sequential checking procedure and a mathematical parser were developed.

5.4.2 Pseudo-sequential checking

The purpose of the pseudo-sequential checking procedure is to initially read the data file, Q1.DAT, and generate a COMmand SEQuence, COMSEQ, which would be used by the data file checker KB. External FORTRAN code, COMSEQ.FOR shown in Appendix C, defines the order in which the commands are to be read by the KB. This establishes a

pseudo-sequential checking order, Jambunathan et al. (1991a). The pseudo-sequential checking order is read directly into LEONARDO, in a pseudo-list format, through its internal information passing files $$LEOINP.DAT and $$LEOOUT.DAT. The information that is contained within the list can be seen in Figure 5.6, and is structured in a predefined order.

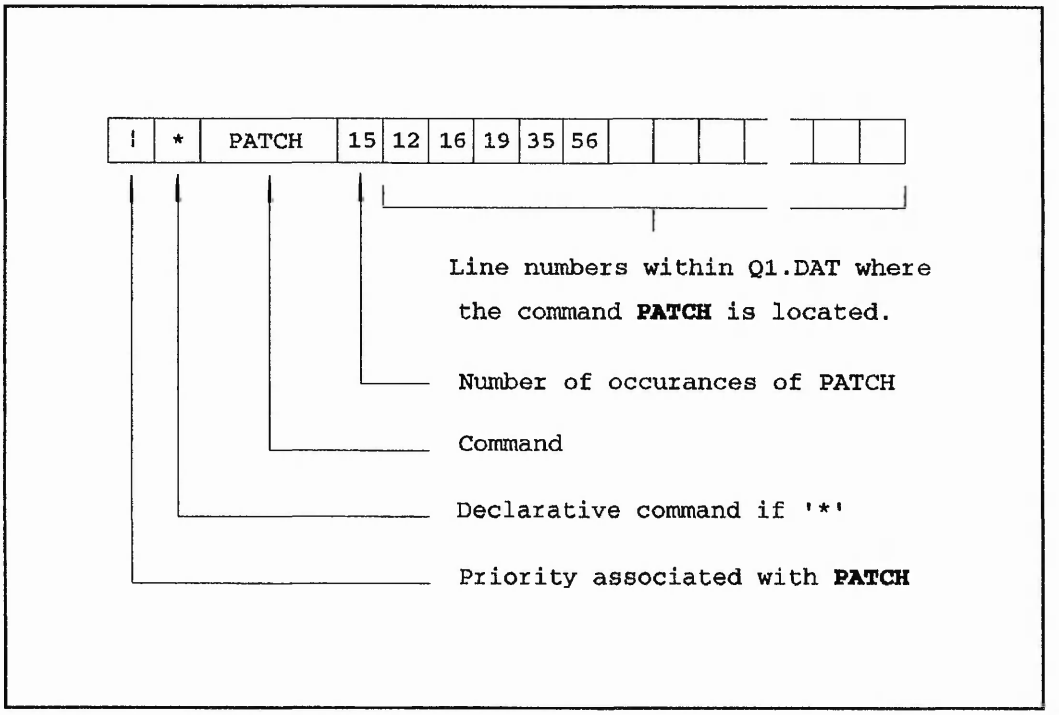


Figure 5.6: Elemental structure for the COMSEQ one dimensional array for pseudo-sequential checking information

COMSEQ is a FORTRAN based code that works with a CHARACTER*125 array which is one dimensional with a maximum of 500 elements. The array stores a total of 377 possible PHOENICS (version 1.4) commands. The remaining 123 locations provides adequate memory for user defined variables. The array of statements forms the pseudo-sequential checking order after the Q1.DAT file has been read, and the priority of the commands have been sorted. The two main operations performed by COMSEQ are (i) the reading and modification of the array information, and (ii) the sorting of the commands into a prioritised list. The basic structure of the information within the array is given in Figure 5.6.

The sequence of reading Q1.DAT, and generating the pseudo-sequential checking order is as follows :-

(a) Read the current line in the data file

(b) Write the line to an auxiliary file, Q11.DAT, which will be used for direct access by the KBFE. Direct access reading within LEONARDO required that all records within a file have the same record size, this was fixed at 75 characters.

(c) Check the line for one of the PHOENICS commands.

(d) If it is a command other than 'REAL' or 'INTEGER' then append the line number to the appropriate character string in the one dimensional array. The line numbers are delimited with commas.

(e) If the command is 'REAL' or 'INTEGER' then append to the end of the character array all of the declared variables within the PIL command. For example, REAL(WIN1,WIN2,WIN3) will append to the array the dynamic commands WIN1, WIN2, and WIN3. Once these are declared within Q1.DAT they will take the form of the structure shown in Figure 5.6.

(f) The data statements within COMSEQ.FOR, shown in Appendix C, contain predefined settings according to the ability of one command referencing other commands. These priorities are based on the decimal equivalent of the ASCII numbers. For example, '|', which is ASCII 124, has the highest priority, and are assigned to user declared variables. A blank in the first position has the lowest priority with the ASCII equivalent of 32.

(g) A general bubble sorting routine shifts all of the commands that have been used into the top 'n' elements of the array, where 'n' is the number of variables and commands that have been defined within the data file.

(h) A printing routine formats the output from COMSEQ into one continuous string and writes the entire contents to $$LEOOUT.DAT, which is read by LEONARDO into a list object. Figure 5.7 shows a section of the one dimensional array in

```
*NY,1,7,
*YVLAST,1,8,
*NZ,1,24,
*ZWLAST,1,25,
*ENUL,1,62,
*RHO1,1,63,
YFRAC,15,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
ZFRAC,36,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,
47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,
TALK,1,1,
TEXT,1,2,
CARTES,1,3,
SOLUTN,3,4,5,6,
PATCH,7,64,67,69,72,75,78,81,
COVAL,13,65,66,68,70,71,73,74,76,77,79,80,82,83,
CONPOR,1,84,
LSWEEP,1,85,
NPLT,1,86,
RESREF,3,87,88,89,
RELAX,3,90,91,92,
OUTPUT,3,93,94,95,
STOP,1,96,
```

(a)

```
'*NY,1,7,*YVLAST,1,8,*NZ,1,24,*ZWLAST,1,25,*ENUL,1,62,*RHO1,1,63,YFRAC,15,
9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,ZFRAC,36,26,27,28,29,30,31,32,
33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,
57,58,59,60,61,TALK,1,1,TEXT,1,2,CARTES,1,3,SOLUTN,3,4,5,6,PATCH,7,6,4,67,
69,72,75,78,81,COVAL,13,65,66,68,70,71,73,74,76,77,79,80,82,83,CONPOR,1,84,
LSWEEP,1,85,NPLT,1,86,RESREF,3,87,88,89,RELAX,3,90,91,92,OUTPUT,3,93,94,95,
STOP,1,96'
```

(b)

Figure 5.7: (a) COMSEQ one dimensional array, (b) Pseudo-sequential checking order used for a manual data file

COMSEQ and the output string in $$LEOOUT.DAT.

User defined declarative commands, such as WIN=1.234, are read pseudo-sequentially and stored internally within the KBFE for use at a later stage. The reason for storing the commands and their respective values is that they may be referenced by subsequent statements, such as ....

COVAL(INLET,P1,FIXFLU,RHO1*WIN1).

This, when read by the KBFE, would expect previously defined numeric values for RHO1 and WIN in order to calculate RHO1*WIN. The values for RHO1 and WIN are stored in a list structure of the form ...

$$Variable_1, Value_1, Variable_2, Value_2, ..., Variable_n, Value_n$$

... and when an expression is required to be evaluated the contents are used, in conjunction with a mathematical parser, to calculate the expression. The values used for resolving such expressions should have been instantiated by means of the priority settings previously discussed. Once the expression has been evaluated the result is substituted back into the command.

5.4.3 Parsing of mathematical expressions

Parsing with respect to AI and expert systems usually refers to analysing natural language. Clocksin and Mellish (1984) introduce the concept of parsing using PROLOG grammar rules to study the structure of an English sentence. However, the main thrust of the problem that we are concerned with does not include parsing English sentences, but mathematical expressions.

Parsing of mathematical expressions was highlighted as being an extremely important facet of the KBFE. LEONARDO was used to stage the first development of the parser, whereby recursive procedures were employed. This proved to be successful but was extremely slow. To improve the response of the code a FORTRAN parser was developed, EVALUATE.FOR, as shown in Appendix D.

Essentially the mathematical parser reduces an expression into the fundamental components of operators and operands, and then proceeds to determine their values. The dissection of an expression involves delimiting operators and operands within the expression. Assume an expression to be made up of the following ...

NX/2+1

with the variable-value list containing ...

NX,400,NY,200,WIN,23.7

The expression and variable-value list are passed to $$LEOINP.DAT as character strings and upon execution of EVALUATE.EXE, they are read from the transition file. As a result of LEONARDO's use of pseudo-lists, similar list manipulation functions as those within LEONARDO were developed for EVALUATE.FOR. Dissecting and delimiting

the expression with commas, leads to ...

NX,/,2,+,1

Substitution for the variables is performed by removing the appropriate value from the variable-value string and performing the necessary insertion. The expression then becomes ...

400,/,2,+,1

Precedence rules, Kernighan and Ritchie (1988), are applied to determine the order of the calculations, and for the given example the substitution and reduction leads to ...

200,+,1

201

Once there exists no operators within the expression the result has been obtained. If parentheses were in the original expression then the order of precedence moves to the inner most set of parentheses, where the sub-expression is resolved using the conventional operator precedence.

During the checking process errors or omissions within the Q1.DAT file are registered in a list structure that details the command, the line number, and the type of error associated with the line. The latter information is coded into the system by using error numbers that can be assigned to strings of text. The error list is used upon the completion of the initial check when the appropriate error messages are displayed. At this point the user is able to modify the entry into the data file interactively with the checker or he can exit from the system and modify the results independently. After the error messages have been displayed the system creates a summary of the analysis relating to the supplied information and presents this to the user. The user can easily determine whether he has made any obvious omissions that the system is unable to detect by reading the analysis summary.

5.4.4 The data file generator

Most numerical simulation/analysis packages utilise input data files for defining the problem to be analysed. The usual information that they contain refers to the geometry, material/medium properties, boundary conditions and possible solution parameters. The most time consuming task that one must complete prior to becoming proficient with any numerical simulation package concerns familiarisation of the semantics, syntax and structure of the language used within the data file. This problem can be exacerbated if the fundamentals behind the theory need to be appreciated in order to aid the learning process. This combination of becoming familiar with the language and the fundamentals behind the theory is essential if a moderate understanding in the usage of PHOENICS is to be attained. The concepts that really need to be appreciated concern discretisation methods (the requirements, techniques, limitations and implications), the reasoning behind the method of assigning boundary conditions, and the control of the solution algorithm which is heuristic in nature.

The necessity to become aware of fundamental concepts, as required by PHOENICS, can be diminished if a KBFE were available to aid the user, allowing him to concentrate on describing the problem to be analysed. That is, by describing the geometry, specifying the necessary boundary conditions, and requesting specific output requirements. The task of defining the grid, monitoring and controlling the solution algorithm, specifying the necessary commands and submitting the job should be completed by the KBFE. This leads to a convenient modularisation of the KBFE, as shown in Figure 5.8.

The analysis definition details the basic preliminary information regarding the type of analysis to be performed. This consists of the number of dimensions required, type of coordinate system (cartesian or cylindrical), extreme axis dimensions, number of inlets, number of outlets, number of domain walls and the number of obstructions. The specification of the geometry relates to the initial information and essentially consisted of a combination of inlets, outlets and walls. The decomposition of a geometry into these three items leads to a dynamic list structure for representing the geometric properties and defined boundary conditions, to be discussed in section 5.4.5. Preliminary grid generation, which utilised the stored geometric and boundary condition data, was performed using external FORTRAN code. The FORTRAN routines progressively generated the mesh for regionalised sections of the geometry using a power law relationship, and was a function of

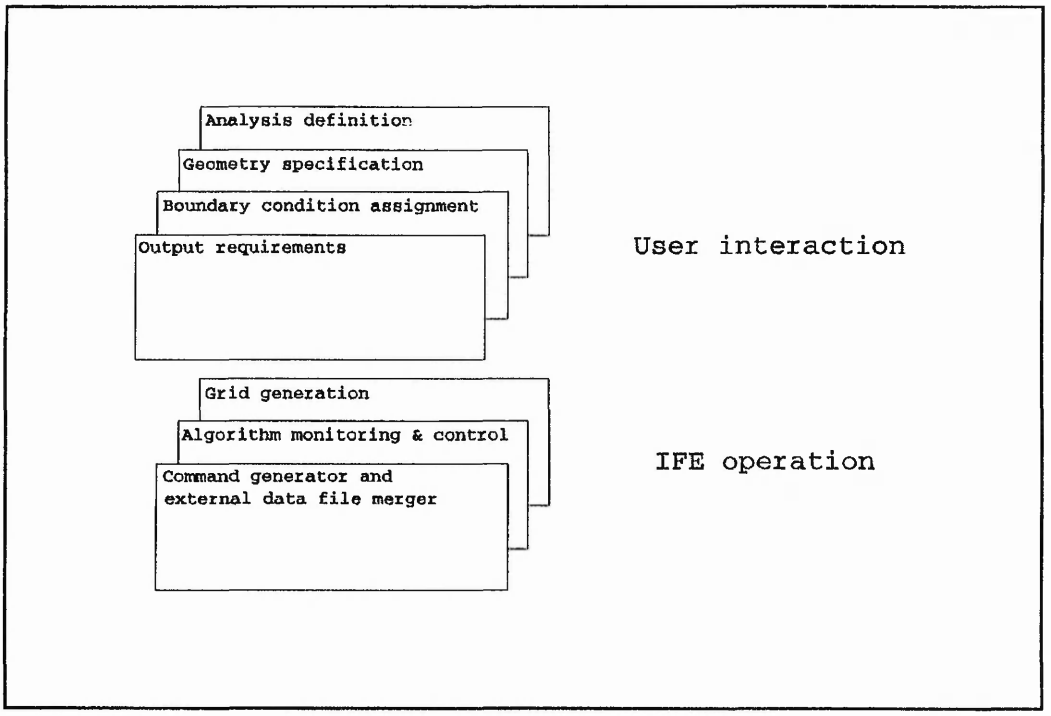


Figure 5.8: Modularity of the Q1.DAT data file generator developed with LEONARDO

the minimum cell size and maximum allowed aspect ratio. Furthermore, they created the necessary commands to fully describe the geometry and boundary conditions using the appropriate PHOENICS commands. The final grid generation technique, discussed in section 3.8, superseded the procedures developed for the prototype KBFE.

5.4.5 Information storage within pseudo-list structures

During the initial stages of the data entry procedures for the analysis definition, geometry specification and boundary conditions the information was stored within pseudo-list structures. The lists have an identical structure for each type of boundary, i.e. walls, inlets and outlets. The stored information consists of the name of the boundary, the patch type associated with the geometry, the coordinates required to totally describe the boundary, the priority of the boundary for the grid generation routines, the dependent variables specified at the boundary and the appropriate arguments required by the PHOENICS COVAL statement for boundary condition specifications. The conceptual structure for the lists is shown in Figure 5.9, and it simulates lists within lists as used within LISP for data

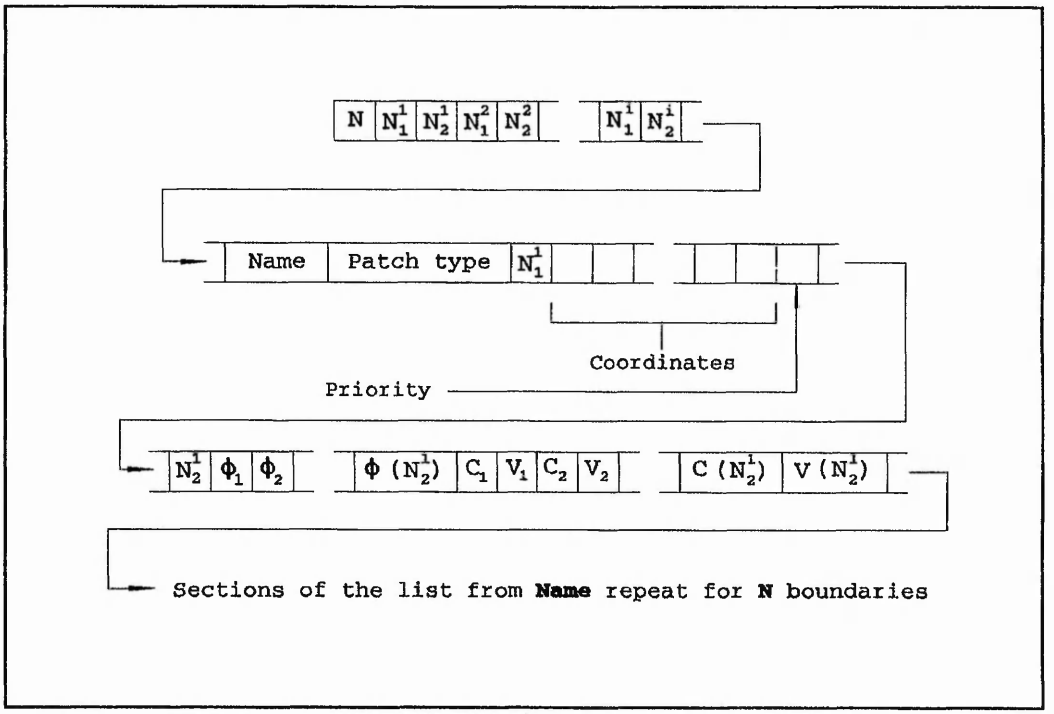


Figure 5.9: Conceptual, complex, list structure within LEONARDO

storage. The individual modules, the lists within the main list, are accessed by evaluating the position of the required data from pre-determined equations. Individual sets of equations are created for specific list structures. Equations **(5.1)** to **(5.9)** were used for the location of specific data from the conceptual list structure shown in Figure 5.9. A feasibility study of incorporating the mathematical parser and information storage using pseudo-lists was performed, Jambunathan et al. (1991b). However, because of the inherent slowness experienced with LEONARDO and the potential to exceed the maximum number of allowed characters within the list structures, it was decided not to implement the technique.

$$N_1^i = 2 + 2(i - 1) \tag{5.1}$$

$$N_2^1 = N_1^i + 1 \tag{5.2}$$

$$Index_i = 2 + 2N + \sum_{n=1}^{i-1}\left[4 + Min(1,N_2^n) + 3(N_1^n + N_2^n)\right] \quad (5.3)$$

$$Name_i = Index_i \quad (5.4)$$

$$Patch\ type_i = Index_i + 1 \quad (5.5)$$

$$Priority_i = Index_i + 3 + 3N_1^i \quad (5.6)$$

$$\phi_{ji} = Priority_i + 1 + j \quad (5.7)$$

$$C_{ji} = Priority_i + 2 + N_2^i + 2(j - 1) \quad (5.8)$$

$$V_{ji} = C_{ji} + 1 \quad (5.9)$$

5.5 Conclusions

The performance of the expert system shell, LEONARDO, for knowledge representation, inferencing and data storage when applied to the development of a prototype KBFE for PHOENICS was assessed. Problems were experienced, such as the spontaneous corruption of knowledge bases, poor data storage facilities and the use of pseudo-lists. Re-emphasising the points made by Ramirez and Belytschko (1989) and Barber (1984), expert system shells are generally restrictive in terms of knowledge representation techniques and it is usual for the application to be written around the chosen shell.

The process of boundary condition assignment and data storage within pseudo-lists, whereby access to specific data items required relatively complex indexing equations, seemed rather cumbersome and as such new techniques for representing such knowledge has been developed within LISP. The mathematical parser proved to be invaluable for the data file checker.

LEONARDO was deficient in some of the modules shown in Figure 4.2. For example, the intrinsic dialogue handler only reliably permitted menus to be used which forced a system controlled KBFE. There was no central communications facility which meant that variables to be used within procedures required either global definition or explicit

transportation into the appropriate routines. User modelling routines were unavailable unless specifically created and package handling procedures needed to be developed. LEONARDO proved to be inadequate for the prototype development because of weak knowledge representation facilities. However, the experienced gained through the use of the expert system shell proved to be invaluable.

The inherent slowness, in conjunction with other factors forced the decision to abandon LEONARDO and to commence further development using a different approach. The experience gained with using LEONARDO proved to be beneficial in so much as to create a foundation upon which to develop the LISP KBFE.

LEONARDO's knowledge representation formalisms created a framework within which to base the usage of lists and object data storage. These approaches were carried forward into the LISP development and have proved to be extremely effective.

CHAPTER 6

A KNOWLEDGE-BASED FRONT END TO PHOENICS USING LISP

6.1 Introduction

This chapter describes the development of a Knowledge-Based Front End for PHOENICS using a traditional Artificial Intelligence language, LISP. References that were found to be excellent for the newcomer to LISP included Winston and Horn (1989), Steele (1990), and Yuasa and Hagiya (1986).

Winston and Horn (1989) have dedicated two chapters to inferencing using forward and backward chaining and one other to pattern matching. The concepts presented for knowledge management formed the basis of the inferencing mechanisms developed for the KBFE, also the pattern matching and unification techniques were utilised for the inferencing processes. Furthermore, a method of representing rules and filtering assertions through the antecedents, thereby resulting in a bindings list containing data for use within the consequents to assert further information is described. Assertions were found to be extremely beneficial to the storage of CFD data for boundary conditions.

A central communications facility was developed that contained data relating to the boundary conditions and geometrical information, in the form of assertions. Assertions were used to aid the knowledge representation required for CFD, and was complemented with objects that were represented by frames having slots to store appropriate information to fully describe a particular object. These objects were created to facilitate the storage of non boundary condition data.

Prior to describing the techniques developed for the KBFE, an overview of the system is given with a brief description of the architecture, auxiliary LISP functions and the data manipulation functions and how these are used to initially set up the database. Object structures are then discussed with a view to highlighting the role they have within the rules. A brief introduction to pattern matching is also given. This leads onto the rulebase language developed for the KBFE and indicates the firing mechanisms that were formulated. Finally, the inference engine is described with respect to forward and backward chaining through inference networks, and how the objects have been included within the rules.

The core of this system revolves around the interaction of the inference engine, knowledge-bases and the database. Extracting LISP functions written specifically for the current application, as well as the associated objects and knowledge bases would leave the inference engine code and the corresponding data representation formalisms. This would then be able to be applied to a different application.

6.2 Symbolic pattern matching

Expressions within LISP are collective groups of atoms and lists. Pattern matching considers two expressions: a pattern and a datum. Patterns contain elements called **pattern variables** which are atoms prefixed with **$**, whereas a datum is an expression which contains knowledge. The following are examples of valid CFD patterns ...

(boundary name for $type $identity $nodes is $name)
(cardinal for surface $nodes is $cardinal)

... and appropriate datum expressions would be ...

(boundary name for inlet 1 (1 7) is entry)
(cardinal for surface (1 7) is west).

When a pattern contains no pattern variables, that pattern matches a datum only if the pattern is exactly the same as the datum, with each corresponding position occupied by the same atom. Thus ...

(X HAS 3 REGIONS) successfully matches **(X HAS 3 REGIONS)**
(X HAS 4 REGIONS) fails to match **(X HAS 3 REGIONS).**

When a pattern contains pattern variables, the corresponding position in the datum can contain anything. Thus ...

(X HAS $N REGIONS) matches **(X HAS 3 REGIONS)**
(X HAS $N $N) fails to match with **(X HAS 3 REGIONS)**

In the latter example failure occurs because the pattern variable **$N** is used twice, on the second occasion the atom 3 is replaced, and the match fails. The function **MATCH** performs the pattern matching operation and returns either an association list, NIL or FAIL. Thus ...

(a) **(match '(x has $n regions) '(x has 3 regions))** returns **(($N 3))**

(b) **(match '(x has 3 $?) '(x has 3 regions))** returns **(($? regions))**

(c) **(match '(x has $n $?) '(x has 3 regions))** returns **(($N 3) ($? regions))**

(d) **(match '(x has 3 regions) '(x has 3 regions))** returns **NIL**

(e) **(match '(x has 3 regions) '(x has 4 regions))** returns **FAIL.**

The pattern matching process, as indicated above, can return three possible answers: an association list, NIL or FAIL. FAIL indicates that the match has been unsuccessful, whereas an association list or NIL suggests a match has been performed. NIL indicates that the match was positive with all atoms being identical in both pattern and datum, without any pattern variables being present. An association list, on the other hand, suggests that pattern variables were present, the answer giving the appropriate bindings.

6.3 Symbolic pattern unification

Pattern unification matches two patterns as opposed to one pattern and one datum. However, the pattern variable is taken from the first pattern if both patterns have variables in the same position. Thus ...

(a) **(unify '(x has $n regions) '(x has 3 regions))** returns **(($N 3))**

(b) **(unify '(x has 3 regions) '(x has $n regions))** returns **(($N 3))**

(c) **(unify '(x has $n1 regions) '(x has $n2 regions))** returns **(($N1 $N2))**

(d) **(unify '(x has 3 regions) '(x has 3 regions))** returns **NIL**

(e) **(unify '(x has 3 regions) '(x has 4 regions))** returns **FAIL.**

As with pattern matching, unification can return one of three answers: FAIL, NIL or an association list. The same conditions apply with the answers as with the pattern matching described above.

6.4 Inferencing techniques

Forward and backward chaining are the two inferencing techniques incorporated within the KBFE, the details of which will be described in section 6.9. Initially forward chaining is performed with the implementation of backward chaining as and when further information cannot be extracted from the assertions or objects. Both techniques extract data from assertions and objects through the use of pattern matching and pattern unification. Inference networks are used to link rules within knowledge bases, and are used to prevent the unnecessary consideration of irrelevant rules. As the system progresses through a network, a bindings list is created which stores current data extracted from assertions. A bindings list consists of numerous lists of bindings. A set of bindings is defined as an association list of pattern variables and corresponding values, **((*variable_1 value_1) (*variable_2 value_2))**. A bindings list is a list of association lists, and will usually contain numerous identical variable-value pairs within each set of bindings. The resulting bindings list, after complete progression through an inference network back to the original base rule, will provide the data required to successfully fire the rule consequents.

6.5 System architecture

The system was developed on a VAX 785 machine using Common LISP (Versions 13.6 and 14.1) within the POPLOG programming environment. The KBFE is independent of PHOENICS in so much as it does not have to be run simultaneously. The KBFE creates a data file on disk which is then read by PHOENICS after initiating an analysis to be performed. The feasibility of pseudo real time monitoring of the solution algorithm has been investigated, and could be initiated from within the KBFE. This will be discussed in Chapter 7.

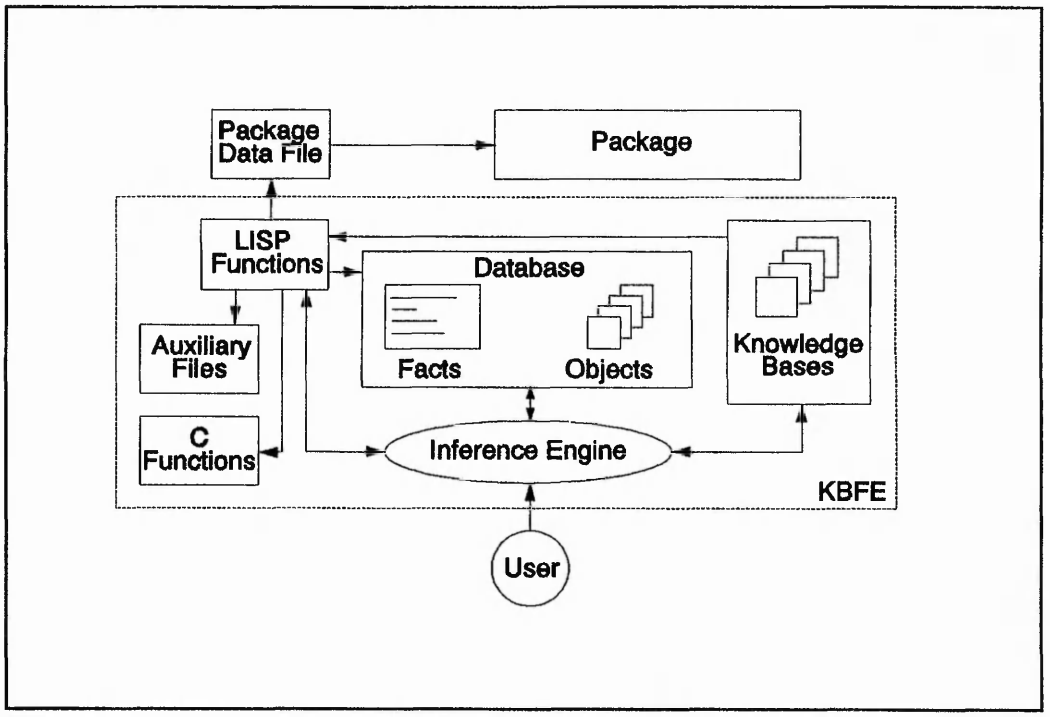


Figure 6.1: Knowledge Based Front End system architecture

Figure 6.1 shows the system architecture and indicates the interactions of the inference engine, knowledge-bases, database, LISP and C functions. The LISP functions contain user interface and data manipulation procedures. Simple user interface functions provide the mechanism with which the user can converse with the system during preliminary input of the geometrical information, for example nodal and connectivity data. The data manipulation functions are those that are specific to each application and perform tasks such as generating initial assertions from the geometrical data.

The three most important facets of the system are the inference engine, the knowledge-bases and the database. The interaction between these parts essentially forms the entire platform on which the development has been built. The inference engine has been developed from the fundamental concepts of forward and backward chaining and implements depth-first search on appropriate inferencing networks. Knowledge-bases are categorised into predefined areas and the database utilises two methods of storing and representing data.

In order to completely describe the interactions and the structure of each key facet it will be necessary to initially consider the way in which the system stores and represents data or information. This then enables a progression onto how the rules utilise this data through describing the rule syntax and available rule firing mechanisms. Finally, a description of the inferencing processes and how inferencing networks are created and used will be given.

6.6 LISP functions

The LISP functions shown in Figure 6.1 consist of User Interface Functions (UIFs) and Data Manipulation Functions (DMFs). The UIFs are used to provide interaction with the user and consist of specific functions written for the application, and functions used by the inference engine. The functions used by the inference engine prompt the user for information relating to specific objects and / or assertion data. The DMFs are written specifically for the application, and manipulate the data provided by the fundamental data entry functions.

6.6.1 User interface functions

There are two types of user interface functions : (1) fundamental data entry and (2) user prompting functions. Fundamental data entry functions are those which gather the initial data required in order to generate a preliminary assertions list. Examples of such fundamental data entry functions are those required to enter geometrical information such as nodal co-ordinates and their connectivities. Also included are functions for entering inlet, outlet and obstruction information. The fundamental data entry functions are specific to the application.

The user interface functions generate dialogue to converse with the user, as indicated in Figure 6.2 and Figure 6.3, which show the methods used for entering the nodal coordinates and nodal connectivities. The fundamental data entry functions are used to obtain data specifically for the application, whereas the user prompting functions are those used by the inference engine to enquire about object values or factual assertions. The user prompting functions will be discussed in section 6.10.

```
Enter the radial ordinate for node 1 == 0
Enter the axial ordinate for node 1 == 0

Enter the radial ordinate for node 2 == 5
Enter the axial ordinate for node 2 == 0

Enter the radial ordinate for node 3 == 6
Enter the axial ordinate for node 3 == 0

Enter the radial ordinate for node 4 == 50
Enter the axial ordinate for node 4 == 0

Enter the radial ordinate for node 5 == 5
Enter the axial ordinate for node 5 == 30

Enter the radial ordinate for node 6 == ?


The nodal-coordinates should be entered in < mm > depending on the prompt.
LIST - lists the nodes
M AXIS NODE - modify the coordinate of the node on axis axis
M AXIS - modify the current nodal coordinate on axis

Enter the radial ordinate for node 6 == list

((1 (0.0 0.0 0.0) ())
 (2 (0.0 0.005 0.0) ())
 (3 (0.0 0.006 0.0) ())
 (4 (0.0 0.05 0.0) ())
 (5 (0.0 0.005 0.03) ()))

Enter the radial ordinate for node 6 == 6
Enter the axial ordinate for node 6 == 30

Enter the radial ordinate for node 7 == 0
Enter the axial ordinate for node 7 == 49

Enter the radial ordinate for node 8 == 50
Enter the axial ordinate for node 8 == 49

Enter the radial ordinate for node 9 == 0
Enter the axial ordinate for node 9 == 50

Enter the radial ordinate for node 10 == 50
Enter the axial ordinate for node 10 == 50

Enter the radial ordinate for node 11 == end
```

Figure 6.2: Fundamental data entry functions : Geometry data entry screen - Nodal coordinates

```
Enter connectivity command, ? for help == ?

--- Connectivity HELP ---
LIST                          - list nodal information
CONNECT   node_i j .... z     - Connects node_i to j .... z
C         node_i j .... z     - Connects node_i to j .... z
REMOVE    node_i j .... z     - Removes node_i from j ... z
R         node_i j .... z     - Removes node_i from j ... z

Enter connectivity command, ? for help == c 1 2 7
Enter connectivity command, ? for help == c 2 3 5
Enter connectivity command, ? for help == c 3 6 4
Enter connectivity command, ? for help == c 5 6
Enter connectivity command, ? for help == c 7 9 8
Enter connectivity command, ? for help == c 8 4 10
Enter connectivity command, ? for help == c 9 10
Enter connectivity command, ? for help == list

((1 (0.0 0.0 0.0) (2 7))
 (2 (0.0 0.005 0.0) (3 5 1))
 (3 (0.0 0.006 0.0) (6 4 2))
 (4 (0.0 0.05 0.0) (8 3))
 (5 (0.0 0.005 0.03) (6 2))
 (6 (0.0 0.06 0.03) (5 3))
 (7 (0.0 0.0 0.049) (9 8 1))
 (8 (0.0 0.05 0.049) (4 10 7))
 (9 (0.0 0.0 0.05) (10 7))
 (10 (0.0 0.05 0.05) (9 8)))

Enter connectivity command, ? for help == end
```

Figure 6.3: Fundamental data entry functions: Geometry data entry screen - Nodal connectivity

Throughout the inferencing process the system endeavours to obtain data as and when it is required, in order to prove or disprove the antecedent the engine is considering. This involves prompting the user for an object value, a template value or initiating backward chaining on the rulebase inference network. Backward chaining will be discussed in section 6.9.4. Prompting the user for data is performed by specific functions developed for either objects or templates, which will be discussed later. Provided that the data for the object cannot be obtained through any of the associated slots, the system invokes a menu

function which utilises specific slot data to establish the means to ask the user for the data. Again, the data entry under such circumstances is performed through dialogue.

```
PHOENICS essentially uses two types of coordinate systems - cartesian and
cylindrical. For two dimensional configurations which this system can initially
develop the XY plane will be utilised. This will be automatically translated into
the respective XY or YZ planes which phoenics requires depending upon your
choice of either cartesian or cylindrical coordinates.

1: CYLINDRICAL
2: CARTESIAN

Default value : CARTESIAN
Are the coordinates cartesian or cylindrical ?
(Enter 1 - 2) : == 1
```

Figure 6.4: User prompting functions: Data entry screen - Object enquiry

```
Enter the w1 velocity for jet == 2.1915
```

Figure 6.5: User prompting functions: Data entry screen - Assertion template enquiry

Figure 6.4 and Figure 6.5 indicate typical data entry screens for object and assertion template enquiry.

6.6.2 Data manipulation functions

The data manipulation functions are used primarily for creating an initial assertions list from information gathered using the fundamental data entry. Geometrical data entered using the preliminary user interface functions consist of the nodal coordinates and the associated connectivity. This is stored in the system global variable *NODES*, as shown in Figure 6.6. This data is manipulated using auxiliary functions to regionalise the

```
((1 (0.0 0.0 0.0) ())
 (2 (0.0 0.005 0.0) ())
 (3 (0.0 0.006 0.0) ())
 (4 (0.0 0.05 0.0) ())
 (5 (0.0 0.005 0.03) ())
 (6 (0.0 0.006 0.03) ())
 (7 (0.0 0.0 0.049) ())
 (8 (0.0 0.05 0.049) ())
 (9 (0.0 0.0 0.05) ())
 (10 (0.0 0.05 0.05) ()))
```

(a) : Without connectivity data entered

```
((1 (0.0 0.0 0.0) (2 7))
 (2 (0.0 0.005 0.0) (3 5 1))
 (3 (0.0 0.006 0.0) (6 4 2))
 (4 (0.0 0.05 0.0) (8 3))
 (5 (0.0 0.005 0.03) (6 2))
 (6 (0.0 0.006 0.03) (5 3))
 (7 (0.0 0.0 0.049) (9 8 1))
 (8 (0.0 0.05 0.049) (4 10 7))
 (9 (0.0 0.0 0.05) (10 7))
 (10 (0.0 0.05 0.05) (9 8)))
```

(b) : With connectivity data entered

Figure 6.6: LISP special variable: ***NODES***

domain. Manipulation of this data to extract additional information results in the need for further global variables, *BOUNDARIES* and *REGIONS*. These variables, as shown in Figure 6.7 and Figure 6.8, contain information used to augment the assertions list with templates such as ...

(Cardinal for surface $nodes is $cardinal)

(surface $nodes is part of $obstruction)

($axis has $N regions)

(surface $nodes is in $axis regions $start to $finish)

(surface $nodes interfaces $axis regions $start and $last)

```
(((7 8) ((Cardinal LOW)     (Type OBSTRUCTION) (Name WALL)))
 ((5 6) ((Cardinal HIGH)    (Type OBSTRUCTION) (Name PIPE)))
 ((3 6) ((Cardinal NORTH)   (Type OBSTRUCTION) (Name PIPE)))
 ((2 5) ((Cardinal SOUTH)   (Type OBSTRUCTION) (Name PIPE)))
 ((3 4) ((Cardinal LOW)     (Type OUTLET)      (Name OUTLET2)))
 ((4 8) ((Cardinal NORTH)   (Type OUTLET)      (Name OUTLET1)))
 ((1 2) ((Cardinal LOW)     (Type INLET)       (Name JET)))
 ((1 7) ((Cardinal SOUTH)   (Type WALL)        (NameUNKNOWN)))
```

Figure 6.7: LISP special variable: ***BOUNDARIES***

These templates, and the information contained therein, are used primarily by the command synthesis rules.

6.7 System database

The system recognises three methods of storing and manipulating information: assertions, objects and LISP variables. Assertions are stored within a pseudo-blackboard data list, and objects as LISP structures. The inferencing process cannot control LISP variables, examples of which are *NODES*, *REGIONS* and *BOUNDARIES*. These do not affect the system but are heavily used for creating the initial assertions from geometrical data, and are used within the LISP functions shown in Figure 6.1. LISP variables can be either lexical or special variables. A lexical variable is one which is only accessible within a LISP virtual fence, Winston and Horn (1989). However, special variables are essentially global to all LISP functions. Geometrical data is stored as special LISP variables for use in functions that create initial assertions.

The two primary methods of storing data, as mentioned above, are assertions and objects. Section 6.7.1 introduces the method of storing information within assertions, Section 6.7.2 explains how assertion categorisation enables a blackboard data structure to be implemented. This is followed by the discussion of object declarations through LISP structures.

```
((X ())
 (Y ((1 ((ALPHA 0.5)
          (C1 0.0)
          (C2 0.005)
          (L 0.005)
          (MESH (  0.136527e-3 0.285139e-3 0.448656e-3 ...

                   ... 0.004543 0.004706 0.004855 0.005))))
     (2 ((ALPHA 0.5)
          (C1 0.005)
          (C2 0.006)
          (L 0.001)
          (MESH (  0.136527e-3   0.378829e-3   0.621042e-3   0.863101e-3
                   0.001))))
     (3 ((ALPHA 0.0)
          (C1 0.006)
          (C2 0.05)
          (L 0.044)
          (MESH (  0.136528e-3 0.273262e-3 0.410556e-3 ...

                   ... 0.042998 0.043506 0.044))))))
 (Z ((1 ((ALPHA 1.0)
          (C1 0.0)
          (C2 0.03)
          (L 0.03)
          (MESH (  0.439251e-3 0.899534e-3 0.001526 ...

                   ... 0.029589 0.029727 0.029863 0.03)))
     (2 ((ALPHA 0.5)
          (C1 0.03)
          (C2 0.049)
          (L 0.019)
          (MESH (  0.136527e-3 0.275869e-3 0.418439e-3 ...

                   ... 0.018692 0.018832 0.018968 0.019))))
     (3 ((ALPHA 0.0)
          (C1 0.049)
          (C2 0.05)
          (L 0.001)
          (MESH (  0.136527e-3 0.306063e-3 0.528761e-3 0.001)))))))
```

Figure 6.8: LISP special variable: ***REGIONS***

6.7.1 Assertions

An assertion is a list of LISP atoms that are combined in such a way as to create a sentence or phrase. These have common, application specific, templates that when applied to separate data create unique assertions. Wildcard variables reside within the templates, prefixed with $, which are used to indicate where relevant data should be located. Given the following assertion template ...

(Boundary name for $Type $Identity $Nodes is $Name)

... and a bindings list ...

(($Type Inlet) ($Identity 1) ($Nodes (1 7)) ($Name Inlet1))

.

.

(($Type Outlet) ($Identity 1) ($Nodes (4 8)) ($Name Exit))),

the following assertions would result ...

(Boundary name for Inlet 1 (1 7) is Inlet1)

.

.

(Boundary name for outlet 1 (4 8) is Exit)

The bindings list is generated dynamically through inferencing on the rules by a combination of forward and backward chaining. This is explained in Section 6.9.3. Table 6.1 gives a sample of current CFD assertion templates used in the KBFE.

6.7.2 Assertions list

The assertions list (blackboard) is part of the main system database used for storing factual declarations or assertions organised into predefined levels or categories. The blackboard is a common platform from which information can be easily accessible to any knowledge base primarily via the inference engine. It does not contain linkages between entries on the same or different levels and cannot pass data between the different levels, as described by Reddy and O'Hare (1991). During the inferencing process data contained

```
( ((Boundary name for $type $identity $nodes is $name))
  ((Cardinal for surface $nodes is $cardinal))
  ((Surface $surface is part of $obstruction))
  (($dependent-variable at $type boundary $name is $condition at $quantity))
  (($axis has $n regions))
  (($axis region $No cells $first to $last))
  (($axis region $No co-ordinates $first to $last))
  ((Surface $nodes is in $axis regions $start to $finish))
  ((Surface $nodes interfaces $axis regions $start and $last)) )
```

Table 6.1: Assertion templates

within various levels of the blackboard is used within the rules which generate further data to be written to any other level, including its own. Figure 6.9 shows the partitioning of the various levels of information stored within the blackboard and how this has been abstracted in LISP code. The blackboard is a complex list whose initial element within each sub-list corresponds to the assertion categories, and the remaining values within each list are the assertions for that partition.

Storage of facts using this technique reduces the number of pattern matching functions required to extract relevant data associated with a rule under consideration by the inference engine. Furthermore, this categorisation of assertions prevents the inefficient consideration of facts that have no possible bearing on the evaluation of a rule.

6.7.3 Objects

Objects are created from LISP structures and store information that has a constrained definition. The simulation of inheritance using POPLOG Common LISP, a non Common LISP Object System (CLOS) version of LISP, can be achieved using structures. Information required to be stored within the KBFE for CFD purposes, which could be considered as being constrained, would consist of number-of-inlets, number-of-outlets, flow-regime, and whole-field-variables. The LISP structure is shown in Figure 6.10.

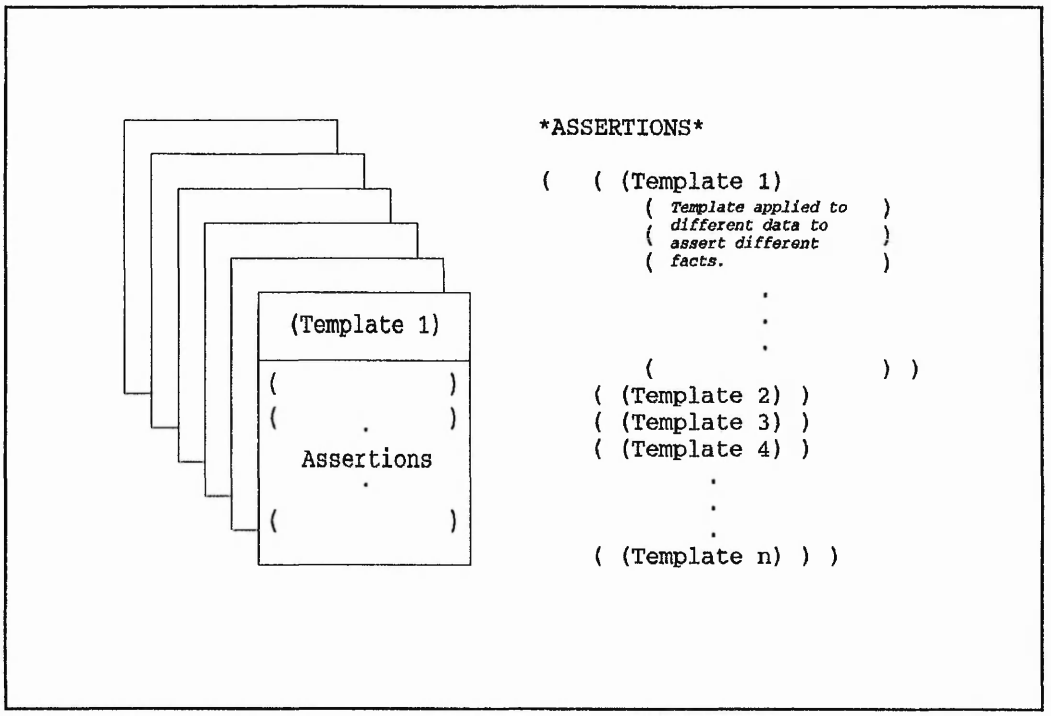


Figure 6.9: Blackboard structure and abstract LISP representation

LISP structures are predefined using the LISP command DEFSTRUCT, Steele (1990). Such structures contain user defined fields, which allow inheritance of default values to subsequent variables created using the LISP command (**make-*structure-name***). These fields have been likened to object slots for the KBFE development and, as such, create an analogy that the LISP structures can be considered as objects. Extending this allows all slots to reside within an object frame.

Figure 6.10 represents each object as a frame with slots associated with each structure field. Each object inherits the field from the basic structure definition, and is only overwritten if the field is explicitly defined for that object.

6.7.4 Object slot descriptions

DESCRIPTION - This slot allows the knowledge engineer developing the system the facility of expanding on the name of the object if this does not adequately define its purpose. The contents of this slot need to be in a list form, thus '(.....). The inference

```
(Defstruct object
    (Description        nil)
    (Type               nil)
    (Preface            nil)
    (FixedValue         nil)
    (DisallowedValues   nil)
    (AllowedValues      nil)
    (DefaultValue       nil)
    (ComputeValue       nil)
    (Units              nil)
    (Value              nil)
    (Prompt             nil)
    (Help               nil)
    (Status             nil)
    (RuleBase           nil))
```

Figure 6.10: Object frame and slots through LISP structures

engine does not access this field as it is purely a documentation facility.

TYPE - This defines the type of object. Allowable types are: Integer, Real, List, Text or String.

PREFACE - This is used to describe the purpose of the object to the user. It acts as a low level help facility that is utilised by the inference engine and is always presented to the user to compliment the object prompt. The preface should be entered as a list and can be dynamically altered depending upon the current status of other objects.

FIXEDVALUE - Occasionally restrictions on the object value may be required whereby it is necessary only to allow one value. Under such circumstances a fixed value is provided to force the inference engine to accept the value within this slot. A typical situation would be to let the value of **number-of-dimensions** be fixed if the knowledge base was only to consider two dimensional problems. This prevents the system ever asking the user to input a value.

DISALLOWEDVALUES - This acts as a data input checking facility. Inclusion of a list of values here would indicate to the system to match the user value with the list. Any equivalences would be disallowed.

ALLOWEDVALUES - Similar in operation to the DisAllowedValues except that an equivalence would be allowed.

DEFAULTVALUE - Providing a list of allowed values enables the inference engine to establish that the user is given a choice of answers to respond with. These choices are then presented to the user using a menu function, an example of which is shown in Figure 6.4. If a default value is declared then entering return would instantiate the object with the default value.

COMPUTEVALUE - If the object requires some form of numerical computation to establish a value, this slot would allow object instantiation as a result of executing some other LISP function. A valid LISP statement or function name could reside in the slot.

UNITS - This slot is used to indicate to the user what units are to be used in association with the object.

VALUE - This slot is allocated a value when the object has been instantiated as a result of either being given a value by the user or through rule firing. Both methods of instantiation are a consequence of the inferencing process.

PROMPT - This slot must be allocated a string prompt for use with the menu function if the object is not a fixed value.

HELP - This slot contains additional information concerning the object. There exists four different levels of help text associated with the active user model. The levels are **DEFAULT, NOVICE, EXPERIENCED,** and **ADVANCED.** The slot contains a four element association list ...

(**(DEFAULT (*Text*)) (NOVICE (*Text*)) (EXPERIENCED (*Text*)) (ADVANCED (*Text*))**)

The text associated with each level follows the structure of the preface slot. If the **DEFAULT** slot is instantiated, and the others are uninstantiated, then the default slot

prevails. Specific, defined help overrides the default help.

STATUS - Initially all objects have a fixed status. This implies that once they have been instantiated their value cannot be altered. However, a 'volatile' status allows an object to be reinstantiated whenever the inferencing process dictates. This was necessitated through the use of variable boundary conditions requiring different assigned values.

RULEBASE - If TRUE (**T**), as opposed to the default of NIL (()), there exists a rulebase of the same name as the object, concatenated with **-RB.** For example, an object with the name **DELTA**, whose rulebase slot is T, would require a rulebase called **DELTA-RB** to contain appropriate rules to instantiate this object. During inferencing this slot value is given NIL while the rulebase is being operated upon. After completion the slot is returned to its original value to allow further inferencing if necessary at a later date.

6.8 RuleBase language

The rulebase language has been developed to accommodate all possible knowledge representation requirements for a KBFE to a CFD package. This has been an evolutionary process and implements concurrent manipulation of the bindings list generated throughout the inferencing process. The rule language basically has two formats, the User Rule Syntax (URS) and the System Rule Syntax (SRS). The former is entered by the knowledge engineer. The SRS is slightly different in so much as a rigid representation is needed by the inference engine.

6.8.1 User Rule Syntax

As mentioned above, the user rule syntax is used directly for entering rules within a knowledge base. Rules are entered into a file using a standard text editor and take the form of a three element LISP structure ...

(REMEMBER-RULE RULEBASE-NAME RULE)

The three element LISP structure is a macro call, the macro being REMEMBER-RULE. The macro is used to create the system rule syntax. The RULEBASE-NAME is a LISP variable used for storing all the rules associated with a particular category, examples of

which are CONVERSION-FACTOR-RB, DELTA-RB, FLUID-RB, G13-RB, and GEOMETRY-RB. The RULE must adhere to one of the three fundamental rule structures, these are ...

(RULE-NAME ANTECEDENTS CONSEQUENTS)
(RULE-NAME CONSEQUENTS-ONLY)
(RULE-NAME LIST-QUANTIFICATION-RULE)

The rule-name is optional. Figure 6.11 shows the omission and inclusion of the rule-name in the user rule syntax. The three fundamental rule structures, given above, are a standard production rule, a consequent only rule and a list quantification rule respectively. These are described in Sections 6.8.3, 6.8.4 and 6.8.9.

```
(REMEMBER-RULE  Rulebase-name
        '( (IF        (ANTECEDENT 1)
                      (ANTECEDENT 2))
           (THEN      (CONSEQUENT 1)
                      (CONSEQUENT 2))))
```

(a): Omission of the rule name from the User Rule Syntax

```
(REMEMBER-RULE  Rulebase-name
        '((RULE_  EXAMPLE  RULE  NAME)
           (IF        (ANTECEDENT 1)
                      (ANTECEDENT 2))
           (THEN      (CONSEQUENT 1)
                      (CONSEQUENT 2))))
```

(b): Inclusion of the rule name in the User Rule Syntax

Figure 6.11: Omission / Inclusion of rule name in User Rule Syntax

Each rulebase, prior to creating the inference network described in section 6.9.2, is a list taking the following form ...

(RULE-1 RULE-2 .... RULE-N)

where N is the number of rules within the associated rulebase.

6.8.2 System Rule Syntax

The system rule syntax is slightly different from the user rule syntax in that the rules have to be a standard production rule or a list quantification rule. Furthermore, a rule-name must be present. The macro REMEMBER-RULE performs this operation. During the loading of the rules into each rule-base the system increments a count, *RULE-COUNT*, which keeps a record of the number of rules entered. If a rule-name is present then this is concatenated with **-RULE-*RuleNumber***, where the ***RuleNumber*** is the current value of *RULE-COUNT*. However, if a rule-name has been omitted then one is automatically assigned using **RULE-*RuleNumber***. Figure 6.12 shows the system rule syntax for the rules given in Figure 6.11. Furthermore, all consequent only rules are transposed to standard production rules whose antecedent becomes (IF NOTHING).

6.8.3 Production rules : Antecedents

Standard production rules, IF .... THEN rules, have by default conjunctively combined antecedents. Disjunctive antecedents can be incorporated. Figure 6.13 shows conjunctive and disjunctive production rules. Figure 6.14 shows the transposition of a consequent only rule to the fundamental production rule structure.

Antecedents are used by the inference engine to establish whether a rule can be fired or not. For a rule to be fired all antecedents must be true. Antecedents can consist of templates which filter the current bindings being considered by the inference engine, perform statement verification, or determine whether an assertion is instantiated or not. A detailed discussion of the inference engine is given in Section 6.9. An antecedent template can be the same as an assertion template or a consequent template, the binding variables in both need not be the same. A statement verification antecedent consists of the fundamental structure given below:-

```
( ( RULE-Rulenumber
        (IF     (ANTECEDENT 1)
                (ANTECEDENT 2))
        (THEN   (CONSEQUENT 1)
                (CONSEQUENT 2))))
    .
    .
    .
  ( EXAMPLE-RULE-NAME-Rulenumber
        (IF     (ANTECEDENT 1)
                (ANTECEDENT 2))
        (THEN   (CONSEQUENT 1)
                (CONSEQUENT 2))))
```

Figure 6.12: System Rule Syntax after Figure 6.11

(OPERAND-1 OPERATOR OPERAND-2)

Valid operators used by such an antecedent consist of:-

INCLUDES - Operand-1 must be a list, (a b c ....), and Operand-2 is either a non-list object, bindings variable or a fixed value. A true value for the antecedent is returned if Operand-2 is a member of Operand-1.

EXCLUDES - The same conditions apply as for the INCLUDES operator, except that a true value for the antecedent is returned if Operand-2 is not a member of Operand-1.

OVERLAPS - Both operands need to be lists, they can be derived from list objects, bindings variables or fixed values. A true value is returned for the antecedent if both lists intersect.

=, IS, ARE - These are equality operators, the first of which is used for numerical Operands, the remaining are used for non-numerical Operands. "Are" has been included for plural definitions, such as **(co-ordinates are cartesian)**.

```
(RULE-Rulenumber
    (IF     (ANTECEDENT 1)
            (ANTECEDENT 2))
    (THEN   (CONSEQUENT 1)
            (CONSEQUENT 2)))
```

(a): Conjunctive production rule

```
(RULE-Rulenumber
    (IF     (OR ((ANTECEDENT 1)
                 (ANTECEDENT 2))
                ((ANTECEDENT 3))))
    (THEN   (CONSEQUENT 1)
            (CONSEQUENT 2)))
```

(a): Disjunctive production rule

Figure 6.13: (a) Conjunctive, (b) Disjunctive production rules

<>, IS-NOT, ARE-NOT - These are inequality operators.

>=, <=, >, < - These are basic numerical operators.

Checking for assertion instantiation is instigated by an assertion with its last two atoms in the template, being one of

(is instantiated), (are instantiated),
(is-not instantiated), (are-not instantiated),
(is uninstantiated) or **(are uninstantiated)**.

This is exemplified in the rule-base BC-RB shown in Appendix E. The antecedent using this facility is ...

($velocity at inlet boundary $Name is constant is uninstantiated).

(REMEMBER-RULE *Rulebase-name*
'(->Q1 ?= LSWEEP 100))

(a): User Rule Syntax

(RULE-*Rulenumber*
(IF NOTHING)
(THEN (->Q1 ?= LSWEEP 100)))

(b): After transposition: System Rule Syntax

Figure 6.14: Transposition of a consequent only rule to a standard production rule

This will check to see if the assertion, assuming that **$Name** is ENTRY, **$velocity** is W1, and the value is 0.25 m/s, ...

(W1 at inlet boundary entry is constant at 0.25),

... exists within the assertions list. If it does exist then the antecedent would return NIL (()), however a TRUE (**T**) value would be returned if such an assertion did not exist. Objects can also be checked for instantiation using the same principles.

Consequent only rules are augmented by the macro REMEMBER-RULE with the antecedent (IF NOTHING). The inference engine recognises this antecedent and moves directly onto firing the rule consequents.

6.8.4 Production rules : Consequents

The consequents of a rule are fired providing all the antecedents are true. Upon entering the firing routines, there exists a bindings list which has been dynamically expanded and contracted as the progression through the antecedents advanced. This bindings list is used to fire the consequents. Section 6.8.10 describes the firing modes with which the

consequents can be implemented, after the antecedents have been successfully proven.

Consequents can be used to run LISP functions, instantiate objects, manipulate object slots, manipulate bindings and synthesise stored data into the target package commands. Each of these features is discussed below.

6.8.5 Object declaration and object slot manipulation consequents

This group of consequents instantiate objects with appropriate values. The structure is given by ...

(OBJECT-NAME DECLARATOR VALUE)

The OBJECT-NAME is any valid object previously created using the macro SET-OBJECT. Any of the valid declarators described below can be used.

ALLOWEDVALUES - Using this as a declarator indicates to the inference engine that the object slot ALLOWEDVALUES is to be instantiated with VALUE, which should be a list. The system automatically instantiates the slot DEFAULTVALUE with the first element of the declared list. LEONARDO did not allow the ALLOWEDVALUE slot of an object to be altered during the running of a particular knowledge base. This was found to be a problem because there were occasions where such a capability would have been useful. This facility was therefore incorporated into the system.

STATUS - This declarator indicates that the status of a particular object is to be modified. The status of all objects is determined when they are initially defined using the macro SET-OBJECT. The status can be either FIXED or VOLATILE. The ability to alter an object's status was found to be necessary when specifying boundary conditions within a CFD analysis. For example, the porosity of blockages within a domain need not be the same, and as such it was necessary to allow the flexibility to alter individual porosity values. The object POROSITY was initially defined as having a volatile status. Depending upon the user's responses to the inferencing performed on the rule-base, POROSITY-DEFINITION-RB, the POROSITY status can be changed to FIXED. If the status remains VOLATILE then every occurrence of the object, POROSITY, within a rule forces the system to prompt the user to enter a value regardless of it already being

instantiated. This is exemplified in the rule-base G11-RB, Appendix E.

PROMPT - This declarator allows the system to dynamically alter an object's user prompt. This was found to be useful when the object's status is VOLATILE. This is, again, exemplified in the rulebase G11-RB, where the porosity value for each blockage within a domain need not be the same, but identification is important when prompting the user to enter an obstruction's porosity value.

EXCLUDES - For use when it is necessary to remove a value from a list object.

INCLUDES - For use when it is necessary to include a value in a list object.

=, IS, ARE - These are used for numeric and non-numeric declarations, for examples **(FLOW-REGIME IS LAMINAR)**, **(COORDINATES ARE CARTESIAN)** and **(DELTA = BOUNDARY-LAYER-THICKNESS / 3.0)**.

+= - This declarator has been taken from the C language, its meaning is described by the following two statements which are identical ...

(INLET-FLOW-AREA = INLET-FLOW-AREA + $AREA)
(INLET-FLOW-AREA += $AREA)

Within the structure of this type of consequent, i.e. (OBJECT-NAME DECLARATOR VALUE), the value is determined prior to firing the consequent. When a consequent is written in the user syntax form the structure need not necessarily contain just three elements, as shown above in the description for the **=, IS, ARE** declarators where an example consequent was **(DELTA = BOUNDARY-LAYER-THICKNESS / 3.0)**, which contains five atoms. The most important aspect is that the first atom must be an object, i.e. **DELTA**, and the second atom must be a valid declarator, i.e. =. The value, which can be considered as the remaining atoms commencing with the third entry, initially has all object values inserted in place of the symbol and all bindings variables replaced with their appropriate values. Upon completion of all necessary replacements, the expression is passed to a mathematical parser, Section 6.9.6, and the result is returned which is then inserted in place of the expression. This automatically reduces the consequent to a three element structure, which can then be fired, providing the necessary object declaration.

6.8.6 Function calling consequents

The KBFE has a predetermined set of LISP functions that it can call from within rule consequents. The name of these functions are stored within the special LISP variable, *IFE-FUNCTIONS*, and are described below.

ABS - A standard mathematical function that returns the absolute value of a number.

ASK - This function has two definitions: to ask for an object value or to ask for a value to a given template. The template may be an assertion template or a consequent template. When querying the user for a value to a given object, the system utilises the object slots to establish any restrictions that will have to be implemented by the menu system. A full description of how this is performed is given in Section 6.10. When asking for a value to a template, the consequent has to employ extra information detailing what sort of data needs to be entered. This is utilised in the rule-base BC-RB, where, for example, the consequent **(U1 VALUE IS (ASK U1 VELOCITY AT $NAME ((TYPE REAL) (UNITS "m / s") (HELP ((DEFAULT (AXIS-1 VELOCITY))))))** asks for an appropriate value to be entered. This consequent is associated to an antecedent of the form **($VALUE VALUE IS $QUANTITY)**, where **$VALUE** is **U1**. The bindings variable **$QUANTITY** will be instantiated through the **ASK** condition in the consequent. This consequent, or consequents adhering the same pattern, must abide by the following rules for their implementation. The first atom must be ASK, the last atom of the template must be an association list. The association list, used in conjunction with the ASK consequent for assertions, contains slots which govern the type of answer the user can enter. The association list includes the following possible keys: **TYPE, ALLOWEDVALUES, DISALLOWEDVALUES, DEFAULTVALUE, UNITS, PREFACE,** and **CONSEQUENT.** When the user is prompted to enter data which would result in a fact being asserted, the system uses the association list key values to instantiate a temporary object, **FACT.** The object slots are instantiated with the assertion slot values, and the fact is presented to the user for response using the object enquiry functions.

INT - This function is most commonly used for the declaration of an object where its position within the consequent is usually on the right hand side of a numerical declarator. Its purpose is to return the integer part of a number, for example INT(3.4) would return 3. This function is used in rule-base G22-RB.

JOIN - This function is used to concatenate two symbols, for example (JOIN N WALL) would return NWALL. This is used in rule-base G13-RB.

MAX - Similar in purpose to INT, it is a mathematical function that returns the maximum of two or more numbers. MAX(1 2) would return 2.

RUN - This needs to be the first atom of a consequent and requests that the function given by the second atom should be executed as a result of firing the consequent under consideration. An example of its use can be seen in the rule-base GRID-RB.

SYMBOL-SPLIT - This function is used to remove part of an atom. For example, (SYMBOL-SPLIT 1 NORTH) would return N, and (SYMBOL-SPLIT 3 NORTH) would return NOR.

XC_1, XC_2, YC_1, YC_2, ZC_1, ZC_2 - These functions are used within the rule-base INLET-FLOW-AREA-RB. They are used to extract the coordinates from a two node surface, hence the affixes _1 and _2. The surfaces are represented using a two element list, such as (1 2), which indicates that the surface connects nodes 1 and 2. Figure 6.15 shows the rule used in INLET-FLOW-AREA-RB and the current status of the bindings list for the specification given prior to firing the consequent. The base rule, IFA1-RULE-*rulenumber*, is linked to the consequent IFA2-RULE-*rulenumber*, through its third antecedent, ...

(Inlet area for $nodes = $area)

... is linked to ...

(inlet area for $nodes = (0.5 * (ABS (((YC_2) ^ 2) - ((YC_1) ^ 2)))))

The third antecedent cannot be proved without backward chaining. As a consequence of backward chaining, the returned bindings list contains the answer for the area given by $AREA. The system recognised the functions YC_1 and YC_2 in the mathematical expression, and utilised the bindings to extract from the special LISP variable, *NODES*, the coordinates associated with nodes 1 and 2. Depending upon the function that is being called, it will return either the first or second of the X, Y or Z coordinate.

->Q1 - This indicates that the consequent is used for command synthesis.

RULEBASE: ***(Base rule and appropriate rules only)***

```
((NETWORK)
   ((IFA1-RULE-Rulenumber
      (IF   (Boundary name for inlet $identity $node is $name)
            (Cardinal for surface $nodes is $cardinal)
            (Inlet area for $nodes = $area))
      (THEN    (INLET-FLOW-AREA += $area))))
      .
      .
      .
   ((IFA2-RULE-Rulenumber
      (IF      (Coordinates are cylindrical)
               ((High Low) includes $cardinal))
      (THEN    (Inlet area for $nodes =
               (0.5 * (ABS (((YC_2) ^ 2) - ((YC_1) ^ 2))))) ))))
```

The bindings list prior to firing the second rule, given above, is ...

```
(((($identity 1) ($nodes (1 2)) ($name jet) ($cardinal low)))
```

After successfully firing the second rule, the returned bindings list is ...

```
(((($identity 1) ($nodes (1 2)) ($name jet)
      ($cardinal low)) ($area 1.8e-5)))
```

Figure 6.15: INLET-FLOW-AREA-RB: Use of the functions XC_1, XC_2, YC_1, ..., ZC_2

->1.0E??? - This function is used for establishing a residual reference value required by PHOENICS. The purpose is to establish the exponent part of a number and to concatenate this to 1.0E as indicated by the symbol for the function. For example, (->1.0E??? 1.786E-9) would return 1.0E-9. Its use can be seen in the rule-base G15-RB.

6.8.7 Bindings manipulation consequents

These consequents are used to manipulate the current bindings list which results from successfully proving all antecedents of a rule. The process of manipulating bindings is indicated as a template within a consequent template. Current, valid, bindings manipulation templates are:-

(AVERAGE *variable* FROM BINDINGS)
(SUM *variable* FROM BINDINGS)

where ***variable*** is the name of a bindings variable within the same rule but it is not prefixed with a $. For example, a bindings variable named $VELOCITY-VALUE would be represented in a bindings manipulation template as VELOCITY-VALUE.

Bindings manipulation consequents can be used in rules which are linked to antecedents in other rules in order to extract specific data during backward chaining. For example, if a current antecedent cannot be matched with an assertion or cannot be proved to be correct, backward chaining in the inference network commences and the antecedent is unified with the consequents of associated rules. A match would result in an association list containing the bindings variable in the antecedent and its matched value in the consequent of the associated rule. Figure 6.16 shows two rules, the first of which proceeds if $value is P1, the INLET-FLOW-AREA is greater than ZERO and the TOTAL INLET VELOCITY is known. There exists no assertion that will allow the TOTAL INLET VELOCITY to be obtained and, as such, backward chaining on the second rule would commence. Unifying the antecedent ...

(TOTAL INLET VELOCITY = $VELOCITY)

with ...

(TOTAL INLET VELOCITY = (SUM VELOCITY-VALUE FROM BINDINGS))

would result in the following bindings ...

(($VELOCITY (SUM VELOCITY-VALUE FROM BINDINGS)).

```
(RULE-Rulenumber
 (IF        ($value is P1)
            (INLET-FLOW-AREA > 0)
            (Total inlet velocity = $velocity)
            (Initial fluid-density = $density))
 (THEN      (Residual reference for $value =
              (->1.0e??? (0.01 * $density * $velocity
                                 * INLET-FLOW-AREA))))))

(RULE-Rulenumber
 (IF        (Boundary name for inlet $identity $nodes is $name)
            (Cardinal for surface $nodes is $cardinal)
            ($phi is perpendicular to $name)
            ($phi at inlet boundary $name is constant at $velocity-value))
 (THEN      (Total inlet velocity =
              (SUM VELOCITY-VALUE FROM BINDINGS))))
```

The bindings list, after successfully proving the antecedents of rule 2, would be ...

```
(((($value P1) ($identity 1) ($nodes (1 2)) ($name jet) ($cardinal low)
                ($phi w1) ($velocity-value 2.1915)))
```

... and the bindings list prior to firing the consequents of rule 1 would be ...

```
(((($value P1) ($velocity 2.1915)))
```

Figure 6.16: Bindings manipulation consequents

This indicates that $VELOCITY is a required binding and its value would be obtained from the successful completion of the second rule. The bindings created during progression through the first rule are carried forward into the second rule whilst trying to prove the antecedent of the first. This ensures that bindings variables in the second rule, which can be instantiated from information gathered in the first, is performed. However, bindings variables in the second rule that are not required in the first are not carried back for use in the first rule upon completing the second rule. Only the information established by the second, required by the first, is carried back. In this example the only

binding carried back would be $VELOCITY and its appropriate value. Upon successful completion of the second rule a bindings list results, which contains information relating to one or more inlet boundaries and their associated perpendicular velocities. Figure 6.16 indicates the bindings list after successfully completing the second rule and how the information contained within the bindings is translated into the bindings list needed to fire the first rule. This inferencing process is discussed in detail in Section 6.9.

6.8.8 Command synthesis consequents

Command synthesis consequents are used to translate the information contained within the database, objects and assertions, into appropriate PHOENICS commands. The user syntax allows three types of rules to fire command synthesis consequents: consequent only, list quantification and standard production rules. Consequent only rules, an example of which is given in G15-RB, are used to synthesise PHOENICS declarative commands, i.e. **LSWEEP = 100.** List quantification rules are used to fire command synthesis consequents for PHOENICS variables, such as the analysis dependent variables, all of which utilise the same PHOENICS commands. Examples of these commands are the specification of how to solve for a dependent variable, relaxation factors, output requirements and residual reference values. The rules used to perform the synthesis of these examples are given in G7-RB, G17-RB, G21-RB, and G15-RB respectively, Appendix E. Standard production rules simply employ conjunctive or disjunctive antecedents, which need to be proved, to fire command synthesis rules.

Command synthesis rules form the base rules from which associated rules are connected within the inference network. All command synthesis consequents must conform to the following template:-

(->Q1 ***Command-template Command Argument-1 ... Argument-n*****)**

The symbol **->Q1** indicates that the consequent is to define a CFD package command which is to be written to a list. Upon completing all of the rule-bases used to synthesise the commands, the list containing all of the commands is then written to a predefined file. Four types of commands can be accommodated within the system each of which is defined with a *Command-template.* Table 6.2 shows examples of valid PHOENICS commands, illustrating each of the four allowed *Command-templates.*

Command Template	Arguments	PHOENICS command
?[]	Solutn P1 Y Y Y N N N	Solutn(Y,Y,Y,N,N,N)
?[]=	Fiinit V1 0.01	Fiinit(V1) = 0.01
?=	Rho1 1.225	Rho1 = 1.225
Message	^Group 1.	GROUP 1.

Table 6.2: Command synthesis templates

6.8.9 List quantification rules

Rules, when conforming to the system rule syntax, have either two or three elements within the list. A list quantification rule is represented using the following template ...

(RULE-NAME (FOR ALL *list-object* ANTECEDENTS CONSEQUENTS))

The second element, which is a list, is used by the inference engine to establish whether the current rule is a list quantification rule or a production rule. The first atom of this second element is either **IF** or **FOR**. **IF** relates to a production rule, and **FOR** relates to a list quantification rule, the latter operates using instantiated list objects. The antecedents for a consequent only rule, specified under the system rule syntax, is (IF NOTHING).

List objects contain more than one value, for example the object DEPENDENT-VARIABLES could contain U1, V1, and H1. Under these circumstances the value of DEPENDENT-VARIABLES would be '(U1 V1 H1). Figure 6.17 shows a list quantification rule taken from rule-base G11-RB. Essentially the only difference between a production rule and a list quantification rule is that, on forward chaining, the production rule within the list quantification rule is always passed an initial bindings list, whereas a pure production rule need not have a bindings list passed to it. The initial bindings list for a list quantification rule is generated using the values of the list object. The values form the value of a bindings pair, and the bindings variable is $VALUE. Taking the example object DEPENDENT-VARIABLES given above, the initial bindings list, upon entering the production rule within the list quantification rule, given in Figure 6.17, would

be ...

(($VALUE U1))
(($VALUE V1))
(($VALUE H1)))

```
(RULE-Rulenumber
      (FOR ALL DEPENDENT-VARIABLES
        (IF   (Initial value for $value = $initial-value)
        (THEN (->Q1 ?[]= fiinit $value $initial-value))))
```

Figure 6.17: List quantification rule: System Rule Syntax

... this is assuming that no bindings list was already passed to the quantification rule. If the following bindings list was to be passed to the quantification rule ...

(($NAME JET))
(($NAME OUTLET1))
(($NAME OUTLET2)))

... the following bindings list would be used by the production rule within the list quantification rule ...

(($NAME JET) ($VALUE U1))
(($NAME JET) ($VALUE V1))
(($NAME JET) ($VALUE H1))
(($NAME OUTLET1) ($VALUE U1))
(($NAME OUTLET1) ($VALUE V1))
(($NAME OUTLET1) ($VALUE H1))
(($NAME OUTLET2) ($VALUE U1))
(($NAME OUTLET2) ($VALUE V1))
(($NAME OUTLET2) ($VALUE H1)))

The process of using each association list within the bindings list, to either prove or disprove an antecedent, will be covered in Section 6.9.3.

6.8.10 Rule firing modes

Figure 6.18 shows two methods of firing rule consequents. Figure 6.18a illustrates how each association list within the bindings list is applied to each consequent in turn, i.e. 1a, 1b, 1c, 2a, 2b, 2c. Figure 6.18b illustrates how the sequence can be altered such that each association list is used to complete the consequents, in turn, before using the next association list, i.e. 1a, 2a, 1b, 2b, 1c, 2c. Deviation from the default method of firing the consequents is performed by ensuring that the first consequent of the rule is ...

(APPLY BINDINGS TO EACH CONSEQUENT)

Upon successfully proving all antecedents within a rule, the function used to fire the rule consequents, FIRE-CONSEQUENTS, is passed a bindings list. This bindings list contains the necessary data required to fire each consequent. When specifying boundary conditions for a CFD analysis, data is grouped relative to a common reference, which is the boundary name. Extracting all of the necessary data to fully describe a boundary condition is performed while progressing through the antecedents of the rule. It is important to ensure that data for a particular boundary is successive within the bindings list, and as such is achieved through the use of the consequent ...

(FIRE IN BLOCK RELATIVE TO $NAME)

The last atom of this consequent need not be $NAME, but could be any other bindings variable. It is important that the chosen bindings variable is common within each association list. Indicating that such a grouping is to be performed, i.e. firing the consequents in block, causes the bindings list to be reduced to more than one bindings list. This facility is primarily used in G13-RB, and is illustrated in Figure 6.19.

6.9 Inference engine

The inference engine is the main reasoning mechanism within the system, and essentially filters data through the antecedents of the rules in order to apply the surviving data to the

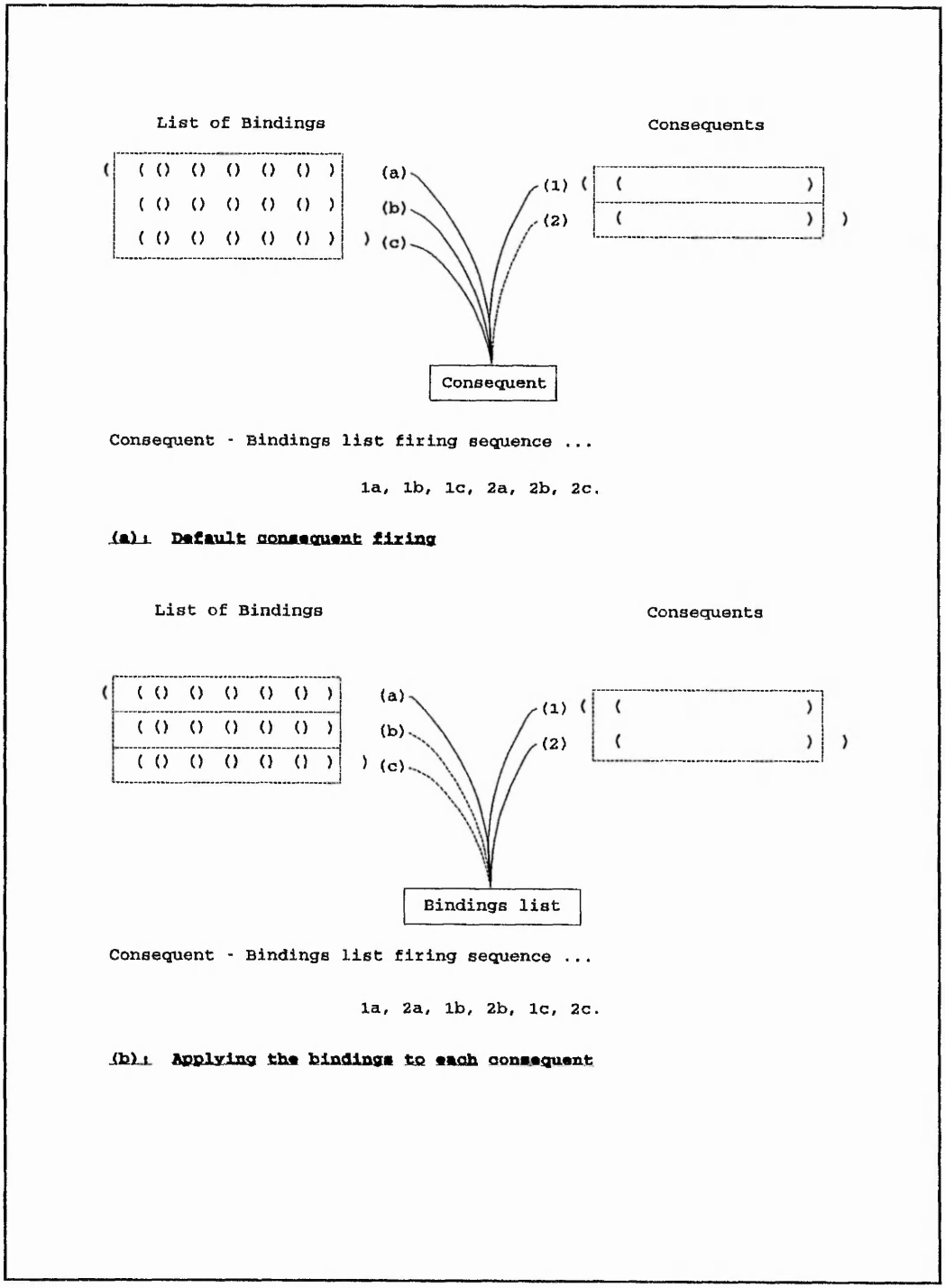


Figure 6.18: Consequent firing: (a) Default; (b) Applying the bindings list to each consequent

The bindings list before encountering the consequent ...

(FIRE IN BLOCK RELATIVE TO $NAME)

is ...

(((\$Type inlet) (\$Identity 1) (\$Nodes (1 2)) (\$Name jet)
(\$Phi V1)) (\$Co onlyms) (\$Val 0.0))
((\$Type inlet) (\$Identity 1) (\$Nodes (1 2)) (\$Name jet)
(\$Phi W1) (\$Co onlyms) (\$Val 2.1915))
((\$Type outlet) (\$Identity 1) (\$Nodes (4 8))
(\$Name outlet1)(\$Phi P1) (\$Co fixp) (\$Val 0.0))
((\$Type outlet) (\$Identity 1) (\$Nodes (3 4))
(\$Name outlet2)(\$Phi P1) (\$Co fixp) (\$Val 0.0)))

Upon leaving the bindings manipulation functions, there exists three separate bindings lists, each grouped relative to the value of the bindings variable ***Name***, thus ...

Block 1

(((\$Type inlet) (\$Identity 1) (\$Nodes (1 2)) (\$Name **jet**)
(\$Phi V1)) (\$Co onlyms) (\$Val 0.0))
((\$Type inlet) (\$Identity 1) (\$Nodes (1 2)) (\$Name **jet**)
(\$Phi W1) (\$Co onlyms) (\$Val 2.1915)))

Block 2

(((\$Type outlet) (\$Identity 1) (\$Nodes (4 8))
(\$Name **outlet1**)(\$Phi P1) (\$Co fixp) (\$Val 0.0)))

Block 3

(((\$Type outlet) (\$Identity 1) (\$Nodes (3 4))
(\$Name **outlet2**)(\$Phi P1) (\$Co fixp) (\$Val 0.0)))

Figure 6.19: Firing the consequents in block

consequents. It uses rules contained within rule-bases and implements forward and backward chaining. The system, upon compiling individual rule-bases, generates an inference network which links rules together to reduce unnecessary pattern matching processes performed by the inference engine. The inference networks prevent the

inference engine considering rules that have no bearing on the base rules within the particular rule-base. Forward chaining is initially performed on base rules, and backward chaining is implemented, where appropriate, in order to prove individual antecedents of the base rule. Figure 6.20 shows, diagrammatically, the structure of the inference engine, and highlights the developed functions and the order in which they are used. The functions given in bold are described with flow charts shown in Appendix F, and the LISP functions given in Appendix G.

The inference engine is given a rulebase on which to operate. Within each rulebase there exists a complete set of rules and an inference network. The engine initially forward chains on each of the base rules by utilising one of the functions **USE-FOR-ALL-RULE** or **USE-IF-THEN-RULE. USE-IF-THEN-RULE** is ultimately used in all instances. Forward chaining is commenced through the function **APPLY-FILTERS**, which simulates the filtration of data through the antecedents of a rule. For each antecedent, the initial bindings is NIL. The function **FILTER-BINDINGS-LIST** accepts a bindings list and each association list is, in turn, filtered using the function **FILTER-BINDINGS**. Prior to evaluating the antecedent, the pattern is instantiated with existing bindings, object values are inserted and any mathematical parsing is performed. This essentially creates a unique pattern with which to perform a preliminary evaluation and then, if appropriate, match with current assertions. The function **EVALUATE-ANTECEDENT** coordinates the evaluation of the antecedent. If the antecedent cannot be evaluated, or it cannot be matched with any antecedents, the backward chaining process is initiated. The strategies behind forward and backward will be discussed later.

6.9.1 Rulebases

Rule-bases are stored as LISP variables and are given meaningful names. The names must end in *-RB*, examples of which are CONVERSION-FACTOR-RB, POROSITY-DEFINITION-RB, DENSITY-THERMAL-DEPENDENCE-RB, VISCOSITY-THERMAL-DEPENDENCE-RB, and TMP1-EQUATION-RB.

Generation of the rule-bases into their final form is performed using two processes: loading rules into the rule-base and generating the inference network. Loading rules into a rule-base is performed using the macro REMEMBER-RULE, discussed in Section 6.8.1, and results in a LISP variable with the following structure:-

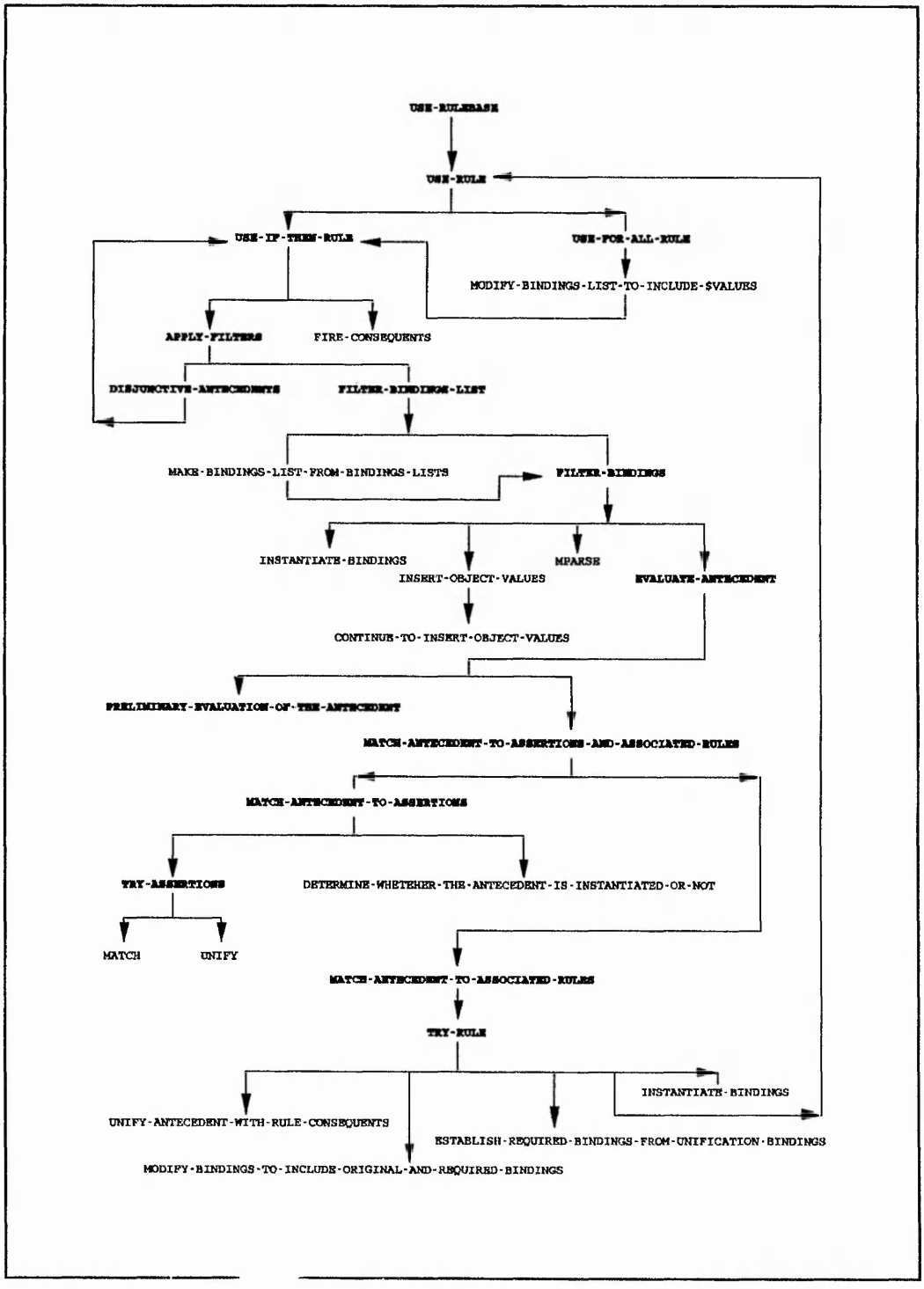


Figure 6.20: Inference Engine Logic

(RULE-1 RULE-2 ... RULE-n)

where ***n*** is the number of rules within the rule-base. Generating the inference network causes the LISP structure of the rule-base to alter such that the appropriate network is included. The resulting structure is as follows:-

(INFERENCE-NETWORK (RULE-1 RULE-2 ... RULE-n))

Manual alteration/modification of the network may be required in order to obtain desired linkages between rules.

6.9.2 Inference networks

Inference networks link rule antecedents with consequents of another rule. This is performed using pattern unification, described in Section 6.3, which matches the antecedent of a rule with the consequents of all other rules within that rule-base. If the antecedent and the consequent match, a link is formed. If the antecedent cannot be proved by either matching with an assertion or verifying an object, then an associated rule is used in backward chaining. If no associated rule exists, the rule fails. The purpose of introducing inference networks was to reduce the unnecessary consideration of rules.

Figure 6.21 shows how an inference network links rules together. The rules within each rule-base are referenced within the network as 1, 2, ..., n, where n is the total number of rules within the rule-base. Figure 6.22 shows how the rules given in Figure 6.21 are represented in an inference network which in turn is represented as a complex LISP structure.

Each rule in a network is represented by a two element list, the first of which is the rule number and the second a list containing associated rules, as given below:-

(*Rule-number* (*Associated-rules*))

For example, (2 ((1 ()))) indicates that rule number 2 has only one associated rule, which is rule number 1. However, rule number 1 has no associated rules because of the NIL list.

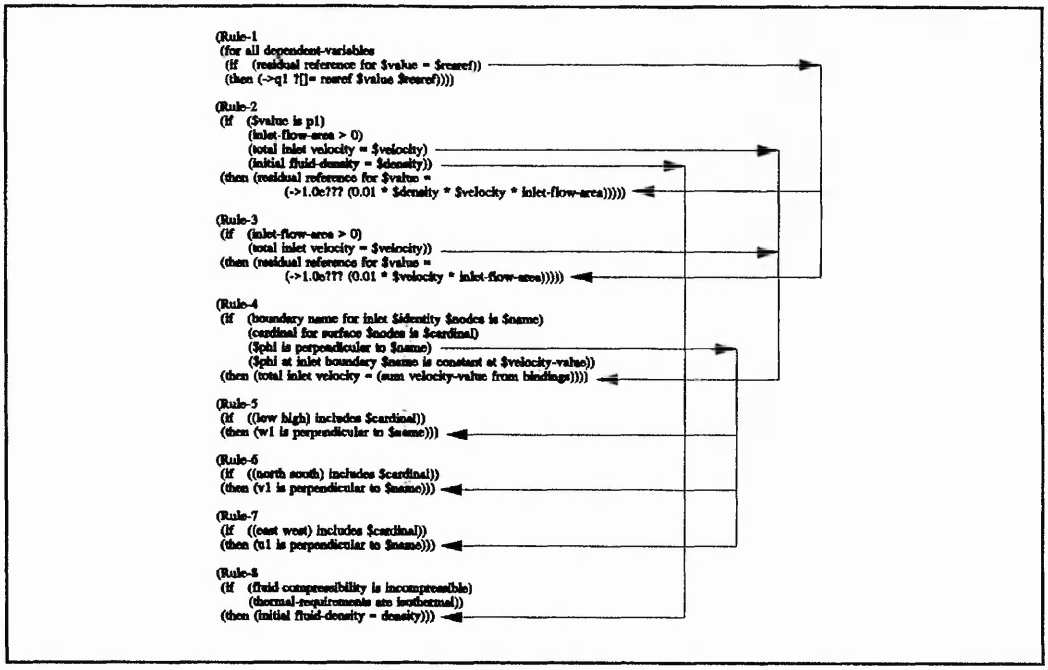


Figure 6.21: Inference Networks: Antecedent - Consequent linkages

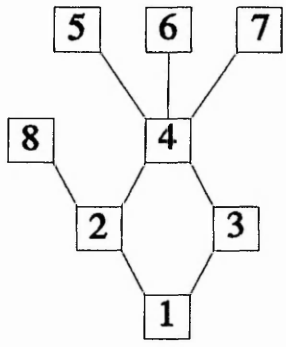
The definition of a base rule is a rule that has no link between any of its consequents and the antecedents of any other rule. Within a network these are the fundamental elements of the LISP structure. Therefore, a network is represented by:-

(BASE-RULE-1 BASE-RULE-2 ... BASE-RULE-n)

where **n** is the total number of base rules within the rule-base. Each base rule represented in the network then follows the two element list structure described above.

6.9.3 Forward chaining

The inference process initially commences with forward chaining on the base rules within the inference network, only backward chaining when necessary on associated rules. The pattern matching process is identical in both forward and backward chaining. Figure 6.23 indicates the sequences associated with the pattern matching process in forward chaining.



(a): Abstract Inference Network

```
( (1  ( (2 ( (8 ()  (4 ((5 ()) (6 ()) (7 ())) ) ) )
                    (3 ( (4  ((5 ()) (6 ()) (7 ())) ) ) ) ) ) )
```

(b): LISP representation of the Inference Network

Figure 6.22: Abstracted inference network

Initially any previously known bindings are inserted into the antecedent to make it unique for that particular set of bindings. Having effectively created a new antecedent, the templates contained within the assertions list are then individually matched with the new antecedent. The assertion templates are referred to as datums and the antecedent is the pattern.

Figure 6.24 illustrates the logic of the forward chaining mechanism. The inference engine performs forward chaining on all base rules within an inference network. Initiation of the forward chaining process is performed by calling the function USE-RULEBASE with the LISP command ...

(USE-RULEBASE *rulebase-name*)

USE-RULEBASE takes the specified rule-base and sequentially calls the function USE-RULE. This is performed until all the base rules have been considered. The function USE-RULE is fed with a two element list used to reference the base rule and its

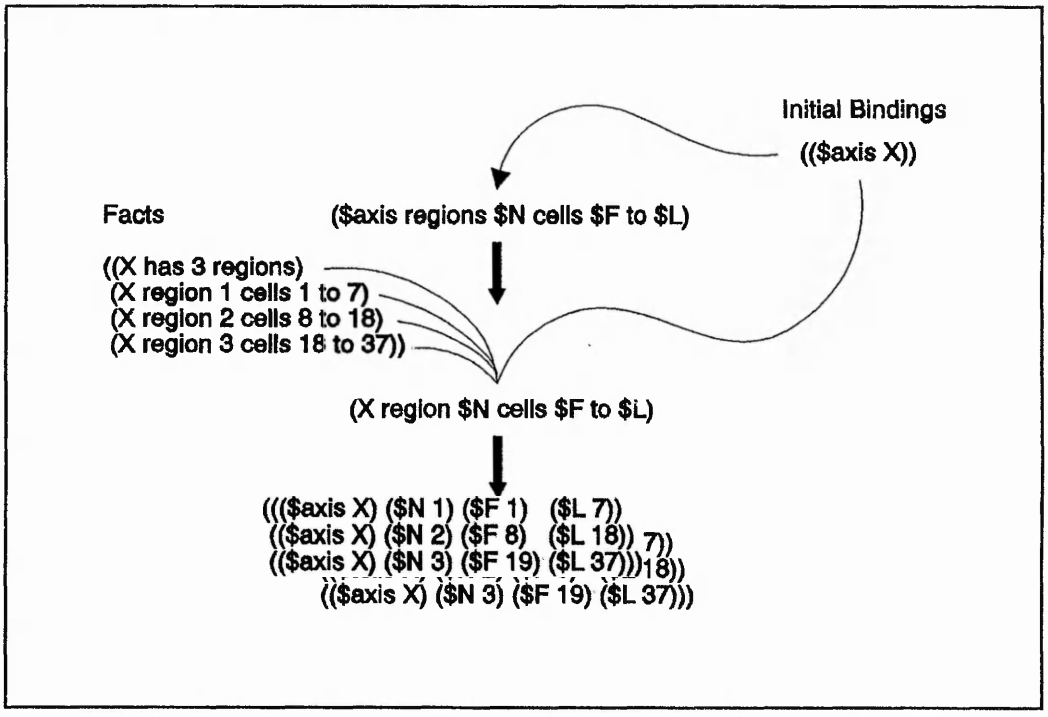


Figure 6.23: Pattern matching and forward chaining

relationship with all associated rules, discussed in Section 6.9.2. Along with this two element list, USE-RULE is also given a complete list of rules within the current rule base. USE-RULE checks to see if the rule is either a production rule or a list quantification rule, and calls either USE-IF-THEN-RULE, or USE-FOR-ALL-RULE respectively. USE-FOR-ALL-RULE generates the initial bindings list, and reduces the rule to a standard production rule, after which it calls USE-IF-THEN-RULE. The function USE-IF-THEN-RULE checks the antecedents of the rule for the default of (IF NOTHING). If this is the case, user syntax implied a consequent only rule and as such progresses on to firing the consequents. However, if antecedents are present then the filtering process commences with the first antecedent.

The forward chaining mechanism presented by Winston and Horn (1989) has been modified and extended for use within the system's inference engine. Extensions include conjunctive / disjunctive antecedents, objects, mathematical expressions, and multiple consequents. With the exception of changing the method of representing pattern variables in the pattern matching process and how assertions and bindings lists are represented, the

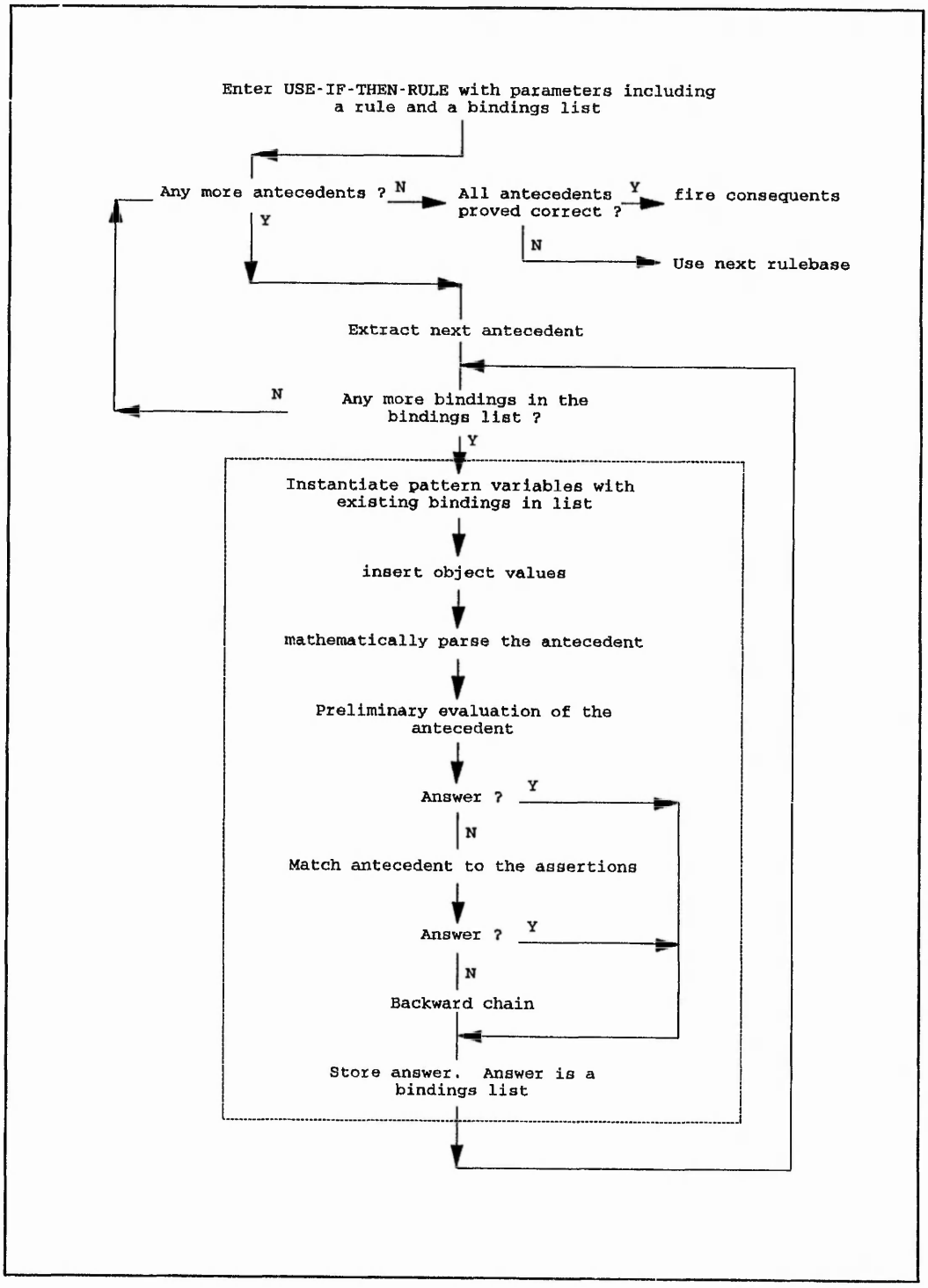


Figure 6.24: Forward chaining logic

assertions / antecedent filtering process is the same. The function **FILTER-BINDINGS** contains major modifications that need to be discussed. This function contains all of the essential forward chaining mechanisms, the logic of which is shown in Figure 6.24, contained within the dotted line. The function is passed an antecedent, along with an optional set of bindings, associated rules and a rulebase. Initially the antecedent is instantiated with any bindings that are applicable. This is illustrated in Figure 6.25. The new, modified, antecedent is then checked for any object names within the list. If a list atom corresponds to any object name, the system inserts its value in place of the name.

Antecedent: **(Cardinal for $nodes is $cardinal)**

Bindings: **(($nodes (1 2)) ($name jet))**

Antecedent after instantiating the bindings is ...

(Cardinal for (1 2) is $cardinal)

Figure 6.25: Bindings instantiation

When trying to insert an object's value into an antecedent, the system utilises the function **INSERT-OBJECT-VALUES**. Provided the antecedent is not enquiring whether an object is instantiated or not, the object's slots are accessed to help determine its value. If the object is uninstantiated or its status is volatile, the slots are used to try and determine the value. The order of slot usage is: fixedvalue, computevalue and rulebase. A fixedvalue automatically instantiates an object. An entry in the computevalue slot implies that a valid LISP command or function needs to be executed to determine the object value. Finally, a rulebase slot instantiated as **T** suggests that there exists a knowledge base upon which inferencing is to be performed. Failure to use any of these slots forces the system to ask the user for the object's value directly, using the function **ASK-OBJECT**.

Having obtained the object's value, and correctly inserting it in the antecedent, complete mathematical parsing is performed. This is implemented to reduce any resident mathematical expressions to a single value. Mathematical parsing, relative to the LISP

system, is discussed in Section 6.9.6.

Once the antecedent is in the fully modified form, evaluation commences with the function **EVALUATE-ANTECEDENT**. Initially the antecedent undergoes a preliminary examination to assess whether it is checking the validity of a statement. For example, enquiring if an assertion or object is instantiated, or if a statement is true or false in either equality or inequality. Examples of such antecedents are ...

(Coordinates are instantiated)
(Coordinates are cylindrical)
(Cardinal for surface (1 2) is instantiated)
(Number-of-dimensions = 2)

... where Coordinates and Number-of-dimensions are defined objects. The third statement, given above, is derived from the assertion template **(Cardinal for surface $nodes is $cardinal)**. For each of these antecedents the preliminary evaluation would return either True (T) or False (NIL), depending on the validity. A true return would force the system to retain the existing set of bindings and transform them into a bindings list to be returned to the function **FILTER-BINDINGS-LIST**. A false, NIL, answer would return NIL to the function **FILTER-BINDINGS-LIST**. This process is repeated for each antecedent in the rule. If the antecedent did not conform to the requirements of the preliminary evaluation the antecedent would be returned, thus indicating that the filtering process needed to commence, using the function **MATCH-ANTECEDENT-TO-ASSERTIONS** called from within the function **MATCH-ANTECEDENT-TO-ASSERTIONS-AND-ASSOCIATED-RULES**. Matching the antecedent to the assertions is essentially the same process as defined in Winston and Horn (1989), with the exception that the blackboard structure adopted for the assertions list only requires a match with the template to be performed. A successful match immediately extracts the data from the assertions under that template. The need to continue through the assertions list looking for other valid matches is removed due to the categorisation of the assertions.

If the antecedent cannot be proved to be correct in forward chaining, thereby losing the bindings carried forward into the function **FILTER-BINDINGS**, backward chaining is initiated through the use of the function **MATCH-PATTERN-TO-ASSOCIATED-RULES.**

6.9.4 Backward chaining

The backward chaining mechanism is invoked when an antecedent of the current rule cannot be proved. If the antecedent is a statement that is given a true or false answer during its preliminary evaluation, or if the antecedent matches an assertion, and an answer results, backward chaining is not performed. An antecedent that conforms to a template can only be used for backward chaining. If the antecedent does not match any of the templates given as assertions, it is assumed that there exists a rule within the rulebase whose consequents will provide the relevant match. Under these circumstances, each of the associated rules are tried for successful unification with the antecedent, if no unification results the rule fails. However, as soon as a rule is located whose consequents provide the required link with the antecedent, backward chaining ceases and the rule is made unique relative to the bindings, prior to forward chaining. In order to explain the processes employed by the system during backward chaining, we shall consider two separate situations, depicted in Figure 6.26 and Figure 6.27.

Figure 6.26 indicates an isolated view of three objects: **Dependent-variables, Fluid-compressibility** and **Thermal-requirements**. The assertions list has been reduced to show specific information and relevant data to be used in the rulebase G15-RB. The rulebase has been reduced to one base rule and its associated rules for the purpose of explaining the inference process. The inference network is also given.

Forward chaining commences with RULE-1. Upon entering the function USE-RULE, hence USE-FOR-ALL-RULE, there exists no bindings list. However, because this is a list quantification rule, the system generates a bindings list according to the list object, **Dependent-variables**, and its associated value. This leads to the bindings list ...

```
(($VALUE P1))
(($VALUE U1))
(($VALUE V1)))
```

... being passed to the function USE-IF-THEN-RULE in conjunction with other required parameters. Each of the association lists within the bindings list, given above, is applied to each antecedent in the rule. All bindings are filtered through one antecedent before progression onto the next. In this case there is only one antecedent. The function

Objects: Dependent-variables: (P1 U1 V1)
Fluid-compressibility: Incompressible
Thermal-requirements: Isothermal

Assertions: ((( )

((Boundary name for $type $identity $nodes is $name)
(Boundary name for inlet 1 (1 2) is jet))
((Cardinal for surface $nodes is $cardinal)
(Cardinal for surface (1 2) is low))
(($dependent-variable at $type boundary $name is $condition at $quantity)
(w1 at inlet boundary jet is constant at 2.1915)))

Rulebase: (Modified from the actual rulebase to illustrate the inference network and backward chaining only. The first and last base rules have been omitted.)

```
(Rule-1
 (for all dependent-variables
  (if   (residual reference for $value = $resref))
  (then (->q1 ?[]= resref $value $resref))))

(Rule-2
 (if   ($value is p1)
       (inlet-flow-area > 0)
       (total inlet velocity = $velocity)
       (initial fluid-density = $density))
 (then (residual reference for $value =  (->1.0e??? (0.01 * $density * $velocity * inlet-flow-area)))))

(Rule-3
 (if   (inlet-flow-area > 0)
       (total inlet velocity = $velocity))
 (then (residual reference for $value =  (->1.0e??? (0.01 * $velocity * inlet-flow-area)))))

(Rule-4
 (if   (boundary name for inlet $identity $nodes is $name)
       (cardinal for surface $nodes is $cardinal)
       ($phi is perpendicular to $name)
       ($phi at inlet boundary $name is constant at $velocity-value))
 (then (total inlet velocity = (sum velocity-value from bindings))))

(Rule-5
 (if   ((low high) includes $cardinal))
 (then (w1 is perpendicular to $name)))

(Rule-6
 (if   ((north south) includes $cardinal))
 (then (v1 is perpendicular to $name)))

(Rule-7
 (if   ((east west) includes $cardinal))
 (then (u1 is perpendicular to $name)))

(Rule-8
 (if   (fluid-compressibility is incompressible)
       (thermal-requirements are isothermal))
 (then (initial fluid-density = density)))
```

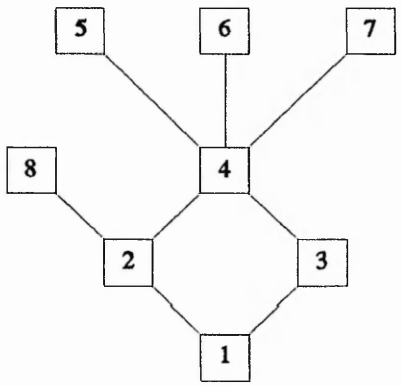


Figure 6.26: Rules, inference network and data used to illustrate backward chaining only

Base rule antecedents:

```
        (Rule-1
          (if   (Boundary name for $type $identity $nodes is $name)
                (Cardinal for surface $nodes is $cardinal)
                .
                .
(A) -------->   ($name x cells are $ixf to $ixl)
                .
                .
                (          ))
          (then (          )))
```

At **(A)** the bindings list for the inlet boundary is ...

(($type inlet) ($identity 1) ($nodes (1 2)) ($name jet) ($cardinal low))

Associated rule:

```
        (Rule-n
          (if   ((west low south) includes $cardinal)
                .
                .
                .
                ($axis region 1 cells $if to $il))
          (then ($name $axis cells are $if to $if)))
```

Figure 6.27: Antecedents, bindings and rule used to illustrate data transition through inference networks

FILTER-BINDINGS initially uses the bindings (($VALUE P1)) when trying to prove the antecedent. As described in section 6.9.3, the antecedent has all pattern variables instantiated with the values of corresponding pattern variables in the association list. In this case the antecedent becomes ...

(residual reference for P1 = $resref)

Insertion of object values and mathematical parsing are performed, these processes have no significance with this antecedent. Preliminary evaluation of the antecedent fails

because it is not a statement. This forces the system to try to match current assertions with the antecedent. This, again, fails because it does not conform to any assertion template. At this stage the system calls the function MATCH-ANTECEDENT-TO-ASSOCIATED-RULES. As can be seen from the inference network, RULE-1, is only linked to RULE-2 and RULE-3 through antecedent-consequent unification. In this case both associated rules are linked to the same antecedent, because there is only one. However, if there existed more than one antecedent, all associated rules need not necessarily be linked to the same clause. The function MATCH-ANTECEDENT-TO-ASSOCIATED-RULES controls the process of trying each associated rule. Backward chaining is performed with the function TRY-RULE, the logic of which is shown in Figure 6.28. The antecedent passed to TRY-RULE, **(residual reference for P1 = $resref)**, has already been instantiated with the bindings (($value P1)). Rule 2 is now instantiated with the same bindings, as shown in Figure 6.29. A set of unification bindings is now obtained through unifying the rule consequents with the antecedent, this gives ...

(($resref (->1.0e??? (0.01 * $density * $velocity * inlet-flow-area))))

The initial bindings passed to TRY-RULE are modified, this is discussed under the second example illustrated by Figure 6.27, and the required bindings are determined. A required binding is defined as any pair within the unification bindings whose second element is itself a pattern variable or a list. The latter forms a system function, as given above, or a bindings manipulation consequent.

Once the required variables are determined the rule is instantiated with the unification bindings. The rule has now been made unique with respect to the current bindings list passed to the function TRY-RULE.

After completing the inferencing on the rule, and any other backward chaining deeper into the inference network, the bindings in the resulting bindings list are each condensed to include only the original bindings and the required bindings. This is performed using the function MODIFY-BINDINGS-TO-INCLUDE-ORIGINAL-AND-REQUIRED-BINDINGS. It is within this function that the pattern variables within the required bindings are matched with the appropriate value from the bindings, which is to be returned as part of the resulting bindings list. Furthermore, any bindings manipulations and evaluation of system functions through object value insertion and mathematical

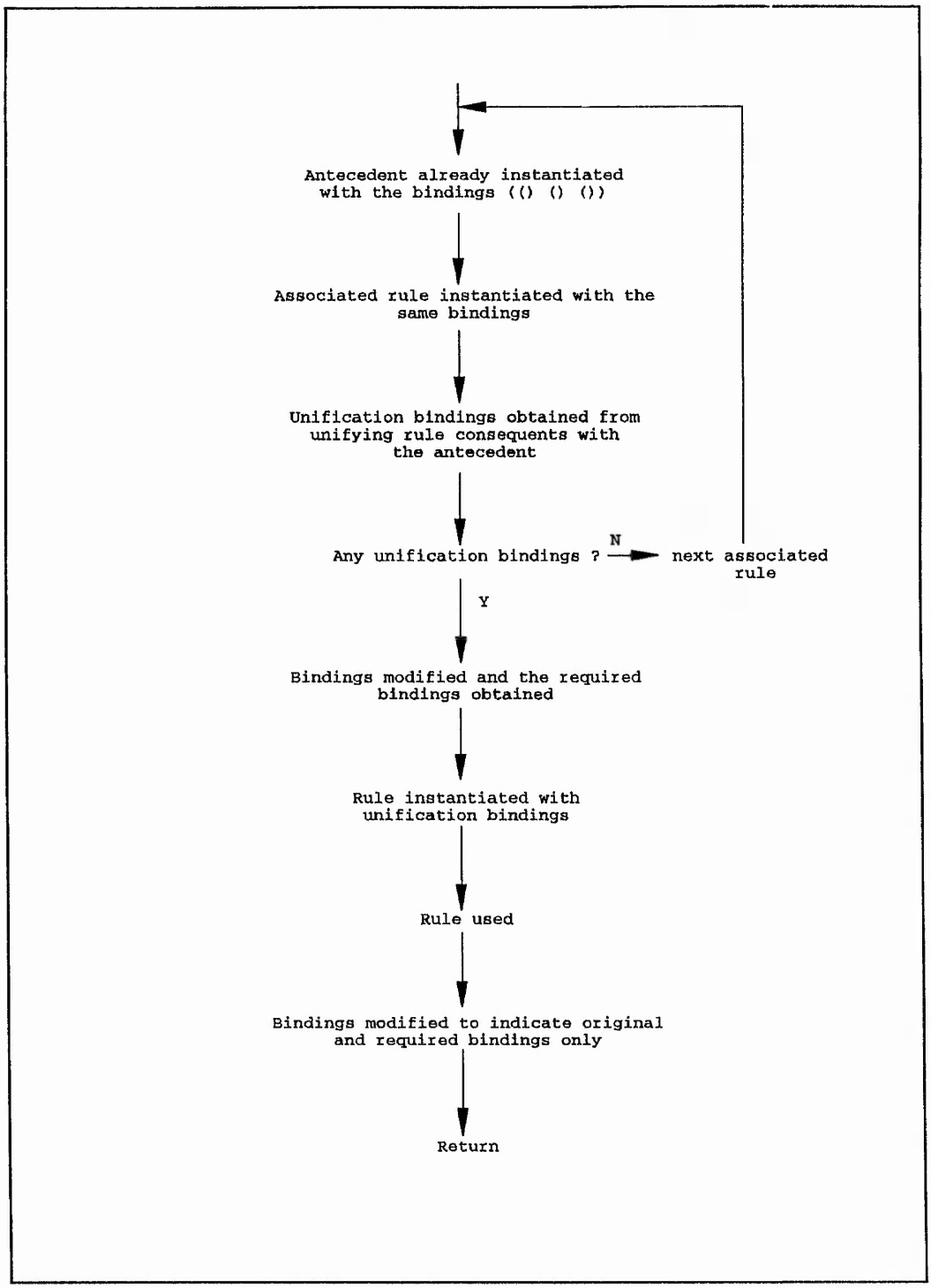


Figure 6.28: Logic associated with the function TRY-RULE

```
(RULE-2
      (IF     (P1 is P1)
              (Inlet-flow-area > 0)
              (Total inlet velocity = $velocity)
              (Initial fluid-density = $density))
      (THEN (Residual reference for P1 =
                    (->1.0??? (0.01 * $density * $velocity
                                          * Inlet-flow-area)))))
```

Figure 6.29: Backward chaining: Bindings instantiation within an associated rule

parsing are performed. This is exemplified by the required bindings list in the following example.

Forward chaining performed on the antecedents presented in Figure 6.27 establishes the following set of bindings upon reaching the antecedent indicated with arrow A ...

(($TYPE INLET) ($IDENTITY 1) ($NODES (1 2))
($NAME JET) ($CARDINAL LOW))

Instantiating the antecedent with the above bindings produces ...

(JET X CELLS ARE $IXF TO $IXL)

This antecedent cannot be proved, therefore backward chaining on the associated rule commences. After instantiating the associated rule with the bindings we have ...

```
(if       ((west low south) includes low)
                    .
                    .
          ($axis region 1 cells $if to $il))
(then (jet $axis cells are $if to $if))
```

The unification bindings are (($AXIS X) ($IXF $IF) ($IXL $IF)). The required bindings are established from these unification bindings, and are given by (($IXF $IF) ($IXL $IF)). Also the actual bindings are modified to include unification bindings pairs whose second element is itself not a list or pattern variable. This is necessary to ensure that continuity within the rules is maintained when moving further into the inference network, i.e. towards the leaf rules. This results in the bindings becoming ...

(($TYPE INLET) ($IDENTITY 1) ($NODES (1 2))
($NAME JET) ($CARDINAL LOW) ($AXIS X))

The rule is instantiated with the unification bindings, giving ...

```
(if   ((west low south) includes low)
                  .
                  .
      (x region 1 cells $if to $il))
(then (jet x cells are $if to $if))
```

Forward chaining is performed on the associated rule with the bindings being passed as a bindings list, thus ...

(((($TYPE INLET) ($IDENTITY 1) ... ($AXIS X))).

Upon completion of forward chaining, a bindings list is returned whose principal pairs within each set of bindings are the original bindings, as well as other pairs obtained during inferencing. These extra bindings would consist of the remaining pattern variables in the rule, along with their corresponding values. For example, proving all antecedents to be correct might result in ..

```
(((($TYPE INLET) ($IDENTITY 1) ... ($AXIS X) ($IF 1) ($IL 1))
                                .
                                .
((              ) (              ) ... (        ) (      ) (      )))
```

... noting that the last two pairs in each set of bindings gives values for $IF and $IL. With $IF being the subject of the required bindings, (($IXF $IF) ($IXL $IF)), the values of $IXF and $IXL can be determined. Therefore, TRY-RULE would return the original bindings along with the required bindings as a bindings list, thus ...

(((\$TYPE INLET) (\$IDENTITY 1) (\$NODES (1 2))
(\$NAME JET) (\$CARDINAL LOW) (\$IXF 1) (\$IXL 1))).

6.9.5 Bindings transition through inference networks

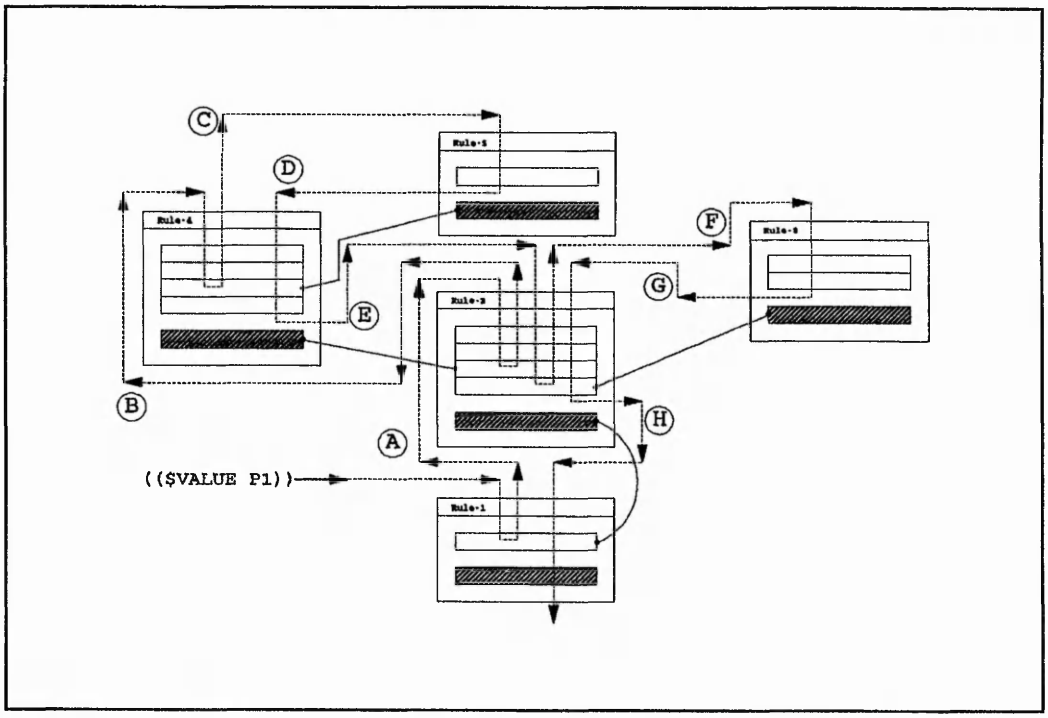


Figure 6.30: Bindings transition through inference networks

Chaining through the inference network only permits bindings to be carried deeper into the network. Whenever the system backtracks to lower level rules the bindings information gathered for that rule is not carried back, only the original bindings and the required bindings are. Figure 6.30 illustrates the inference network for rules 1, 2, 4, 5 and 8 given in Figure 6.26. Each rule is presented as a box within which is contained the number of antecedents and consequents for the appropriate rule, shown as empty and

hatched boxes respectively. The solid lines represent the links between antecedents and consequents, whereas the dotted line shows the path taken by the inference engine. Points A to H will be used to describe the status of the bindings, required bindings and the bindings lists established through inferencing on each rule. The rules are given in Figure 6.26

At (A), backward chaining has commenced from rule 1 to rule 2 in order to prove the only antecedent. The bindings have been converted to a bindings list with which forward chaining is to commence, thus ...

(((\$VALUE P1))),

and the required bindings are ...

((\$RESREF (->1.0e??? (0.01 * \$DENSITY * \$VELOCITY * INLET-FLOW-AREA))))

The first two antecedents of rule 2 are successfully proved, as these are statements, therefore not altering the bindings. The bindings list at (B) is the same as at (A), and backward chaining on the third antecedent establishes the required bindings for rule 2 as ((\$VELOCITY (SUM VELOCITY-VALUE FROM BINDINGS))). Forward chaining on rule 4 is interrupted on its third antecedent, whereby further backward chaining on rule 5 yields the following required bindings: ((\$PHI W1)). The bindings list at point (C) is ...

(((\$VALUE P1) (\$IDENTITY 1) (\$NODES (1 2)) (\$NAME JET)
(\$CARDINAL LOW))).

At point (E), rule 5 has been successful, and rule 4 is complete, therefore, the bindings list is ...

(((\$VALUE P1) (\$IDENTITY 1) (\$NODES (1 2)) (\$NAME JET)
(\$CARDINAL LOW)) (\$PHI W1) (\$VELOCITY-VALUE 2.1915))).

The required bindings for rule 2 states that the pattern variable \$VELOCITY should be : (SUM VELOCITY-VALUE FROM BINDINGS). This means that the bindings list returned from rule 4 needs to be manipulated so that a single value for \$VELOCITY

would result. Only one set of bindings is within the bindings list, forcing the required pattern variable, $VELOCITY, to the calculated value 2.1915. At (F) the bindings list consists of ...

(((\$VALUE P1) (\$VELOCITY 2.1915))),

and the required bindings for rule 2 has been altered to suit the needs of the fourth antecedent, thus: ((\$DENSITY DENSITY)). The fourth antecedent of rule 2, (INITIAL FLUID-DENSITY = \$DENSITY), caused problems during rule validation because the atoms FLUID and DENSITY were not hyphenated. The atom DENSITY was treated to be identical to the name of an object DENSITY, thus causing the system to insert the appropriate object value making the antecedent senseless, i.e. (INITIAL FLUID 1.225 = \$DENSITY). This indicated that atoms within antecedent templates should not coincide with object names unless intended. To avoid this problem the atoms FLUID and DENSITY were hyphenated. The antecedents of rule 8 are statements thus causing the bindings list to be unchanged, and the resulting bindings list is ...

(((\$VALUE P1) (\$VELOCITY 2.1915))).

At (H) the returned bindings list contains the required pattern variable, $VELOCITY, and its inferred value 2.1915. These items are being returned to the base rule as a bindings list, thus ...

(((\$VALUE P1) (\$VELOCITY 2.1915) (\$DENSITY 1.225))).

The consequents of the base rule, rule 1, can now be fired with the bindings list resulting from the inference process.

6.9.6 Mathematical parser

The need for a mathematical parser has already been presented, Section 5.4.3. The need was further substantiated during the development of the rule base language using LISP. Mathematical processing in LISP is possible but requires a rather unorthodox syntax with which to present the expression. For example, the expression 4 * 3 + 5 would need to be written as (+ 5 (* 4 3)). The more complex the expression, the more daunting the LISP

equivalent. Furthermore, if the rulebase language was to permit the inclusion of mathematical expressions in both the antecedents and consequents, it would be necessary to enter the expression as a user would enter it, and not as a LISP expression. To this end, a LISP mathematical parser was developed. Appendix H lists the LISP mathematical parser which implements recursion. Again standard precedence laws and associativity rules were utilised. A comparison with the equivalent FORTRAN parser, given in Appendix D, illustrates the power of LISP.

6.10 Presentation facilities

The presentation facilities created for the KBFE revolve around system / user dialogue. There exists two methods with which the system can ask the user for information: defining an explicit ask command within the rule consequents, or the system automatically asks for information when nothing can be inferred.

The presentation facilities for objects involve basic menu interfaces, whereas asking the user for information to assert a fact consists of a single line of text which prompts the user to enter a value. Both of these methods, shown in Figure 6.31, use the same presentation facilities.

Defining an explicit ASK command within a rule consequent can be used for either objects or assertions. The use of the ASK command for assertions has been discussed in section 6.8.6. However, for objects, the system makes extensive use of some of the object slots within the presentation facilities. The object slots are: Preface, Disallowedvalues, Allowedvalues, Defaultvalue, Prompt, and Status. The slots for Allowedvalues, Status, and Prompt can be altered / modified from within rule consequents using the template ...

(OBJECT SLOT-NAME VALUE)

... where OBJECT is the object name, i.e. porosity, density etc., SLOT-NAME is either Allowedvalues, Status or Prompt. VALUE is the new value to be assigned to the particular object slot. Even though the system only allows the three slots to be modified at this stage, it would be easy to include any of the remaining slots in the list of those whose values may be allowed to vary. The function INSTANTIATE-OBJECT performs the modifications relative to the information contained within the rule consequents. The

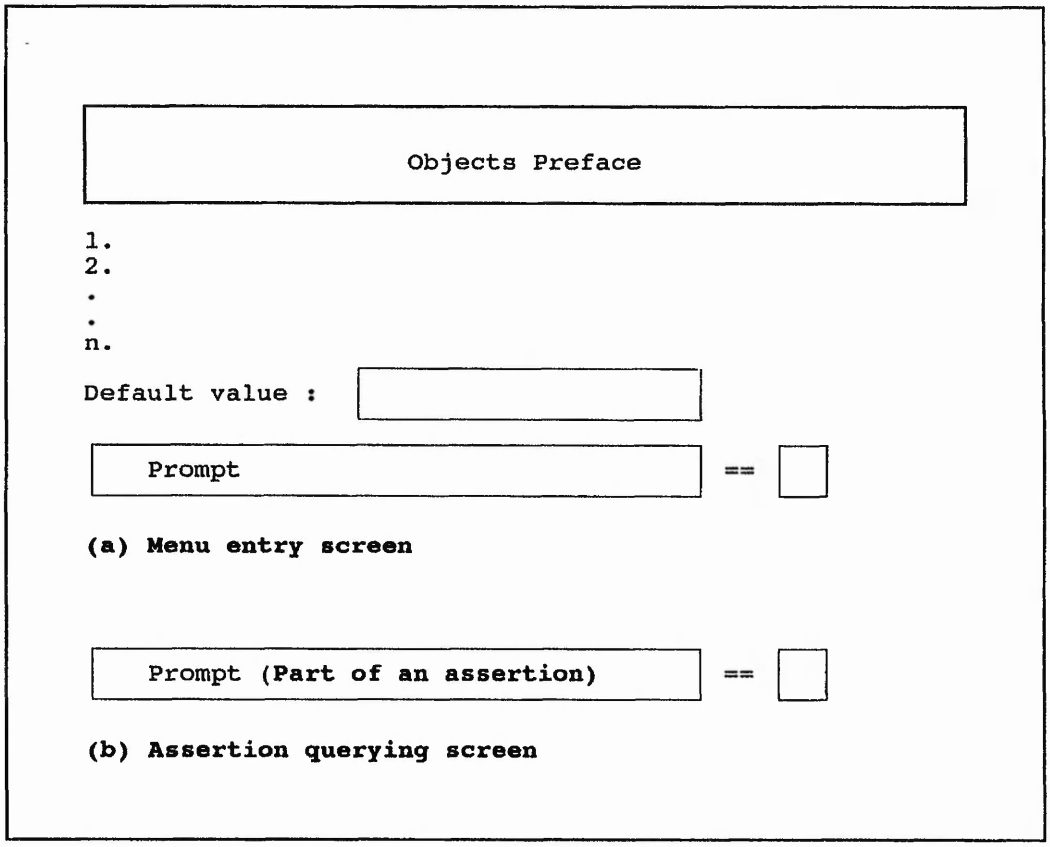


Figure 6.31: Presentation facilities: (a) menu entry screen; (b) Assertion querying screen

slot Allowedvalues can only be replaced with a list of alternatives, i.e. (option-1 option-2 ... option-n), and Status can be either FIXED or VOLATILE. The need for altering the prompt slot was brought about because of the potential volatile nature of the object POROSITY. For each blockage within a domain, if the porosity was to vary from obstruction to obstruction, there needed to be a method of differentiating between obstructions when asking for the porosity value. Rulebase G11-RB accommodates the possibility of using a potential volatile object, POROSITY, and illustrates the usage of consequents to alter object slots.

The Preface slot utilises a method of relating data values to a phrase template, similar to that used in MYCIN, Shortliffe (1975). This allows the preface to dynamically alter according to the values of other objects within the database. The structure of the Preface slot is ...

(Template Value-1 Value-2 ... Value-n)

where the template has marked positions within the text for the location of Value-1, Value-2, ..., Value-n. However, Value-1, Value-2 to Value-n may be omitted. The positions of the values are indicated with a number in parentheses, the number being 1, 2, ..., n, corresponding to Value-1, Value-2, ..., Value-n respectively. For example ...

((1) is (2) dependent and follows the relationship (3))

... where (1), (2) and (3) could be different values which would enable different facts to be presented. For example, depending upon the type of analysis required by a user, who might wish to specify that the kinematic viscosity of a fluid is temperature dependent. Under these circumstances, PHOENICS allows the user to specify either one of the two PHOENICS defined relationships between temperature and viscosity or a user model. The latter is entered through GROUND coding, which has not been included within the KBFE. The two PHOENICS predefined equations relate kinematic viscosity to temperature via A+BT or A+BT+CT**2, and requires the instantiation of coefficients A, B and/or C, given as PHOENICS variables ENULA, ENULB, and ENULC respectively. The objects ENULA, ENULB, and ENULC in the KBFE database use the template/value relationships in their preface slots. It was also found necessary to establish a method of recognising which equation was required, performed through inferencing, and allowing the preface of the objects used for the coefficients to change depending on the chosen equation. This was achieved by defining the Preface slots of ENULA, ENULB and ENULC as ...

((Viscosity equation - (1)) viscosity-equation).

During inferencing the system would ask the user which relationship they would like to use, thus instantiating the object VISCOSITY-EQUATION. Furthermore, rules within the rulebase, FLUID-RB, determine which coefficients to ask for, thereby forcing the contents of the preface slots to be printed to the screen above the user prompt, see Figure 6.31a. The function PRINT-TO-SCREEN is used to present the Preface to the user in its final form, prior to this several stages are executed. Initially the structure is instantiated with any object values, in this case, assuming that VISCOSITY-EQUATION is instantiated with A+BT, the structure would become ...

((Viscosity equation - (1)) A+BT)

This template is then modified such that the markers are replaced with their corresponding values, thus ...

(Viscosity equation - A+BT).

The resulting list is then transformed to strings and printed onto the screen. The transformation process takes into account the predefined width of the screen and truncates the text to fit within the range. The text then commences on the next line. This process continues until the entire preface has been presented to the user.

The sequence, operating on a more complex template, would be ...

```
(    ((1) (2) is related to (5) via the (4) relationship (3))
     kinematic viscosity A+BT+CT**2 quadratic temperature   )
```

This, when manipulated, would present the following statement ...

Kinematic viscosity is related to temperature via the quadratic relationship A+BT+CT2**

Using the same template, and different values ...

```
(    ((1) (2) is related to (5) via the (4) relationship (3))
     NIL Temperature A+BT linear enthalpy  )
```

yields ...

Temperature is related to enthalpy via the linear relationship A+BT

6.11 Extensibility

The fundamental structure of the database, knowledge bases, and the inference engine of the LISP KBFE have been developed with extensibility as an important factor. The assertions templates consist of groups of symbols which have specific meanings for CFD problems. The inclusion of other templates would automatically cause the inference engine to consider their contents during forward chaining. Furthermore, if their context were to be changed then any other knowledge domain could be considered. There is a similar argument for the objects contained within the system. The quantity of objects could be increased such that the breadth of the knowledge domain increases.

The ability to remove the existing objects, assertions, and knowledge bases, and to replace them with equivalent counterparts relating to a different domain is a measure of the extensibility of the system.

6.12 Conclusions

This chapter tried not to assume any prior understanding of LISP by the reader, and as such, the principles of standard pattern matching and pattern unification were presented. These processes are used to form the basis of the inference engine developed for the KBFE. The inference engine initially uses forward chaining on a rulebase. The rulebase contains, apart from the rules, an inference network which is used when backward chaining is required.

The overall system architecture was presented, and the interactions of the inference engine, rulebases and the database highlighted. LISP functions were used to provide the user interface and data manipulation functions. User interface functions were developed to provide basic dialogue sessions with the user throughout inferencing, which consists of menus and prompting the user to enter data to assert factual information. The data manipulation functions were required (a) to gather initial information from the user regarding the geometry of the analysis to be performed, and (b) to manipulate this information to assert facts which were to be used during inferencing on subsequent rulebases. The functions written in the C language were included to increase the numerical processing power of the system for the grid generation techniques.

The database consisted of two forms of data: assertions and objects. Assertions were used for geometrical and boundary condition information, and objects were used for PHOENICS variables and other specific data. Objects were created using a LISP structure which essentially created a frame with slots. The slots consisted of the structure fields, and were used to guide the inference engine through the possible methods of establishing a value for the object, i.e. FixedValue, AllowedValues, ComputeValue, Rulebase. Certain slots were used to store data related to the presentation facilities developed within the system. The assertions were categorised using templates, each of which could have applied to it varying items of data, thus representing different boundaries. This categorisation lead to a modular assertions list, referred to as a blackboard throughout. This blackboard is not the same as those described by Hayes-Roth (1985), and Reddy and O'Hare (1991), in so much as that it does not contain linkages between entries on the same or different levels. Furthermore, it cannot pass data between the different levels. Even so, the technique reduced the number of pattern matches required to consider all the assertions, thus improving the efficiency of the inference engine.

The rulebases consist of a set of rules, written in a developed language. There exists two types of syntax for the rules: User Rule Syntax (URS) and System Rule Syntax (SRS). The URS allows a little more flexibility when defining the rules within each base, whereas the system creates the SRS from the URS. The SRS must conform to a predefined structure. There are two types of rule in the SRS: production rules and list quantification rules. List quantification rules are rules that use list objects upon normal production rules. Production rules allow either disjunctive or conjunctive antecedents with multiple consequents. Various firing modes have been developed to suit different synthesis rules, these determine the order of firing the bindings on the rule consequents. Furthermore, block firing of bindings can be performed relative to reference data. The rulebases not only contain the list of rules, but also an inference network that links the rules together. Rules that do not have any links from their consequents to the antecedents of other rules are classified as base rules. It is these base rules that are initially used when a new rulebase is being considered by the inference engine. Forward chaining on the base rules is followed by backward chaining, when required, on any associated rules. Data management in backward chaining allows data to be carried forward deeper into the network, but information obtained within the network cannot be brought back out of the network, with the exception of the original data.

Finally, the presentation facilities developed for the system are described. These facilities are used by the inference engine to query the user for data entry. Menus and user prompting are the two techniques used, implemented for objects and assertion data gathering respectively. Generalisation of object prefaces has been performed using a modified technique developed for the MYCIN project, Shortliffe (1975).

CHAPTER 7

CASE STUDIES

7.1 Introduction

This chapter illustrates the interaction of the user with the KBFE by presenting the dialogue sessions for two case studies. For each study, a brief description of the geometry, and required analysis will be given before presenting the listed interaction with the system. The final data file, and the results of submitting the run to PHOENICS, will be presented.

PHOENICS requires certain data to be entered in order to completely define an analysis. The essential data required by PHOENICS, which is created by the KBFE through inferencing and user interaction, consists of ...

Preliminary information: This is used to define variables used within the data file, and the standard commands that have to be present for PHOENICS to successfully execute the data file. Furthermore, messages can be incorporated into the data file by inserting at least two spaces before typing anything on the line of the data file.

Grid definition: The grid definition is synthesised using the calculated aspect ratio dependent grid generation data. Initial interaction with the user establishes the nodal coordinates and their connectivities which are used to establish the initial assertions from which subsequent inferencing is performed.

Dependent variables: The dependent variables for the analysis consists of pressure and velocities. Depending on the type of analysis, turbulence properties, and temperatures may also be required. The rulebase DEPENDENT-VARIABLES-RB determines which variables to include following user interaction.

Fluid properties: Inferencing on the rulebase FLUID-RB provides the required fluid properties.

Initial values: PHOENICS is a finite volume package which requires an initial best guess of dependent variable results. It is from this initial best guess that the iteration cycles commence until convergence is achieved. The rulebase G11-RB is used to establish the best guess initial values.

Boundary conditions: Depending on the type of analysis required by PHOENICS, different boundary conditions would be required. The rulebase BC-RB is used to determine, through inference, what type of boundary condition information is required. The data obtained through inference is stored as assertions. The assertion data is synthesised using the rulebase G13-RB.

Solution algorithm control parameters and result presentation commands: These are determined through inference and intelligent defaulting using the rulebases G15-RB, G16-RB and G17-RB.

Depending on the complexity of a problem, the size of the resulting data file could vary from, say 30 lines, to, in excess of, 300 lines. Figure 3.6 indicates a simple data file with the seven specific areas highlighted. The synthesis of the commands from the entered and inferred data is performed using the rulebases G1-RB, G2-RB, ..., G24-RB.

The two case studies that were chosen are: Two dimensional thermal jet impingement (Turbulent), and a two dimensional axisymmetric orifice plate.

7.2 Two dimensional thermal jet impingement - Turbulent

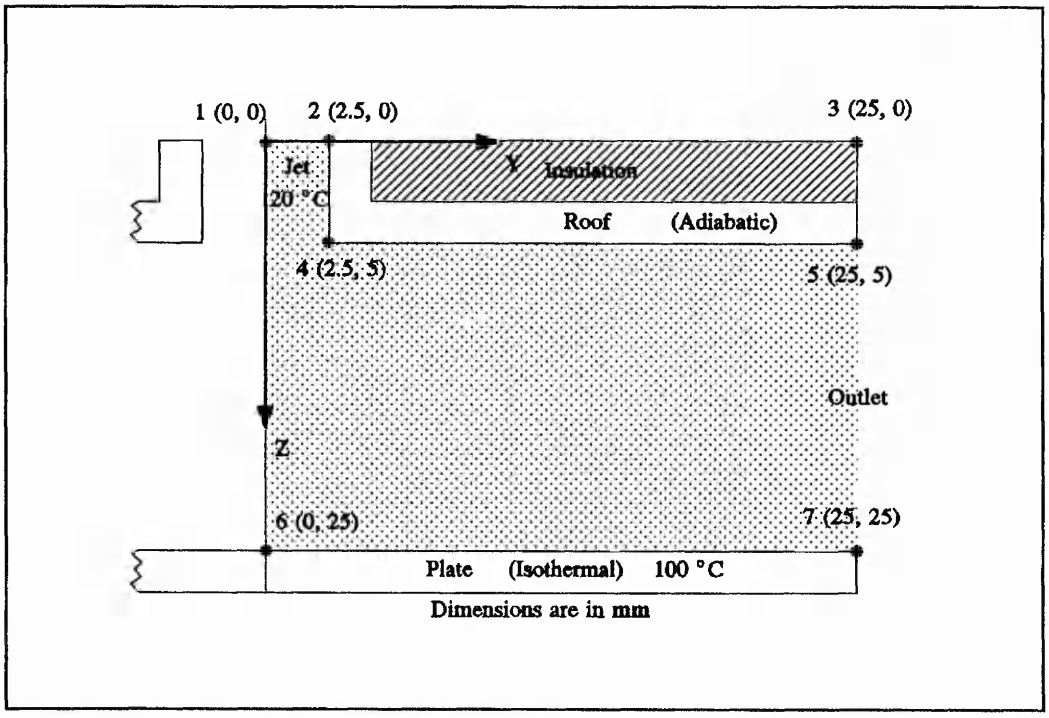


Figure 7.1: 2D confined jet impingement

Figure 7.1 shows the geometry of a confined impinging jet. The jet consists of compressed air whose fluid properties can be taken as ...

Density, ρ = 1.177 kg m^{-3}

Kinematic viscosity, ν = 1.568e-5 m^2 s^{-1}

Prandtl Number, Pr = 0.707

Also ...

Maximum Aspect Ratio = 10:1

To start the interactive session with the system, the command **(PHOENICS)** is entered at the LISP prompt, ==.

== (phoenics)
;;; LOADING object.lsp

1: NOVICE
2: EXPERIENCED
3: ADVANCED

Default value : NOVICE

What type of PHOENICS user to you consider yourself to be ?
(Enter 1 - 3) : == 2

1: NOVICE
2: EXPERIENCED
3: ADVANCED

Default value : NOVICE

What kind of KBFE user do you consider yourself to be ?
(Enter 1 - 3) : == 2

The analysis title cannot be more than 40 characters long. The main purpose of this is to be able to identify the analysis.

What is the analysis title ? == 2d Thermal Jet Impingement - Turbulent

PHOENICS essentially uses two types of coordinate systems - cartesian and cylindrical. For two dimensional configurations which this system can initially develop the XY plane will be utilised. This will be automatically translated into the respective XY or YZ planes which phoenics requires depending upon your choice of either cartesian of cylindrical coordinates.

1: CYLINDRICAL
2: CARTESIAN

Default value : CARTESIAN

Are the coordinates cartesian or cylindrical ?
(Enter 1 - 2) : == 1

What are the coordinate units

1: M
2: MM

Default value : MM

What are the dimensional units ?
(Enter 1 - 2) : ==

Enter the radial ordinate for node 1 == 0
Enter the axial ordinate for node 1 == 0

Enter the radial ordinate for node 2 == 2.5
Enter the axial ordinate for node 2 == 0

Enter the radial ordinate for node 3 == 25
Enter the axial ordinate for node 3 == 0

Enter the radial ordinate for node 4 == 2.5
Enter the axial ordinate for node 4 == 5

Enter the radial ordinate for node 5 == 25
Enter the axial ordinate for node 5 == 5

Enter the radial ordinate for node 6 == 0
Enter the axial ordinate for node 6 == 25

Enter the radial ordinate for node 7 == 25
Enter the axial ordinate for node 7 == 25

Enter the radial ordinate for node 8 == end

Enter connectivity command, ? for help == c 1 2 6
Enter connectivity command, ? for help == c 2 3 4
Enter connectivity command, ? for help == c 3 5
Enter connectivity command, ? for help == c 4 5
Enter connectivity command, ? for help == c 5 7
Enter connectivity command, ? for help == c 6 7
Enter connectivity command, ? for help == list

((1 (0.0 0.0 0.0) (2 6))
(2 (0.0 0.0025 0.0) (3 4 1))
(3 (0.0 0.025 0.0) (5 2))
(4 (0.0 0.0025 0.005) (5 2))
(5 (0.0 0.025 0.005) (7 4 3))
(6 (0.0 0.0 0.025) (7 1))
(7 (0.0 0.025 0.025) (6 5)))

Enter connectivity command, ? for help == end

Default value : 1
How many inlets are within the domain ? ==

Enter the boundary name for inlet 1 == jet

Current value of Surface nodes for jet : NIL
Surface nodes for jet == 1 2 end

Default value : 1
How many outlets are within the domain ? ==

Enter the boundary name for outlet 1 == outlet

Current value of Surface nodes for outlet : NIL
Surface nodes for outlet == 5 7 end

Default value : 0
How many obstructions are within the domain ? == 1

For the blockages you can define a default porosity - 0 for a solid - 1 for no obstruction or greater than 1 for simulating expanded cells. The default which you can predefine will be applied to all obstructions. Alternatively you can individually specify obstruction porosities.

1: CONSTANT-0.0
2: CONSTANT-PREDEFINED
3: INDIVIDUALLY-DEFINED

Default value : CONSTANT-0.0

Porosity definition :
(Enter 1 - 3) : ==

Enter the boundary name for obstruction 1 == roof

Current value of Surface nodes for roof : NIL
Surface nodes for roof == 2 3 4 5 end

Enter the boundary name for wall surface (6 7) == plate

1: THERMAL
2: ISOTHERMAL

Default value : ISOTHERMAL

Is the analysis thermal or isothermal ?
(Enter 1 - 2) : == 1

Enter the Prandtl number for the fluid [Dimensionless] == 0.707

If you wish to simulate the change of viscosity within the domain depending upon the calculated local temperatures then enter REQUIRED at the prompt.

1: REQUIRED
2: NOT-REQUIRED

Default value : NOT-REQUIRED

Viscosity thermal dependence required or not-required ?
(Enter 1 - 2) : ==

Enter the kinematic viscosity [m^2 / s] == 1.568e-5

1: LAMINAR
2: TURBULENT

Default value : LAMINAR

Is the flow to be laminar or turbulent ?
(Enter 1 - 2) : == 2

If you wish to simulate the change of density within the domain depending upon the localised thermal conditions then enter the appropriate value.

1: NOT-REQUIRED
2: ENTHALPY
3: TEMPERATURE

Default value : NOT-REQUIRED

Density thermal dependence
(Enter 1 - 3) : ==

Default value : 1.0
Enter the density [kg / m^3] == 1.177

Enter the v1 velocity at jet [m / s] == 0

Enter the w1 velocity at jet [m / s] == 31.4

Default value : 0.01
Enter the turbulence intensity at jet
(inlet 1 nodes (1 2)) ==

1: ISOTHERMAL
2: CONSTANT-HEAT-FLUX

Default value : ISOTHERMAL

Enter the thermal condition at jet
(inlet 1 nodes (1 2))
(Enter 1 - 2) : ==

Enter the temperature at jet
(inlet 1 nodes (1 2)) [Degrees Celsius] == 20

Default value : 0.0
Enter the outlet pressure at outlet [N / m^2] ==

1: ISOTHERMAL
2: ADIABATIC
3: CONSTANT-HEAT-FLUX

Enter the thermal condition at plate
(wall surface nodes (6 7))
(Enter 1 - 3) : == 1

Enter the temperature at plate
(wall surface nodes (6 7)) [Degrees Celsius] == 100

The recommended minimum cell size has been calculated as 0.08328 mm according to the geometry and existing boundary conditions. You can accept this default value by pressing return - or you can enter a new value.

Default value : 0.832803e-4
Enter the minimum cell size ==

Default value : 5
Enter the maximum allowed aspect ratio [Dimensionless] == 10

Group 1 complete
Group 2 complete
Group 3 complete
Group 4 complete
Group 5 complete
Group 6 complete
Group 7 complete
Group 8 complete
Group 9 complete
Group 10 complete
Group 11 complete
Group 12 complete
Group 13 complete
Group 14 complete
Group 15 complete
Group 16 complete

```
Group 17 complete
Group 18 complete
Group 19 complete
Group 20 complete
Group 21 complete
Group 22 complete
Group 23 complete
Group 24 complete

Enter the target PHOENICS data file.  Please include the file extension
- ie |target.file|

Default value : Q1.DAT
PHOENICS target data file ==

File has been written
NIL

The *ASSERTIONS* made during the session are ...

== *assertions*
(((boundary name for |$type| |$identity| |$nodes| is |$name|)
  (boundary name for inlet 1 (1 2) is jet)
  (boundary name for outlet 1 (5 7) is outlet)
  (boundary name for obstruction 1 (2 3 4 5) is roof)
  (boundary name for wall surface (6 7) is plate))
 ((cardinal for surface |$nodes| is |$cardinal|)
  (cardinal for surface (4 5) is high)
  (cardinal for surface (2 4) is south)
  (cardinal for surface (5 7) is north)
  (cardinal for surface (1 2) is low)
  (cardinal for surface (6 7) is high)
  (cardinal for surface (1 6) is south))
 ((surface |$surface| is part of |$obstruction|)
  (surface (4 5) is part of roof)
  (surface (2 4) is part of roof))
 ((|$dependent-variable| at |$type| boundary |$name| is |$condition| at
   |$quantity|)
  (w1 at inlet boundary jet is constant at 31.4)
  (V1 at inlet boundary jet is constant at 0.0)
  (Ke at inlet boundary jet is constant at 1.64327)
  (Ep at inlet boundary jet is constant at 1538.22)
  (H1 at inlet boundary jet is isothermal at 20.0)
  (P1 at outlet boundary outlet is constant at 0.0)
  (H1 at wall boundary plate is isothermal at 100.0))
 ((|$Axis| has |$n| regions)
  (x has 1 regions)
  (y has 2 regions)
  (z has 2 regions))
 ((|$axis| region |$no| cells |$first| to |$last|)
  (x region 1 cells 1 to 1)
  (y region 1 cells 1 to 13)
```

```
 (y region 2 cells 14 to 120)
 (z region 1 cells 1 to 25)
 (z region 2 cells 26 to 123))
((|$axis| region |$no| co-ordinates |$first| to |$last|)
 (y region 1 co-ordinates 0.0 To 0.0025)
 (Z region 1 co-ordinates 0.0 To 0.005)
 (Y region 2 co-ordinates 0.0025 To 0.025)
 (Z region 2 co-ordinates 0.005 To 0.025))
((Surface |$nodes| is in |$axis| regions |$start| to |$finish|)
 (surface (4 5) is in x regions 1 to 1)
 (surface (4 5) is in y regions 2 to 2)
 (surface (2 4) is in x regions 1 to 1)
 (surface (2 4) is in z regions 1 to 1)
 (surface (5 7) is in x regions 1 to 1)
 (surface (5 7) is in y regions 2 to 2)
 (surface (5 7) is in z regions 2 to 2)
 (surface (1 2) is in x regions 1 to 1)
 (surface (1 2) is in y regions 1 to 1)
 (surface (1 2) is in z regions 1 to 1)
 (surface (6 7) is in x regions 1 to 1)
 (surface (6 7) is in y regions 1 to 2)
 (surface (6 7) is in z regions 2 to 2)
 (surface (1 6) is in x regions 1 to 1)
 (surface (1 6) is in y regions 1 to 1)
 (surface (1 6) is in z regions 1 to 2))
((surface |$nodes| interfaces |$axis| regions |$start| and |$last|)
 (surface (4 5) interfaces z regions 1 and 2)
 (surface (2 4) interfaces y regions 1 and 2)))
```

... and the resulting data file is ...

```
Talk=F; Run(1, 1); VDU=TTY
   GROUP 1 run identifiers and other preliminaries
Text(2d thermal jet impingement - turbulent)
   GROUP 2 transience - time step specification
   GROUP 3 x-direction grid specification
Cartes=F
   Make sure that the array MAXFRC in SATLIT is at least 14760
   GROUP 4 y-direction grid specification
Ny=120
Yvlast=0.025
Yfrac(1)=0.836697e-4
Yfrac(2)=0.180809e-3
Yfrac(3)=0.296992e-3
Yfrac(4)=0.440924e-3
.
.
.
Yfrac(117)=0.023018
Yfrac(118)=0.023698
Yfrac(119)=0.024351
Yfrac(120)=0.025
```

```
   GROUP 5 z-direction grid specification
Nz=123
Zwlast=0.025
Zfrac(1)=0.458968e-3
Zfrac(2)=0.916301e-3
Zfrac(3)=0.001361
.
.
.
Zfrac(120)=0.024767
Zfrac(121)=0.024853
Zfrac(122)=0.024937
Zfrac(123)=0.025
   GROUP 6 body fitted cylindrical
   GROUP 7 variables - including porosities - named stored and
  solved
Store(ENUT)
Vist=50
Name(50)=ENUT
Solutn(P1,Y,Y,Y,N,N,N)
Solutn(H1,Y,Y,Y,N,N,N)
Solutn(V1,Y,Y,N,N,N,N)
Solutn(W1,Y,Y,N,N,N,N)
   GROUP 8 terms - in differential equations - and devices
Terms(H1,N,Y,Y,N,Y,N)
   GROUP 9 properties of the medium
Enul=0.1568e-4
Rho1=1.177
Prndtl(H1)=0.707
Turmod(KEMODL)
   GROUP 10 interphase transport processes and properties
   GROUP 11 initialisation of fields of variables porosities etc
Conpor(ROOF,0.0,CELL,1,1,-14,120,1,-25)
Fiinit(EP)=1538.22
Fiinit(KE)=1.64327
Fiinit(H1)=60.0
Fiinit(W1)=31.4
Fiinit(V1)=0.1
   *** when restarting deactivate previous FIINIT commands and
  activate the following RESTRT and FIINIT commands
   Restrt(All)
   Fiinit( p1 )= readfi
   Fiinit( v1 )= readfi
   Fiinit( w1 )= readfi
   Fiinit( h1 )= readfi
   Fiinit( ke )= readfi
   Fiinit( ep )= readfi
   Fiinit( enut )= readfi
   GROUP 12 unused
   GROUP 13 boundary and internal conditions and special sources
Patch(JET,LOW,1,1,1,13,1,1,1,1)
Coval(JET,P1,FIXFLU,36.9578)
```

```
Coval(JET,W1,ONLYMS,31.4)
Coval(JET,V1,ONLYMS,0.0)
Coval(JET,EP,ONLYMS,1538.22)
Coval(JET,KE,ONLYMS,1.64327)
Coval(JET,H1,ONLYMS,20.0)
Patch(OUTLET,NORTH,1,1,120,120,26,123,1,1)
Coval(OUTLET,P1,FIXP,0.0)
Patch(PLATE,HWALL,1,1,1,120,123,123,1,1)
Coval(PLATE,V1,GRND2,0.0)
Coval(PLATE,W1,FIXVAL,0.0)
Coval(PLATE,H1,FIXVAL,100.0)
Coval(PLATE,KE,GRND2,GRND2)
Coval(PLATE,EP,GRND2,GRND2)
   GROUP 14 down stream pressure - for free parabolic flow.
   GROUP 15 termination criteria for sweeps and outer iterations
Lsweep=100
Resref(EP)=1.0e-7
Resref(KE)=1.0e-7
Resref(H1)=1.0e-7
Resref(W1)=1.0e-7
Resref(V1)=1.0e-7
Resref(P1)=1.0e-7
   GROUP 16 termination criteria for inner iterations
   GROUP 17 under-relaxation and related sources
Relax(P1,LINRLX,0.8)
Relax(V1,FALSDT,0.5)
Relax(W1,FALSDT,0.5)
Relax(H1,LINRLX,1.0)
Relax(KE,FALSDT,0.01)
Relax(EP,FALSDT,0.01)
   GROUP 18 limits on variable values or increments to them
   GROUP 19 data communicated by SATELLITE to GROUND
   GROUP 20 control of preliminary printout
   GROUP 21 frequency and extent of field printout
Output(P1,Y,Y,Y,Y,Y,Y)
Output(V1,Y,Y,Y,Y,Y,Y)
Output(W1,Y,Y,Y,Y,Y,Y)
Output(H1,Y,Y,Y,Y,Y,Y)
Output(KE,Y,Y,Y,Y,Y,Y)
Output(EP,Y,Y,Y,Y,Y,Y)
   Group 22 location of spot-value and frequency of residual
  printout
Ixmon=1
Iymon=67
Izmon=74
   GROUP 23 variable-by-variable field printout
   GROUP 24 preparations for continuation runs.
Stop
```

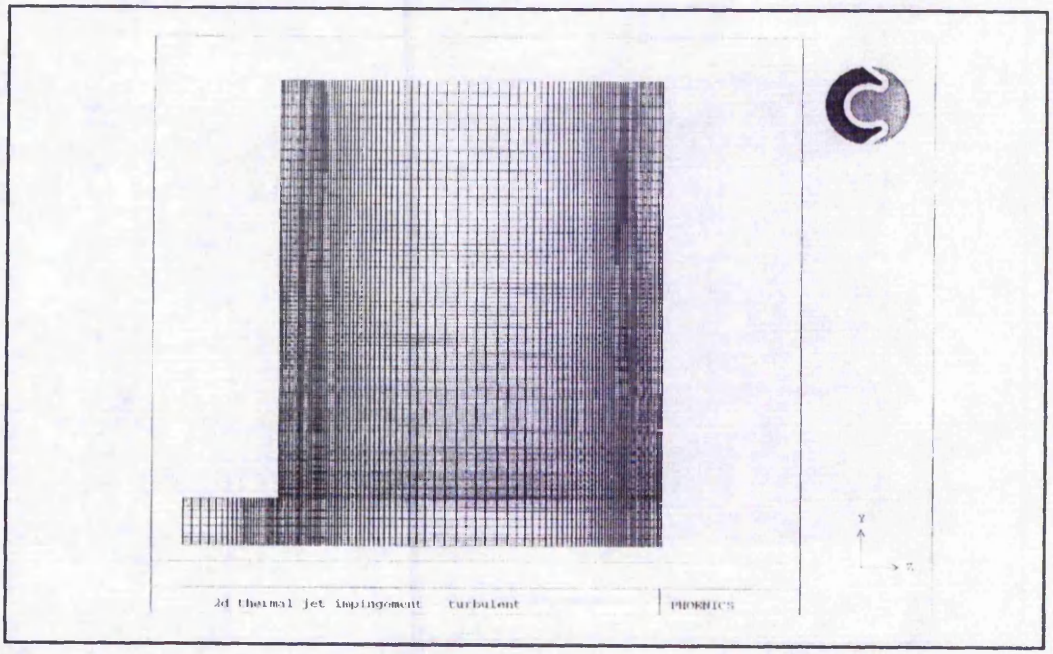


Figure 7.2: 2D meshed region of turbulent, confined, thermal jet impingement

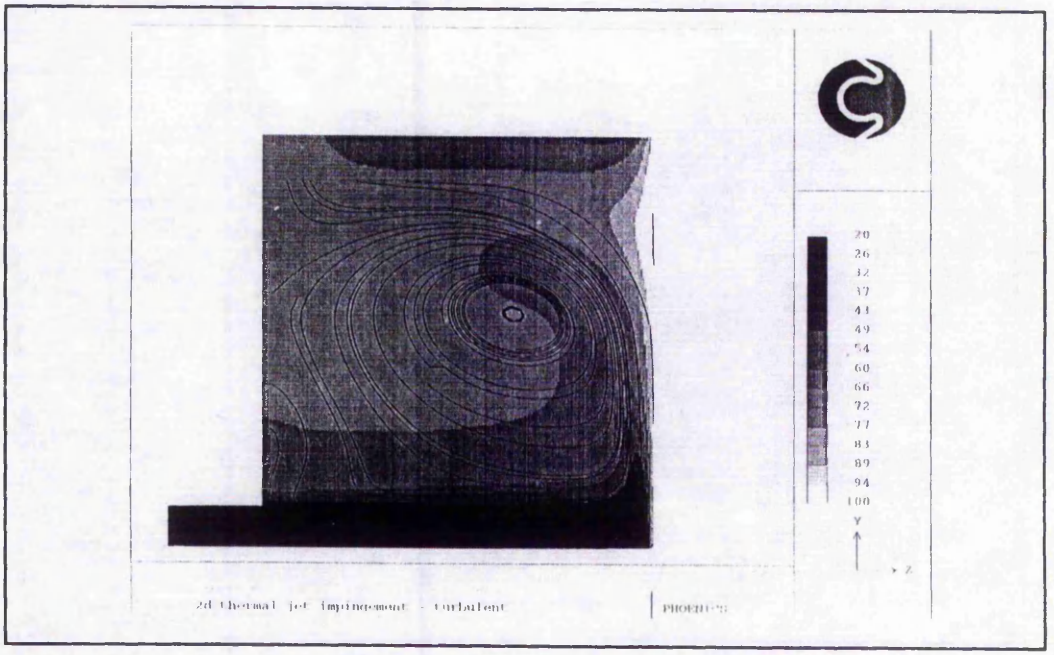


Figure 7.3: Filled temperature contours and stream lines

7.3 Two dimensional axisymmetric flow meter

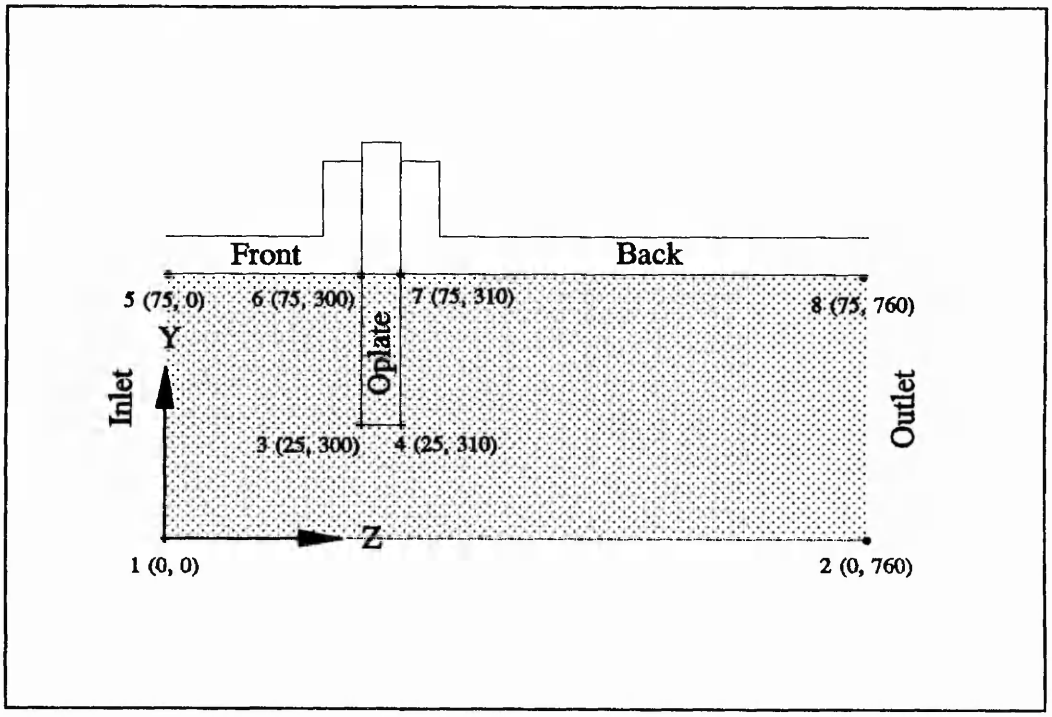


Figure 7.4 Two dimensional axisymmetric flow meter - Orifice plate

Figure 7.4 shows the geometry of an orifice meter. The fluid is air whose properties can be taken as ...

$$\text{Density, } \rho = 1.177 \text{ kg m}^{-3}$$

$$\text{Kinematic viscosity, } \nu = 1.568\text{e-}5 \text{ m}^2 \text{ s}^{-1}$$

Also ...

$$\text{Maximum Aspect Ratio} = 10{:}1$$

To start the interactive session with the system, the command **(PHOENICS)** is entered at the LISP prompt, ==.

```
== (phoenics)
;;; LOADING object.lsp
```

1: NOVICE
2: EXPERIENCED
3: ADVANCED

Default value : NOVICE

What type of PHOENICS user to you consider yourself to be ?
(Enter 1 - 3) : == 2

1: NOVICE
2: EXPERIENCED
3: ADVANCED

Default value : NOVICE

What kind of KBFE user do you consider yourself to be ?
(Enter 1 - 3) : == 2

The analysis title cannot be more than 40 characters long. The main purpose of this is to be able to identify the analysis.

What is the analysis title ? == 2D Axisymmetric Flow Meter

PHOENICS essentially uses two types of coordinate systems - cartesian and cylindrical. For two dimensional configurations which this system can initially develop the XY plane will be utilised. This will be automatically translated into the respective XY or YZ planes which phoenics requires depending upon your choice of either cartesian of cylindrical coordinates.

1: CYLINDRICAL
2: CARTESIAN

Default value : CARTESIAN

Are the coordinates cartesian or cylindrical ?
(Enter 1 - 2) : == 1

What are the coordinate units

1: M
2: MM

Default value : MM

What are the dimensional units ?
(Enter 1 - 2) : ==

Enter the radial ordinate for node 1 == 0
Enter the axial ordinate for node 1 == 0

Enter the radial ordinate for node 2 == 0

Enter the axial ordinate for node 2 == 760

Enter the radial ordinate for node 3 == 20
Enter the axial ordinate for node 3 == 300

Enter the radial ordinate for node 4 == 20
Enter the axial ordinate for node 4 == 310

Enter the radial ordinate for node 5 == 75
Enter the axial ordinate for node 5 == 0

Enter the radial ordinate for node 6 == h

The nodal-coordinates should be entered in < mm > depending on the prompt.
list - lists the nodes
m axis node - modify the coordinate of node on axis axis
m axis - modify the current nodal coordinate on axis

Enter the radial ordinate for node 6 == m y 3

Original coordinates (3 (0.0 0.02 0.3))
modify y ordinate for node 3 == 25

Enter the radial ordinate for node 6 == list

((1 (0.0 0.0 0.0) ())
(2 (0.0 0.0 0.76) ())
(3 (0.0 0.025 0.3) ())
(4 (0.0 0.02 0.31) ())
(5 (0.0 0.075 0.0) ()))

Enter the radial ordinate for node 6 == m y 4

Original coordinates (4 (0.0 0.02 0.31))
modify y ordinate for node 4 == 25

Enter the radial ordinate for node 6 == list

((1 (0.0 0.0 0.0) ())
(2 (0.0 0.0 0.76) ())
(3 (0.0 0.025 0.3) ())
(4 (0.0 0.025 0.31) ())
(5 (0.0 0.075 0.0) ()))

Enter the radial ordinate for node 6 == 75
Enter the axial ordinate for node 6 == 300

Enter the radial ordinate for node 7 == 75
Enter the axial ordinate for node 7 == 310

Enter the radial ordinate for node 8 == 75
Enter the axial ordinate for node 8 == 760

Enter the radial ordinate for node 9 == list

((1 (0.0 0.0 0.0) ())
 (2 (0.0 0.0 0.76) ())
 (3 (0.0 0.025 0.3) ())
 (4 (0.0 0.025 0.31) ())
 (5 (0.0 0.075 0.0) ())
 (6 (0.0 0.075 0.3) ())
 (7 (0.0 0.075 0.31) ())
 (8 (0.0 0.075 0.76) ()))

Enter the radial ordinate for node 9 == end

Enter connectivity command, ? for help == c 1 2 5
Enter connectivity command, ? for help == c 2 8
Enter connectivity command, ? for help == c 3 4
Enter connectivity command, ? for help == c 3 6
Enter connectivity command, ? for help == c 4 7
Enter connectivity command, ? for help == c 5 6
Enter connectivity command, ? for help == c 6 7
Enter connectivity command, ? for help == c 7 8
Enter connectivity command, ? for help == list

((1 (0.0 0.0 0.0) (2 5))
 (2 (0.0 0.0 0.76) (8 1))
 (3 (0.0 0.025 0.3) (6 4))
 (4 (0.0 0.025 0.31) (7 3))
 (5 (0.0 0.075 0.0) (6 1))
 (6 (0.0 0.075 0.3) (7 5 3))
 (7 (0.0 0.075 0.31) (8 6 4))
 (8 (0.0 0.075 0.76) (7 2)))

Enter connectivity command, ? for help == end

Default value : 1
How many inlets are within the domain ? ==

Enter the boundary name for inlet 1 == inlet

Current value of Surface nodes for inlet : NIL
Surface nodes for inlet == 1 5 end

Default value : 1
How many outlets are within the domain ? ==

Enter the boundary name for outlet 1 == outlet

Current value of Surface nodes for outlet : NIL

Surface nodes for outlet == 2 8 end

Default value : 0
How many obstructions are within the domain ? == 1

For the blockages you can define a default porosity - 0 for a solid - 1 for no obstruction or greater than 1 for simulating expanded cells. The default which you can predefine will be applied to all obstructions. Alternatively you can individually specify obstruction porosities.

1: CONSTANT-0.0
2: CONSTANT-PREDEFINED
3: INDIVIDUALLY-DEFINED

Default value : CONSTANT-0.0

Porosity definition :
(Enter 1 - 3) : ==

Enter the boundary name for obstruction 1 == oplate

Current value of Surface nodes for oplate : NIL
Surface nodes for oplate == 3 4 6 7 end

Enter the boundary name for wall surface (7 8) == back

Enter the boundary name for wall surface (5 6) == front

1: THERMAL
2: ISOTHERMAL

Default value : ISOTHERMAL

Is the analysis thermal or isothermal ?
(Enter 1 - 2) : ==

Enter the kinematic viscosity [m^2 / s] == 1.568e-5

1: LAMINAR
2: TURBULENT

Default value : LAMINAR

Is the flow to be laminar or turbulent ?
(Enter 1 - 2) : == 2

Default value : 1.0
Enter the density [kg / m^3] == 1.177

Enter the v1 velocity at inlet [m / s] == 0

Enter the w1 velocity at inlet [m / s] == 1.05

Default value : 0.01
Enter the turbulence intensity at inlet
(inlet 1 nodes (1 5)) ==

Default value : 0.0
Enter the outlet pressure at outlet [N / m^2] ==

The recommended minimum cell size has been calculated as 0.91084 mm according to the geometry and existing boundary conditions. You can accept this default value by pressing return - or you can enter a new value.

Default value : 0.91084e-3
Enter the minimum cell size ==

Default value : 5
Enter the maximum allowed aspect ratio [Dimensionless] == 10

Group 1 complete
Group 2 complete
Group 3 complete
Group 4 complete
Group 5 complete
Group 6 complete
Group 7 complete
Group 8 complete
Group 9 complete
Group 10 complete
Group 11 complete
Group 12 complete
Group 13 complete
Group 14 complete
Group 15 complete
Group 16 complete
Group 17 complete
Group 18 complete
Group 19 complete
Group 20 complete
Group 21 complete
Group 22 complete
Group 23 complete
Group 24 complete

Enter the target PHOENICS data file. Please include the file extension
- ie |target.file|

Default value : Q1.DAT
PHOENICS target data file ==

File has been written
NIL

The ***ASSERTIONS*** made during the session are ...

```
== *assertions*
(((boundary name for |$type| |$identity| |$nodes| is |$name|)
  (boundary name for inlet 1 (1 5) is inlet)
  (boundary name for outlet 1 (2 8) is outlet)
  (boundary name for obstruction 1 (3 4 6 7) is oplate)
  (boundary name for wall surface (7 8) is back)
  (boundary name for wall surface (5 6) is front))
 ((cardinal for surface |$nodes| is |$cardinal|)
  (cardinal for surface (4 7) is high)
  (cardinal for surface (3 4) is south)
  (cardinal for surface (3 6) is low)
  (cardinal for surface (2 8) is high)
  (cardinal for surface (1 5) is low)
  (cardinal for surface (7 8) is north)
  (cardinal for surface (5 6) is north)
  (cardinal for surface (1 2) is south))
 ((surface |$surface| is part of |$obstruction|)
  (surface (4 7) is part of oplate)
  (surface (3 4) is part of oplate)
  (surface (3 6) is part of oplate))
 ((|$dependent-variable| at |$type| boundary |$name| is |$condition| at
   |$quantity|)
  (w1 at inlet boundary inlet is constant at 1.05)
  (V1 at inlet boundary inlet is constant at 0.0)
  (Ke at inlet boundary inlet is constant at 0.001838)
  (Ep at inlet boundary inlet is constant at 0.001917)
  (P1 at outlet boundary outlet is constant at 0.0))
 ((|$Axis| has |$n| regions)
  (x has 1 regions)
  (y has 2 regions)
  (z has 3 regions))
 ((|$axis| region |$no| cells |$first| to |$last|)
  (x region 1 cells 1 to 1)
  (y region 1 cells 1 to 13)
  (y region 2 cells 14 to 37)
  (z region 1 cells 1 to 130)
  (z region 2 cells 131 to 135)
  (z region 3 cells 136 to 330))
 ((|$axis| region |$no| co-ordinates |$first| to |$last|)
  (y region 1 co-ordinates 0.0 To 0.025)
  (Z region 1 co-ordinates 0.0 To 0.3)
  (Y region 2 co-ordinates 0.025 To 0.075)
  (Z region 2 co-ordinates 0.3 To 0.31)
  (Z region 3 co-ordinates 0.31 To 0.76))
 ((Surface |$nodes| is in |$axis| regions |$start| to |$finish|)
  (surface (4 7) is in x regions 1 to 1)
  (surface (4 7) is in y regions 2 to 2)
  (surface (3 4) is in x regions 1 to 1)
  (surface (3 4) is in z regions 2 to 2)
  (surface (3 6) is in x regions 1 to 1)
```

```
  (surface (3 6) is in y regions 2 to 2)
  (surface (2 8) is in x regions 1 to 1)
  (surface (2 8) is in y regions 1 to 2)
  (surface (2 8) is in z regions 3 to 3)
  (surface (1 5) is in x regions 1 to 1)
  (surface (1 5) is in y regions 1 to 2)
  (surface (1 5) is in z regions 1 to 1)
  (surface (7 8) is in x regions 1 to 1)
  (surface (7 8) is in y regions 2 to 2)
  (surface (7 8) is in z regions 3 to 3)
  (surface (5 6) is in x regions 1 to 1)
  (surface (5 6) is in y regions 2 to 2)
  (surface (5 6) is in z regions 1 to 1)
  (surface (1 2) is in x regions 1 to 1)
  (surface (1 2) is in y regions 1 to 1)
  (surface (1 2) is in z regions 1 to 3))
 ((surface |$nodes| interfaces |$axis| regions |$start| and |$last|)
  (surface (4 7) interfaces z regions 2 and 3)
  (surface (3 4) interfaces y regions 1 and 2)
  (surface (3 6) interfaces z regions 1 and 2)))
==
```

... and the resulting data file is ...

```
Talk=F; Run(1, 1); VDU=TTY
   GROUP 1 run identifiers and other preliminaries
Text(2d axisymmetric flow meter)
   GROUP 2 transience - time step specification
   GROUP 3 x-direction grid specification
Cartes=F
   Make sure that the array MAXFRC in SATLIT is at least 12210
   GROUP 4 y-direction grid specification
Ny=37
Yvlast=0.075
Yfrac(1)=0.9151e-3
Yfrac(2)=0.001993
.
.
.
Yfrac(35)=0.073927
Yfrac(36)=0.074849
Yfrac(37)=0.075
   GROUP 5 z-direction grid specification
Nz=330
Zwlast=0.76
Zfrac(1)=0.007794
Zfrac(2)=0.015581
Zfrac(3)=0.023203
Zfrac(4)=0.030415
Zfrac(5)=0.037259
Zfrac(6)=0.043769
```

```
Zfrac(7)=0.049976
Zfrac(8)=0.055907
.
.
.
.
Zfrac(328)=0.749182
Zfrac(329)=0.754654
Zfrac(330)=0.76
   GROUP 6 body fitted cylindrical
   GROUP 7 variables - including porosities - named stored and
  solved
Store(ENUT)
Vist=50
Name(50)=ENUT
Solutn(P1,Y,Y,Y,N,N,N)
Solutn(V1,Y,Y,N,N,N,N)
Solutn(W1,Y,Y,N,N,N,N)
   GROUP 8 terms - in differential equations - and devices
   GROUP 9 properties of the medium
Enul=0.1568e-4
Rho1=1.177
Turmod(KEMODL)
   GROUP 10 interphase transport processes and properties
   GROUP 11 initialisation of fields of variables porosities etc
Conpor(OPLATE,0.0,CELL,1,1,-14,37,-131,-135)
Fiinit(EP)=0.001917
Fiinit(KE)=0.001838
Fiinit(W1)=1.05
Fiinit(V1)=0.1
   *** when restarting deactivate previous FIINIT commands and
  activate the following RESTRT and FIINIT commands
   Restrt(All)
   Fiinit( p1 )= readfi
   Fiinit( v1 )= readfi
   Fiinit( w1 )= readfi
   Fiinit( ke )= readfi
   Fiinit( ep )= readfi
   Fiinit( enut )= readfi
   GROUP 12 unused
   GROUP 13 boundary and internal conditions and special sources
Patch(INLET,LOW,1,1,1,37,1,1,1,1)
Coval(INLET,P1,FIXFLU,1.23585)
Coval(INLET,W1,ONLYMS,1.05)
Coval(INLET,V1,ONLYMS,0.0)
Coval(INLET,EP,ONLYMS,0.001917)
Coval(INLET,KE,ONLYMS,0.001838)
Patch(OUTLET,HIGH,1,1,1,37,330,330,1,1)
Coval(OUTLET,P1,FIXP,0.0)
Patch(BACK,NWALL,1,1,37,37,136,330,1,1)
Coval(BACK,V1,FIXVAL,0.0)
Coval(BACK,W1,GRND2,0.0)
```

```
Coval(BACK,KE,GRND2,GRND2)
Coval(BACK,EP,GRND2,GRND2)
Patch(FRONT,NWALL,1,1,37,37,1,130,1,1)
Coval(FRONT,V1,FIXVAL,0.0)
Coval(FRONT,W1,GRND2,0.0)
Coval(FRONT,KE,GRND2,GRND2)
Coval(FRONT,EP,GRND2,GRND2)
   GROUP 14 down stream pressure - for free parabolic flow.
   GROUP 15 termination criteria for sweeps and outer iterations
Lsweep=100
Resref(EP)=1.0e-5
Resref(KE)=1.0e-5
Resref(W1)=1.0e-5
Resref(V1)=1.0e-5
Resref(P1)=1.0e-5
   GROUP 16 termination criteria for inner iterations
   GROUP 17 under-relaxation and related sources
Relax(P1,LINRLX,0.8)
Relax(V1,FALSDT,0.5)
Relax(W1,FALSDT,0.5)
Relax(KE,FALSDT,0.01)
Relax(EP,FALSDT,0.01)
   GROUP 18 limits on variable values or increments to them
   GROUP 19 data communicated by SATELLITE to GROUND
   GROUP 20 control of preliminary printout
   GROUP 21 frequency and extent of field printout
Output(P1,Y,Y,Y,Y,Y,Y)
Output(V1,Y,Y,Y,Y,Y,Y)
Output(W1,Y,Y,Y,Y,Y,Y)
Output(KE,Y,Y,Y,Y,Y,Y)
Output(EP,Y,Y,Y,Y,Y,Y)
   Group 22 location of spot-value and frequency of residual
  printout
Ixmon=1
Iymon=19
Izmon=233
   GROUP 23 variable-by-variable field printout
   GROUP 24 preparations for continuation runs.
Stop
```

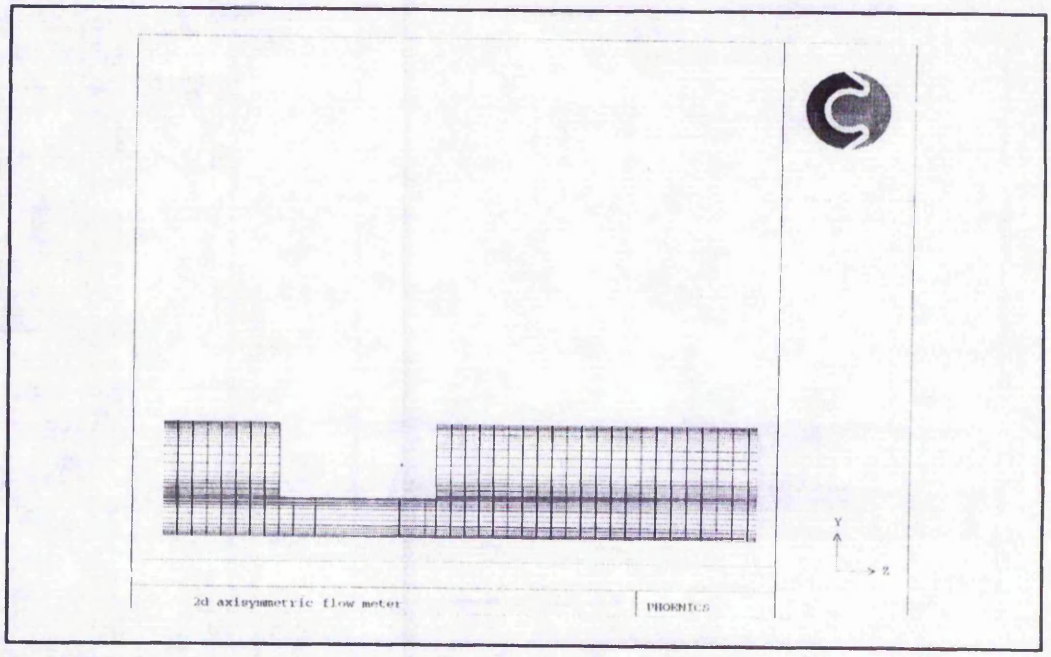


Figure 7.5: 2D meshed region of an axisymmetric flow meter

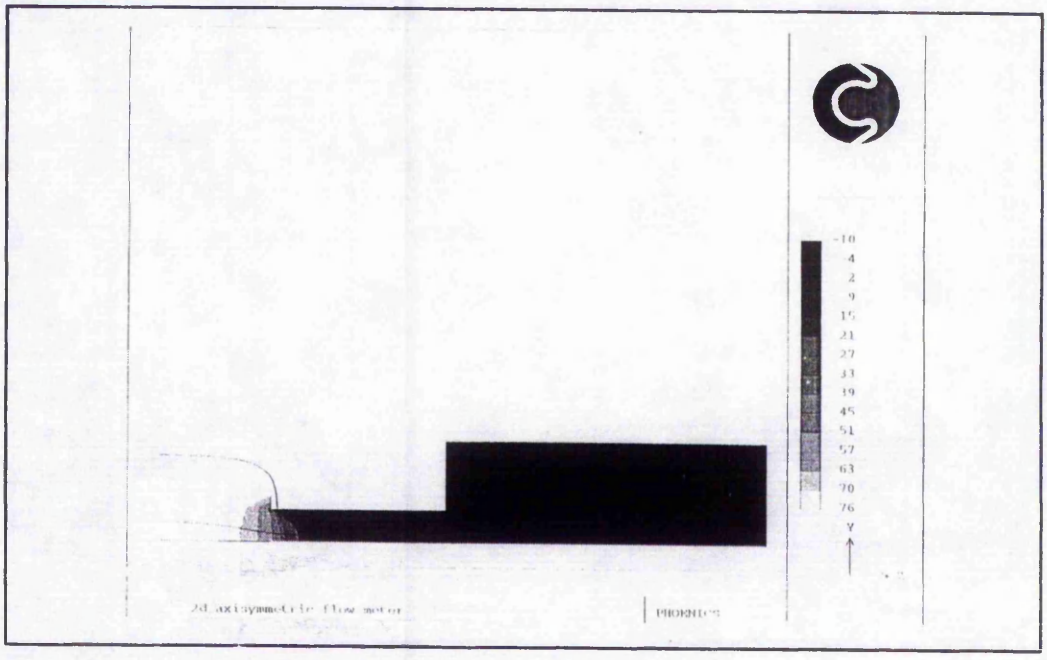


Figure 7.6: Filled pressure contours and stream lines

CHAPTER 8

FEASIBILITY STUDIES

8.1 Introduction

During the process of learning the concepts and fundamentals of Computational Fluid Dynamics, and becoming familiar with PHOENICS, various techniques were encountered that were considered both interesting and important for CFD users. These consisted of: grid generation, taking into account the importance of the cell aspect ratios, Abbott et al. (1988); monitoring and control of the solution algorithm; and the analysis of results through post processing and adaptive grid optimisation.

Grid generation is a pre-requisite to data file preparation, and as such needed to be fully addressed. Furthermore, the importance of cell aspect ratios cannot be over emphasised, but little, if any, work has been performed on aspect ratio dependent grid generation. The work covered in Section 3.8 details the development of a technique implementing a generalised Fourier Series for aspect ratio dependent grid generation.

The areas of solution monitoring and control, as well as the analysis of results through post processing and grid optimisation were investigated. Feasibility studies were performed which indicated the potential for further work. This chapter details the feasibility studies into (a) the monitoring and control of the solution algorithm, and (b) the analysis of results through post processing grid optimisation. The potential for each area will be discussed.

8.2 Monitoring and control of the solution algorithm

During a typical PHOENICS analysis it is necessary to predefine the solution parameters within the Q1.DAT data file. These include the number of iterations/sweeps, residual reference values, and relaxation factors. Monitor spot values are used to determine the condition to the solution at the end of the current number of sweeps.

The number of iterations/sweeps governs the maximum number of combined iterations that are performed in the set of linear equations. Due to the infrastructure of PHOENICS, there exists several levels of iteration, the main level being the sweeps. Sweeps govern the slab by slab iteration cycle, and enable elliptic solutions to be obtained.

A non sweeping iterative cycle results in parabolic flow being assumed, where no recirculation regions exist.

The residual reference values are used to increase the magnitude of the dependent variable absolute residuals between successive sweeps. The residual reference values are of the order of 1.0E-8, and are determined from predefined criteria. These are used by PHOENICS to establish when the solution of a dependent variable has converged.

Relaxation factors are used to dictate the magnitude of the solution variable that is to be carried forward into the subsequent iteration. The relaxation factor is a user defined value between zero and unity for under-relaxation, and greater than unity for over-relaxation. These can be seen to be the controlling factor governing the convergence of a solution.

8.2.1 Heuristic monitoring and control

The manual procedure used to submit, monitor and control the solution algorithm of PHOENICS can be divided into four areas. These are ...

(a) Create the data file, set the residual reference values, monitor spot positions, relaxation factors and number of sweeps. Submit to PHOENICS for analysis.

(b) Upon completion of the predefined number of sweeps, access the RESULT.DAT file and assess the trends of the residuals and the monitor spot values for each dependent variable. These are presented on a low resolution graph, sufficient for a general "feel" of convergence stability.

(c) Determine the mass continuity of the analysis. If mass continuity does not exist, the solution has not converged.

(d) Depending on the continuity and the trends shown on the graphs in the RESULT.DAT file, modify the relaxation values (if necessary), and restart the analysis from the end of the previous submission.

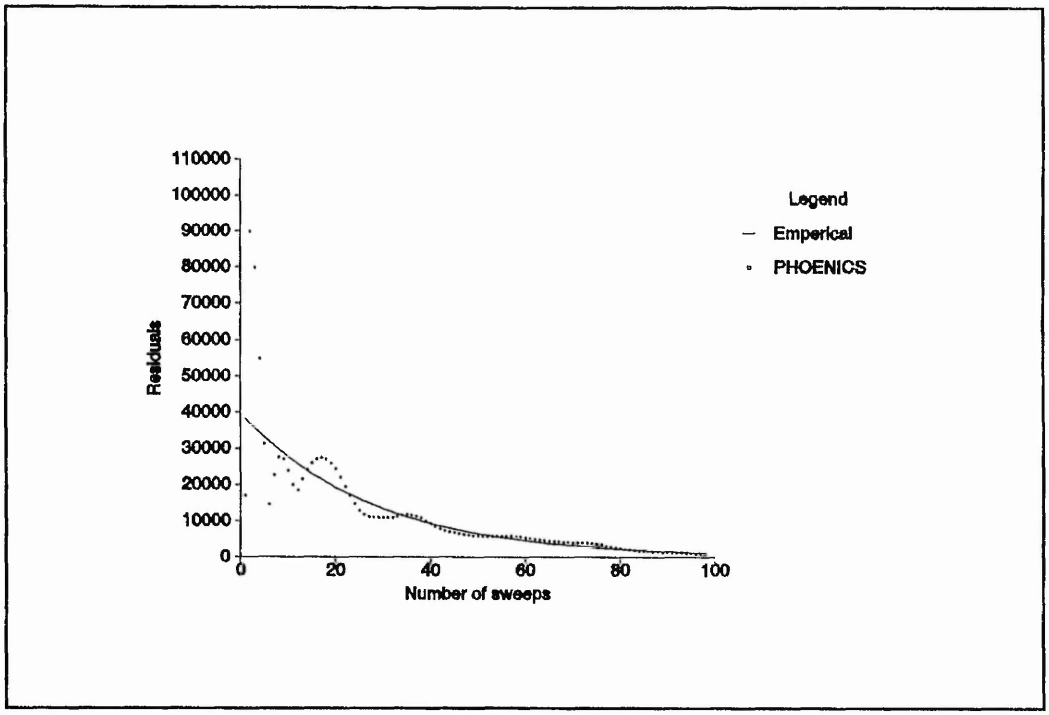


Figure 8.1: Initial instability followed by rapid convergence.
[Correlation coefficient = -0.9728]

There exists several residual profiles during the convergence of a particular dependent variable, which can be categorised into five stages. Initial convergence (Figure 8.1), where the dependent variable required preliminary adjustment to bring it into line for the subsequent rapid convergence; Removal of the outliers (Figure 8.2) using statistical techniques; Rapid convergence (Figure 8.3), where the solution domain is being continually refined according to the present relaxation factors; Figure 8.4 shows that the limit of the current relaxation has been reached whereby oscillations have started in the convergence trend; and Figure 8.5 shows the oscillations have become exacerbated with little or no convergence resulting.

Convergence can be restored from in the last two stages by reducing the relaxation factors for subsequent iterations. This would result in a situation similar to that shown in Figure 8.1. Repeating this procedure with restarted runs could be seen to be infinitely repetitive and thus requires a point at which the process should stop. Whenever a distribution of residuals against sweep has been obtained similar to that shown in

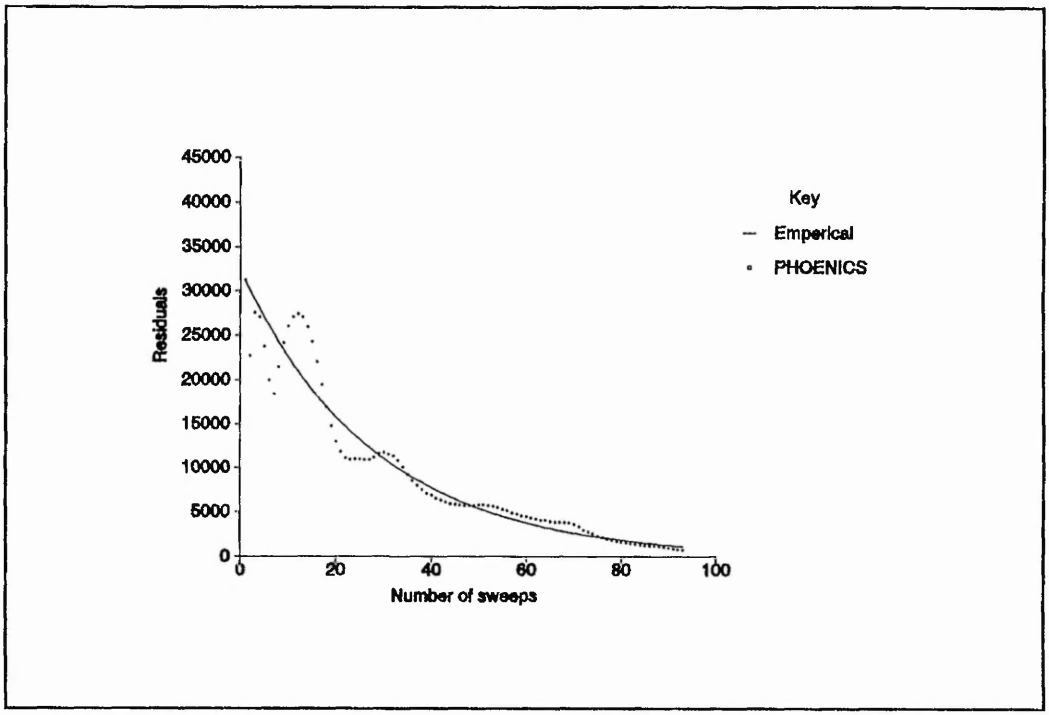


Figure 8.2: Modified version of Figure 8.1 with outliers removed.
[Correlation coefficient = -0.9836]

Figure 8.5, it is normal to consider the convergence performance of the monitor spot values. Monitor spot values are those dependent variable values associated with a predefined cell within a domain. The position of the cell is usually dependent on the geometry and the anticipated flow field, and should coincide with an area which is expected to converge slower compared with the main flow field. For example, an area containing a recirculation zone would converge more slowly than a region within the main stream of the flow. When the residual convergence profile is highly oscillatory, the monitor spot values indicate whether the solution has fully converged. If the maximum and minimum spot values are, say, within 10% of the average spot value and a similar condition for the residuals exists, as well as continuity being satisfied, then it can be assumed that the solution has fully converged. It must be noted that the residuals should still be relatively small. Once a fully converged solution has been established, it is possible to verify the solution by increasing the relaxation factors to unity and observing the effects for a subsequent, say, fifty sweeps. A minimal change in the residuals and spot values would give a degree of confidence in the convergence of the solution.

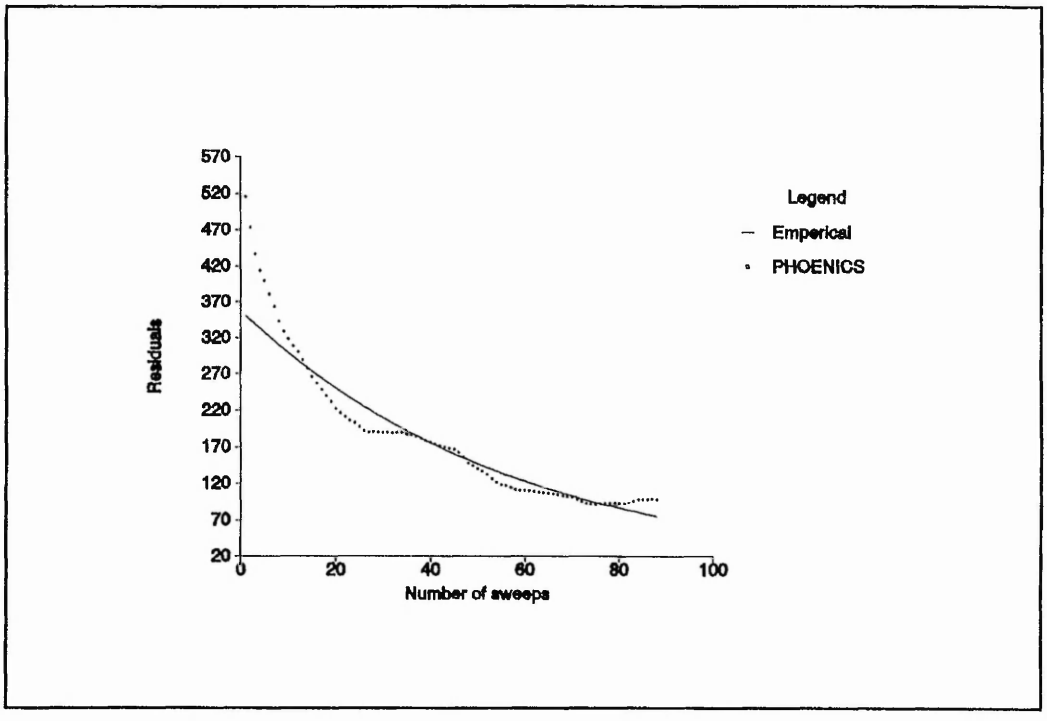


Figure 8.3: Continuous rapid convergence

The technique of physically observing the profiles of the residuals and monitor spot values is known as manual checking and control of the solution algorithm. There is an alternative technique that is considerably less labour intensive when monitoring the solution algorithm. Usually with the manual method, after several restarts of the analysis, each of which reduces the relaxation factors further, the magnitude of the relaxation factors can be quite small, of the order of 1.0E-2. The alternative method relies on the fact that the initial relaxation factors would be set to their anticipated final values. Due to these small relaxation factors, the number of iterations required would be correspondingly large. Small relaxation factors would prevent the observed oscillations in the convergence profile, thus indicating gradual convergence. An optimum situation would be to combine the two methods. That is, allow the computer to control the convergence by monitoring the residuals and appropriately modify the relaxation factors. This could be classified as pseudo real-time control (PRTC).

Pseudo real time control for numerical simulation packages would allow the code to automatically monitor the convergence of the solution. This would be performed using

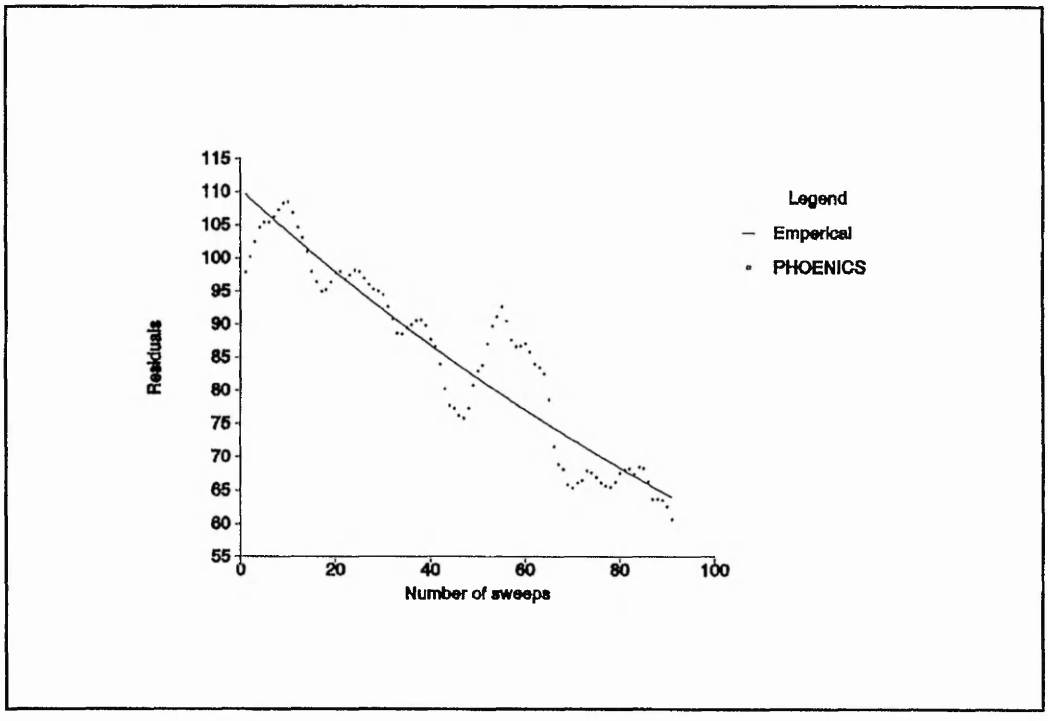


Figure 8.4: Onset of oscillations indicating the limit of current relaxation factors. [Correlation coefficient = -0.9]

standard statistical methods. The following describes the feasibility study, and the method by which it could be implemented.

The overall structure of PHOENICS allows the user to enter his own FORTRAN coding into a section of supplied source code, GROUND.FOR, figure 3.1. To run PHOENICS, a data file, Q1.DAT, is initially prepared and interpreted by SATELLITE, which in turn creates EARDAT.DAT. This is used as the primary input into EARTH. The data given to EARTH via EARDAT.DAT details all the information required to perform the analysis. It is also possible to pass from the Q1.DAT file integer, real and/or logical variables for use in GROUND. During the execution, continuous visits are made by EARTH to GROUND to execute any code that may reside there. The default GROUND source file is empty of user code. Therefore, to implement user routines it is necessary to compile the new ground code and to link the EARTH and GROUND object codes together to create a single executable file, EAREXE.EXE.

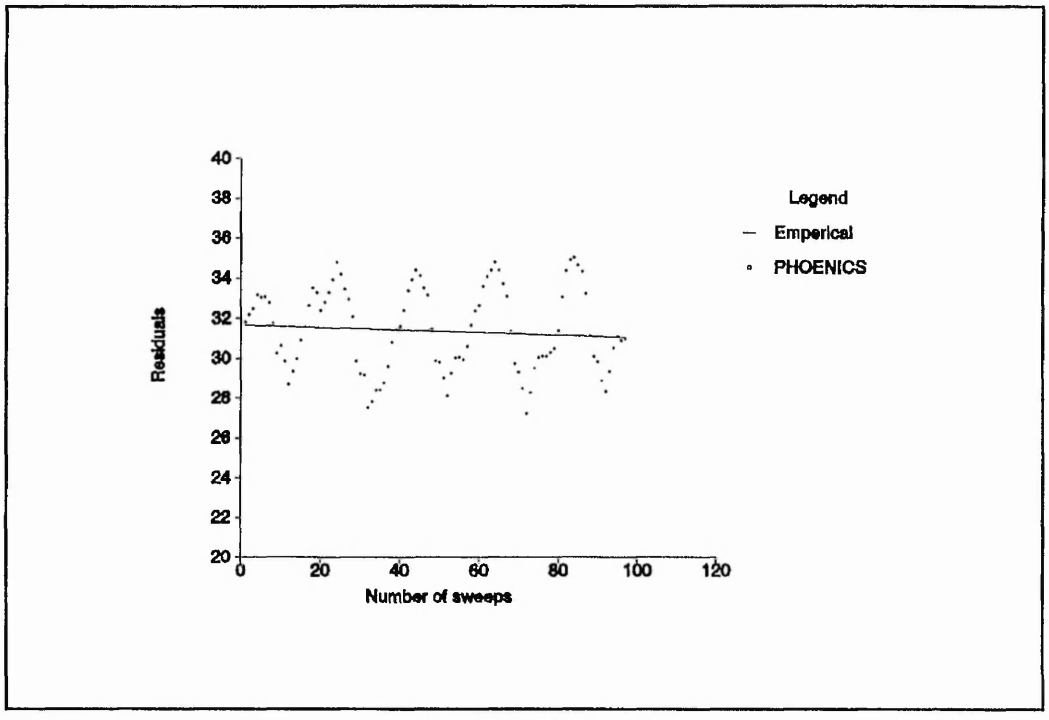


Figure 8.5: Exacerbated oscillations with little or no convergence.
[Correlation coefficient = 0.0857]

8.2.2 Directly requesting user declared code from Q1.DAT

To simplify the extremity of the GROUND code required, a logical variable was used to flag whether the user defined routines needed to be executed. This logical variable, when set to T, signals that the user defined routines needed to be executed. The logical variable is defined in the Q1.DAT file, **LG(20)**. A single line of code is all that is required in GROUND to execute the user defined subroutines. The code is ...

if (LG(20)) call *user subroutines*

The user subroutines are separately coded FORTRAN sequences that have been compiled and linked together with GROUND and EARTH to create a single executable file. Different logical variables could be used to call different routines. PHOENICS has reserved a one dimensional logical array of twenty elements to be used from within Q1.DAT for passing values directly to GROUND, hence LG(20).

Within the *user subroutines* file there could exist code to perform a multitude of tasks, each being called by using a different logical variable. Using different files, separate from GROUND, allows the easy location and modification of the user's own generated code, as opposed to having to become familiar with the layout and structure of GROUND.FOR. This is generally a more preferred method of interfacing with EARTH because new versions of code that have subtle changes only required a single line of code inserting into GROUND as opposed to large quantities of code.

8.2.3 Location of residuals and monitor spot values within GROUND

Having established a method of directly requesting the use of particular code from within the Q1.DAT file it was then necessary to determine how to obtain and manipulate the residuals and monitor spot values with respect to the sweeps. The access of the spot values is relatively straight forward, and can be achieved by means of one of two methods, TR200 (1990), (i) by using PHOENICS functions within GROUND, namely GETYX, or (ii) by directly accessing the F-array from within the user defined code. Both techniques allow access to the solution dependent variables for each cell within the domain. However, access to the residuals for each sweep proved to be somewhat more awkward. Private communications with CHAM established the method for accessing the residuals during run time. This facility was not made available to the general user, for reasons only known to CHAM. However, the only requirement, for the residuals to be accessed, was that the following COMMON statement needed to be included in each subroutine that accessed the residuals ...

```
COMMON /GR1/STOR(50)/GR2/SLBRES(50)/GR3/TOTRES(50)
```

The array TOTRES contains the TOTal RESiduals for the entire sweep. The array is one dimensional with fifty elements which coincide with the fifty dependent variables associated with PHOENICS. The single dimension of the array implies that the values are modified for each sweep. Due to this it was necessary to locally store all of the residuals for manipulation at a later stage.

Experience has shown that a typical number of sweeps required to produce a reasonable assessment is approximately one hundred. The feasibility source code used for the location of residuals, spot values, and the statistical analysis is shown in Appendix I.

8.2.4 Location of relaxation values

As with the dependent variable residuals, the relaxation values are also stored within memory. There are two methods of relaxing a variable: linear relaxation and false time step relaxation. Linear relaxation is used for scalar variables, and false time step relaxation for velocities. The two different techniques of relaxing a variable have the associated value stored in the array DTFALS, which is one dimensional with fifty elements associated with the fifty allowable dependent variables. Within the array, DTFALS, linear relaxation for a particular variable is stored as negative for linear relaxation, and positive for false time step relaxation. The magnitude being the most important as opposed to the sign which indicates the type of relaxation. A pure declarative statement allows the relaxation value to be modified. For example, assume that the original relaxations for pressure, P1, and the z velocity, W1, are 0.8 and 0.5 respectively. To modify the values within the user defined FORTRAN routines, ensuring that the appropriate COMMON statements are present, would be ...

DTFALS(P1) = -0.6
DTFALS(W1) = 0.4

The two declarative statements would modify the linear relaxation of P1 to 0.6, and the false time step relaxation of W1 to 0.4.

8.2.5 Statistical analysis

There exists three stages within the statistical analysis ...

(a) Curve fitting to obtain an approximate correlation,
(b) Scatter analysis,
(c) Gradient analysis.

Essentially, when analysing a graph of any sort it is necessary to try and assess the distribution of points. Curve fitting techniques were used to perform this assessment and the degree of scatter was indicated through the correlation coefficient.

Convergence of iterative solutions is, by nature, a function of an exponential relationship, and as such logarithmic regression analysis can be used implementing the generic equation ...

$$y = A\, e^{Bx} \tag{8.1}$$

Simplifying this into a linear, natural logarithmic relationship, we have ...

$$\log_e y = \log_e A + Bx \tag{8.2}$$

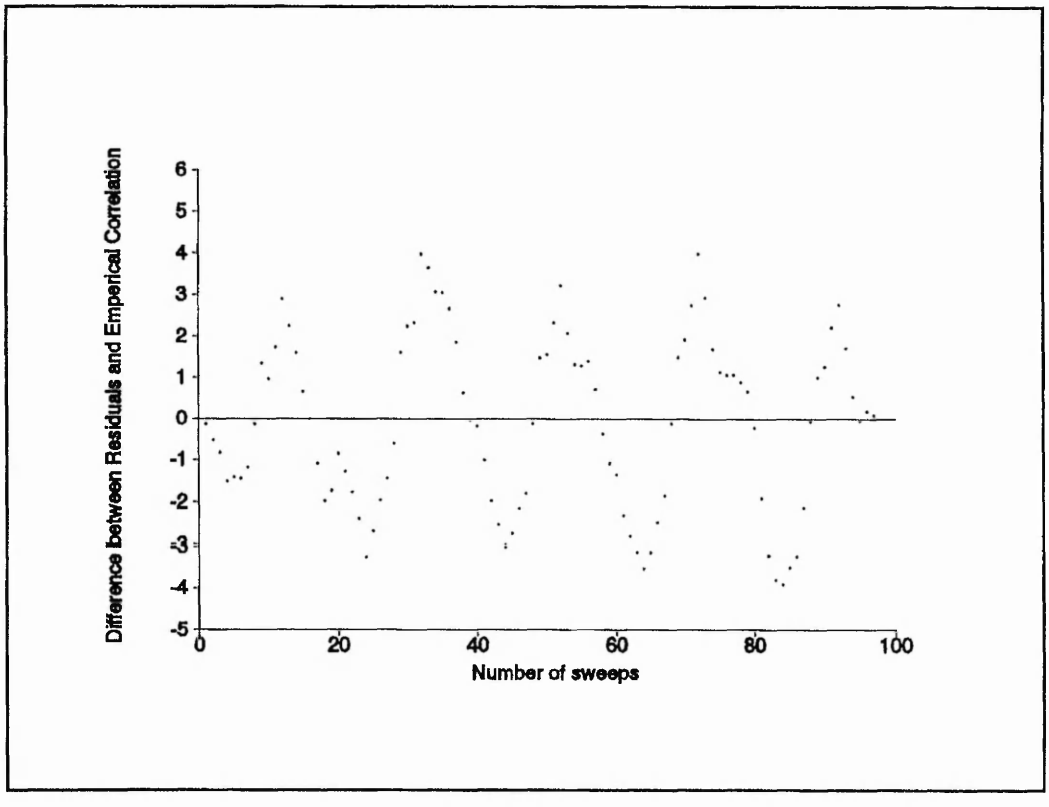


Figure 8.6: A typical residual scatter plot

Thus, standard linear regression techniques enables the constants C, **B** and the correlation coefficient **r** to be obtained, where $C = \log_e A$. Having obtained the correlation equation, the correlation residuals produce a typical scatter plot as shown in Figure 8.6. Calculating the standard deviation for the scatter plot enables the outliers to be determine, thus

removing them from the analysis. The standard deviation is the square root of the mean of the squared deviations from the mean of a set of observations; the square root of the variance. Therefore, the upper and lower limits are based on a factor multiplied by the standard deviation. This factor is obtained from the **t-distribution** tables by Murdoch and Barnes (1985), based on a 95% confidence interval requirement. The t-distribution assumes that a normal distribution of the points on the scatter plot exists. This statistical table was used because of the relatively small number of points used within the analysis, i.e. one hundred sweeps. For the required 95% confidence interval on one hundred sweeps, a factor of 2.0 was obtained. Thus the upper and lower limits for the correlation residuals, and hence the solution residuals, was given by ...

$$\textit{Limits} = \pm\ 2.0\ \textit{(Standard deviation)}$$

Using these limiting factors it is possible to remove spurious points from the data that would have an adverse effect upon the analysis. Having removed the necessary points, the regression analysis is performed again on the modified data which produces the necessary information to assess the convergence.

The gradient analysis consists of assessing the average gradient of the convergence profile of the entire one hundred sweeps, as seen in Figure 8.7. A more complex gradient analysis, consisting of differentiating the correlated equation, and calculating the change of gradient at the first and last sweep could be performed, or by monitoring the change of gradient for each sweep. This was considered excessive because a general trend was only required, hence the use of the average gradient.

8.3 Post processing grid optimisation

The feasibility of introducing adaptive mesh refinement, or grid optimisation, into PHOENICS has been considered. The approach to be taken would involve the submission of an initial data file to PHOENICS which contains a **best guess** grid. The converged solution would then be assessed, and a new mesh developed. The cycle, from initial submission would then be repeated until a grid independent solution results.

Early numerical simulation packages utilised either finite element or finite difference techniques, the former being the most popular and versatile for certain engineering

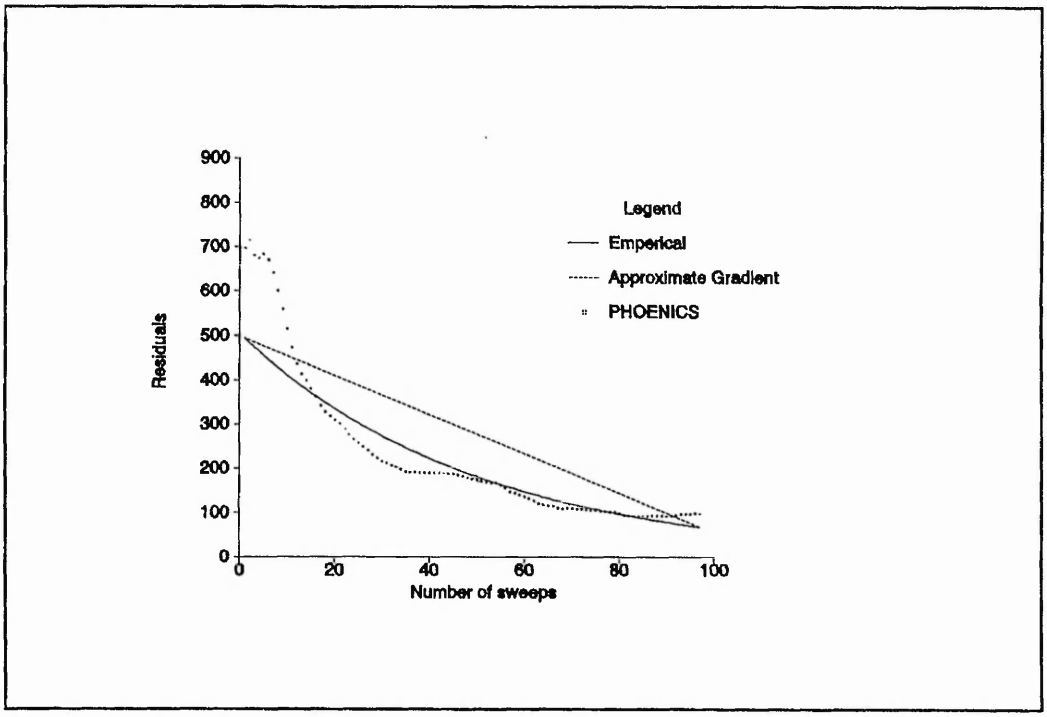


Figure 8.7: Approximate gradient analysis for assessing the convergence of the solution residuals

problems. Due to the need for producing results which are grid independent, extensive research work on grid optimisation has been performed. The main thrust of grid optimisation research is towards the application to finite element techniques. However, finite difference adaptive grid refinement has been investigated, Girdinio et al. (1983). The method presented by Girdinio et al. (1983) applies to both finite element and finite volume techniques, and implements a grid iteration method whereby the solution of the previous analysis is manipulated to modify the grid for the subsequent analysis. The method generates a normalised function, equation **(8.3)**, for each cell within the domain, and refines the grid depending upon the spacial gradient of the function.

$$f = \sum_{i=1}^{N} \left\{ \alpha_1^0 \, \phi_i + \alpha_i^1 \, | \, grad \, \phi_i \, | + \alpha_i^2 \, | \, grad \, | \, grad \, \phi_i \, | \, | \right\} \qquad (8.3)$$

where

$$\alpha_i^j - \alpha_i^{j'} \alpha_i^{j''} \tag{8.4}$$

$$j = 0, 1, 2$$
$$i = 1, 2, 3, 4, ..., \text{Number of } \phi s$$

$\alpha_i^{j'}$ *are factors to control grid generation*

$\alpha_i^{j''}$ *are factors for normalisation of functions to be linearly combined*

ϕ_i *is the* i^{th} *dependent variable*

The terms $|\text{grad } \phi|$ and $|\text{grad}|\text{grad } \phi||$ are obtained from the truncated Taylor series expansion to the second order. A thorough understanding of the values assigned for the α's needs to be obtained.

Implementing the technique with values of α obtained from Viviani (1978) and Molinari and Viviani (1979) into PHOENICS gave encouraging results for a purely thermal analysis. Manually establishing a grid independent solution required in excess of ten progressive analyses. Each analysis increased the fineness of the mesh. Using the grid iteration method, three iterations were required which gave approximately the same cell density and hence the same result.

One limiting factor placed upon the method is that vectorial dependent variables cannot be used for the determination of the function given by (8.3), only scalar variables. To overcome this problem it is anticipated that a scalar quantity can be established as a function of the velocity. Such a quantity would be shear stress, τ, given by ...

$$\tau = \mu \left(\frac{\partial u}{\partial y} \right)$$

8.4 Conclusions

Two different areas have been discussed relating to (1) the monitoring and control of the PHOENICS solution algorithm, and (2) post processing grid optimisation.

Simulating manual control of the PHOENICS solution algorithm would be possible by accessing residual values and the monitor spot values from within GROUND, or the user defined routines, Appendix I, linked to GROUND, and using the gradient and scatter analysis data. Heuristic values for the correlation coefficient and the approximate residual gradient were not established for any particular analysis. To progress the feasibility into a working facility, a research programme could be initiated whereby the monitoring of the residuals and spot values, along with the approximate gradients, such that valid heuristics could be established for a given set of problems. These heuristics would then be coded into standard production rules in FORTRAN, located within the user defined routines linked to GROUND, and be used for the control of convergence.

A feasibility study for the inclusion of pseudo real time monitoring and control of the PHOENICS solution algorithm has been presented. The implementation of the technique would require a thorough determination of the heuristic values applied to the characteristics used for the analysis. These characteristics are represented by the correlation coefficient, the average gradient and tolerances applied to the residuals and the spot values. Having determined these, and after implementing the technique, it is anticipated that a near optimum situation would result. This would consist of reduced CPU time, compared with that obtained from the use of excessively low relaxation factors. Furthermore, it would be less labour intensive compared with the traditional manual method of monitoring convergence.

With respect to post processing grid optimisation, a full programme of work would have to be undertaken to fully assess the feasibility of the grid iteration technique for use with PHOENICS. Furthermore, the possibility of integrating run time mesh adaption should be investigated. Private communications with CHAM has revealed that run time mesh adaption is already being commercially investigated, the status of which is unknown.

CHAPTER 9

CONCLUSIONS AND RECOMMENDATIONS

9.1 Conclusions

The current study has concentrated on the development of a Knowledge Based Front End (KBFE) to an existing commercial Computational Fluid Dynamics (CFD) package, PHOENICS. Initially, an expert system shell, LEONARDO, was used for the KBFE. When using a shell for the development of a system, it is important that the developer is aware of the shell's limitations of data storage, knowledge representation, and inferencing. The experience that has been gained from this study leads to the conclusion that expert system shells are too restrictive, and that the preferred option is the use of a traditional Artificial Intelligence (AI) language, LISP. The use of an AI language removes the restrictions associated with fixed knowledge representation formalisms, data storage techniques and inferencing processes found within shells.

Computational Fluid Dynamics relies on a meshed geometry. The quality of the results is affected by the mesh size, in particular the cell aspect ratios, Abbott et al. (1988). A novel technique has been developed which automatically generates an aspect ratio dependent, one dimensional mesh, given the smallest cell size, geometry, and the maximum allowed aspect ratio. The technique uses a dynamic, generalised, Fourier series to calculate local cell aspect ratios within one dimensional regions of the entire integration domain. In this manner each axis can be considered separately, and superimposed together to provide either a two or three dimensional mesh. The aspect ratio dependent mesh generation was written in C code, and incorporated into the LISP KBFE.

The experience gained using LEONARDO was valuable. Problems were encountered, such as the spontaneous corruption of knowledge bases, poor data storage facilities and the use of pseudo-lists. The pseudo-lists were only intended to store text values separated by commas, and list processing was restricted because of the inability to access data contained within the list. Furthermore, and complex lists could not be created or used. A method was developed to simulate compound lists using indexing techniques. The routines that were written to overcome all of the deficiencies associated with data storage severely affected the response of the system. Mathematical parsing was found to be extremely useful in the prototype KBFE for reducing mathematical expressions to numerical values.

The LISP version of the KBFE makes use of the experience gained through LEONARDO by implementing some of the concepts of data storage and knowledge representation through the use of frames. Data storage was provided through the use of facts and objects. The facts were categorised under assertion templates to reduce the quantity of pattern matching required, this was achieved by only matching the template as opposed to all of the assertions. The assertions were used to store boundary condition data. The objects provided storage for PHOENICS variables, and used LISP structures to simulate frames. The slots within each object allowed the inference engine to use various methods of establishing values to different objects. Multiple rulebases provided categorised knowledge. Each rulebase contained the rules and an inference network. Inference networks stream line the rulebases so that the inference engine only considers relevant rules. Base rules in the rulebases are those rules which have no link between their consequents and the antecedents of any other rule in the same rulebase. Forward chaining commences on the base rules, and backward chaining is used when an antecedent cannot be proved to be correct. The use of slots were extended for use in the rulebase language for asking the user questions. Various consequent firing modes have been established to suite different data synthesis requirements. A complete system has been developed which implements many different techniques. Increased flexibility has been experienced through the development of the system using LISP, compared with the restrictions of using a shell. One major advantage of using a fundamental language is that, if the knowledge representation technique is not available then the restriction is not present because the technique can be developed.

The development of the KBFE has incorporated a rudimentary user model, which has the ability to consider only three types of user: Novice, Experienced, and Advanced. The fundamental nature of KBFEs implies that any type of user should be able to use the system. This requires that improved user modelling concepts should be incorporated into the system. Incorporating such models in a system is a research field in its own right. Throughout the development of the system only one user type has been considered, which consists of a proficient user of PHOENICS who is an engineer, knowledgeable in the field of fluid mechanics. To incorporate other user models and to expand the availability to other engineers who are not proficient users of PHOENICS, would required further research into the area of user model design and implementation.

Pseudo real time control of the PHOENICS solution algorithm has been tentatively investigated. The technique emulates the heuristics used by manually monitoring and controlling the convergence by incorporating, into the PHOENICS executable file, user defined FORTRAN code, Appendix I. The code accesses stored values for each dependent variable, and their respective residual values from appropriate PHOENICS arrays. The residuals are filtered using conventional statistical techniques and then examined for scatter and a change in approximate gradient between a predefined number of sweeps. The results of the assessment would enable appropriate action to be taken in order to control the convergence. To further aid the assessment a similar process for the dependent spot values could be carried out. Finally, for a converged solution mass continuity must be satisfied. Post processing grid optimisation was also investigated, using the grid iteration technique, Girdinio et al. (1983).

9.2 Recommendations for further work

The following recommendations are aimed to provide possible directions for future work with respect to the present research.

1. The presentation facilities provided as part of the user interface are for user dialogue only. The inclusion of a graphical interface should be addressed, whereby particular attention is directed towards the entry of the geometry and corresponding boundary conditions. Screens for dialogue, error messages and other diagnostic facilities should be provided. Essentially, a full user interface should be created that utilises the developed inferencing processes, knowledge representation and data storage techniques.

2. User model research is a field in its own right, and as such attention should be directed towards the inclusion of more complex models. This would allow more appropriate facilities to be incorporated.

3. Manual grid generation relies on pattern recognition to a certain degree, especially when using Body Fitted Coordinates (BFCs). Neural networks have shown promise for pattern recognition, and as such could be potentially useful for grid generation.

4. Further research into the implementation of the pseudo real time monitoring and control of the solution algorithm should be carried out. This would include developing a programme of experiments to extract the necessary parameters for the scatter and gradient relationships relative to differing convergence rates. These parameters, and their relationship, would form the basis of the deep knowledge required to fully implement the control process described in chapter 8.

5. Post processing of the analysis, to assess for valid results from the requirements specified by the user should be fully addressed. One area was briefly investigated: iterative grid optimisation. An extension of this should be performed with the view of possibly incorporating adaptive grid optimisation, i.e. grid optimisation during solution.

REFERENCES

Abbott, G. D., Blake, K. R., and Sheikholeslami, M. Z., 1988. Feasibility of using a Knowledge-Based Expert System in Computational Fluid Dynamics. *Conference on Computers in Engineering, San Francisco, 31 July to 4 August 1988, 377-382.*

Alty, J.L., and Coombs, M. J., 1984. *Expert Systems: Concepts and examples*, NCC publications.

Ambroziak, J. R., and Kleiber, M., 1990. A blackboard consultation system for constitutive modelling in solid mechanics. *In: Kleiber, M., (Ed), Artificial Intelligence in Computational Engineering, 75-95.*

Ambroziak, J. R., and Kleiber, M., 1991. Blackboard consultation in Solid Mechanics. *Engineering Applications in Artificial Intelligence, Vol 4, No 2, 85-95.*

Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., 1984. *Computational Fluid Mechanics and Heat Transfer.* Series in Computational Methods in Mechanics and Thermal Sciences, Hemisphere publishing, McGraw-Hill.

Aseo, J., 1988. Expert Systems: Picking the Right Problem. *ESD: THE Electronic System Design Magazine, 63-68.*

Ashforth-Frost, S., Wang, L. S., Jambunathan, K., Graham, D. P., and Rhine, J. M., 1992. Application of Image Processing to Liquid Crystal Thermography. *IMechE International Conference on Optical Methods and Data Processing in Heat and Fluid Flow, City University, London, 2-3 April 1992, 121-126.*

Banwell, L., 1989. Some thoughts on user modelling for the individual library user. *Proceedings of the Eleventh BCS IRSG Research Colloquium on Information Retrieval, 5/6 July 1989, 83-91.*

Barber, E., 1984. *In: Bundy, A. (Ed), 1984. ALVEY IKBS Research Theme Workshop: Intelligent Front Ends 2, University of Sussex, 10/11 July 1984. IEE, Hitchin. 6-8.*

Barstow, D., Duffey, R., Smoliar, S., and Vestal, S., 1982. An overview of ΦNIX. *Proceedings of the National Conference on Artificial Intelligence, American Association for Artificial Intelligence, Pittsburgh, Pennsylvania, August 1982.*

Bennett, J. S., and Englemore, R. S., 1979. SACON: A Knowledge-Based Consultant for Structural Analysis. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979.*

Benyon, D., 1987. User Models: What's the purpose. *In: Cooper, M., and Dodson, D., (Eds), ALVEY Knowledge based systems club, Intelligent Interfaces Special Group, Proceedings of the Second Intelligent Interfaces Meeting, The City University, May 28/29, 1987, 3-14.*

Blacker, T. D., and Stephenson, M. B., 1991. PAVING: A new approach to automated quadrilateral mesh generation. *International Journal for numerical methods in engineering, vol. 32, 811-847.*

Blacker, T. D., Stephenson, M. B., Mitchiner, J. L., Phillips, L. R., and Lin, Y. T., 1988. Automated Quadrilateral Mesh Generation: A Knowledge System Approach. *Proceedings of the ASME Winter Annual Meeting, Chicago, Illinois, 27 November to 2 December 1988, 1-9.*

Blacker, T. D., Mitchiner, L. R., Phillips, L. R., and Lin, Y. T., 1988. Knowledge System approach to automated two-dimensional quadrilateral mesh generation. *Proceedings of the ASME Conference on Computers in Engineering, Volume 3, 1988, 153-162.*

Brouwer-Janse, M. D., 1990. AI Technologies for user interfaces: knowledge-based front-ends. *Proceedings of the IEE Colloquium on AI in the User Interface, 27 April 1990, (Digest No 118), 2/1-2/3.*

Bundy, A., 1984a. Intelligent Front Ends. *Intelligent Front Ends in Expert Systems.* Pergamon Infotech State of the Art Report "Expert Systems", Pergamon Infotech Ltd., 1984, 1-12.

Bundy, A. (Ed), 1984b. *ALVEY IKBS Research Theme Workshop: Intelligent Front Ends 2,* University of Sussex, 10/11 July 1984. IEE, Hitchin.

Bundy, A., Sharpe, B., Uschold, M., and Harding, N., 1984. *ALVEY IKBS Research Theme Workshop: Intelligent Front Ends, 26/27 September 1983, Cosener's Houser, Abingdon, England.* IEE, Stevenage.

Clarke, J. A., 1990. Advanced Design Tools for Energy Conscious Building Design. *Proceedings of the first World Renewable Energy Congress: Energy and the Environment. Into the 1990s, Vol 4, 2265-2277.*

Clarke, J. A., and MacRandal, D., 1991. An Intelligent Front-End for computer-aided building design. *Artificial Intelligence in Engineering, Special Issue: Intelligent Front-Ends, Guest Editor: Clarke, J. A., Volume 6, No 1, January 1991, 36-45.*

Clarke, J. A., Rutherford, J., and MacRandal, D., 1988. An Intelligent Front End for Building Energy Simulation. *Working Conference of users of Simulation Hardware, Ostend, 6/8 September 1988,* 165-171.

Clocksin, W.F., and Mellish, C.S., 1984. *Programming in PROLOG,* Second Edition, Springer-Verlag, New York.

COM(86) 687 final, COMMISSION OF THE EUROPEAN COMMUNITIES. *ESPRIT, The first phase progress and results, Communication from the Commission to the Council.* Brussels, 8 December 1986.

Conner, R. R., and Purdon, D. J., 1986. PAN Air Knowledge System. *American Institute of Aeronautics and Astronautics, 86-0239, January.*

Cornali, D. J., 1990. Four Adaptive Strategies for Knowledge-Based Front-Ends. *Proceedings of the UK IT 1990 Conference (Conference Publication Number 316), 371-378.*

Dannenhoffer, III, J. F., and Baron, J. R., 1987. A Hybrid Expert System for Complex CFD Problems. *American Institute of Aeronautics and Astronautics, paper 87-1111.*

Drechsler, F. S., and Peppard, J. W., 1988. An Intelligent Front End / Expert System interface to a CIM module. *In: Trappl, R., (Ed), Proceedings of the meeting on Cybernetics and System Research, Austria, 5/8 April 1988, 785-792.*

Edmonds, E., and McDaid, E., 1990. An architecture for knowledge-based front ends. *Knowledge-Based Systems, Vol 3, No 4, December 1990, 221-224.*

Elmaghraby, A. S., and Jagannathan, V., 1985. An Expert System for Simulationists. *In: Birtwistle, G., (Ed), Artificial Intelligence, Graphics and Simulation, 106-109.*

Emmett, J., 1987. Intelligent Front Ends promote plant efficiency. *Control and Instrumentation GB, Volume 19, Part 4, April 1987, 74-75.*

Engquist, B., and Smedsaas, T., 1980. Automatic Computer code generation for hyperbolic and parabolic differential equations. *Society for Industrial and Applied Mathematics, Journal of Scientific and Statistical Computing, Vol 1, No 2, June 1980, 249-259.*

Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R., 1980. The HEARSAY-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys, Vol 12 No 2, June 1980, 213-253.*

Eustace, P., 1985. Has ESPRIT got its sums right, now? *The Engineer, 24 January 1985, 12-13.*

Fang, C., ZhengZhong, W., RenShou, T., 1988. The Expert System in Discrete-Event Simulation System. *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, 8/12 August 1988, Beijing and Shenyang, China, Volume 2, 1054-1056.*

Fink, R. K., Callow, R. A., Larson, T. K., and Ransom, V. H., 1987. *ATHENA AIDE: An Expert System for ATHENA code input model preparation.* Idaho National Engineering Laboratory EG and G Idaho Inc. Idaho Falls (USA), 7p.

Ford, B., Hague, S. J., and Iles, R. M. J., 1989. Numerical Knowledge-based systems. *Mathematics and Computers in Simulation, 31, 395-400.*

Forsyth, R., 1988. Software Review: LEONARDO. *Expert Systems, Vol 5 No 2, May 1988, 160-164.*

Gevarter, W. B., 1983. Expert Systems: Limited but powerful. *IEEE Spectrum, August 1983, 39-45.*

Gilding, B. H., 1988. A Numerical Grid Generation Technique. *Computers and Fluids, Vol 16, No 1, 47-58.*

Guariso, G., Hitz, M., and Werthner, H., 1989. An intelligent simulation model generator. *Simulation, August 1989, 57-66.*

Hartle, S. L., Li, H., Lai, E., Jambunathan, K., and Button, B. L., 1993. A Knowledge Based Approach to Data File Checking for Numerical Simulation Packages using an Expert System Shell. *Submitted for publication in Engineering Applications of Artificial Intelligence.*

Harvey, J. J., 1988. Expert Systems: An introduction. *International Journal of Computer Applications in Technology, Vol 1, Nos 1/2, 53-60.*

Hayes-Roth, B., 1984. A Blackboard Model of Control. *Stanford University, Heuristic Programming Project, Report No. 83-38.*

Hayes-Roth, B., 1985. A Blackboard Architecture for Control. *Artificial Intelligence, 26, 251-321.*

Jambunathan, K., Lai, E., Hartle, S.L., and Button, B.L., 1991a. Development of an Intelligent Front End for a Computational Fluid Dynamics Package. *Artificial Intelligence in Engineering, Volume 6, Number 1, January 1991. 27-35.*

Jambunathan, K., Lai, E., Hartle, S.L., and Button, B.L., 1991b. Development of an Intelligent Front End: An Experience. *Engineering Applications of Artificial Intelligence, Volume 4, Number 5, 385-35.*

Jambunathan, K., Lai, E., Hartle, S.L., and Button, B.L., 1991b. Development of an Intelligent Front End using LISP. *In the Proceedings of the Seventh International Conference on the Applications of Artificial Intelligence in Engineering, AIENG 92, University of Waterloo, Ontario, Canada, 14/16 July 1992, 229-243.*

Jerrams-Smith, J., 1987. User Modelling. *In: Cooper, M., and Dodson, D., (Eds), ALVEY Knowledge based systems club, Intelligent Interfaces Special Group, Proceedings of the Second Intelligent Interfaces Meeting, The City University, May 28/29, 1987, 23-24.*

Kathawala, Y., Elmuti, D., and Timpner, C. J., 1989. Artificial Intelligence: the key to the future? *International Journal of Computer Applications in Technology, Vol 2, No 1, 56-61.*

Kernighan, B. W., and Ritchie, D. M, 1988. *The C Programming Language, Second Edition.* Prentice-Hall Software Series.

Khabaza, T., Sloman, A., and Law, A., 1988. Using a human inference engine. *AISB Quarterly, Issue 67, 4-7.*

Knight, B., and Petridis, M., 1992. FLOWES: An Intelligent Computational Fluid Dynamics System. *Engineering Applications of Artificial Intelligence, Volume 5, Number 1, 51-58.*

Kurstedt, Jr. H. A., Lee, K. W., Mendes, P. M., and Berube, D. S., 1988. The Responsive System: A New Challenge for AI. *Proceedings of the first International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE-88, Vol 1, 177-184.*

Kutler, P., Mehta, U. B., and Andrews, A., 1985. Potential Application of Artificial Intelligence Concepts to Numerical Aerodynamic Simulation. *Proceedings of the Ninth International Conference on Numerical Methods in Fluid Dynamics, Lecture Notes is Physics, Vol 218, Berlin, West Germany: Springer-Verlag.*

LEONARDO (Versions 3.17, 3.18, 3.20), Creative Logic Ltd., Brunel Science Park, Kingston Lane, Uxbridge, Middlesex, England, UB8 3PQ.

MacRandal, D., 1987. The Application of Intelligent Front Ends in Building Design. *In: Sriram, D., and Adey, R. A., (Eds), Artificial Intelligence in Engineering: Tools and Techniques.*

Mao, X., 1988. A knowledge-oriented human-computer interface model. *Proceedings of the 1988 IEEE International Conference on Systems, Man, and Cybernetics (IEEE Cat No. 88CH2556-9), Beijing and Shenyang China, 8-12 August 1988, 670-673.*

MARC User information manual, available from MARC Analysis Research Corporation, 260 Sheridan Avenue, Court House Plaza, Suite 200, Palo Alto, California 94306.

McConnell, B. A., and McConnell, N. J., 1988. A Starter's guide to Artificial Intelligence. *Collegiate Microcomputer, August, Vol VI, No 3, 239-246.*

Mehta, U. B., and Kutler, P., 1984 *Computational Aerodynamics and Artificial Intelligence.* National Aeronautics and Space Administration, NASA Technical Memorandum 85994.

Mehta, U. B., 1986. Knowledge based systems for Computational Aerodynamics and Fluid Dynamics. *In: Kowalik, J.S. (Ed), Knowledge Based Problem Solving, 183-212.*

Michelsen, C., Dreicer, J., and Morgeson, D., 1988. Strategic Engagement Analysis Tool (SEAT). *Proceedings of the Tools for Simulation Profession, 18/21 April 1988, 26-29.*

Moore, R. L., 1985. Adding Real-Time Expert System Capabilities to Large Distributed Control Systems. *Control Engineering USA, Volume 32, Part 4, April 1985, 118-121.*

Morley, R. E., and Taylor, W. A., 1986. Why bother with Expert Systems? *Digital Design, July 1986, 47-49.*

Murdoch, J., and Barnes, J. A., 1985. *Statistical Tables: for Science, Engineering and Management Stud..,* Second Edition. MacMillian Education Ltd.

Oakley, B., and Owen, K., 1989. *ALVEY: Britain's Strategic Computing Initiative.* MIT Press, London, England.

O'Keefe, R., 1986. Simulation and expert systems - A taxonomy and some examples. *Simulation 46:1, 10-16.*

Pang, G. K. H., 1988. An Intelligent Front End for a control system design and analysis package. *Proceedings of the Fourth IFAC Computer Aided Design in Control Systems symposium, Beijing, China, 23-25 August, 1988, 329-334.*

Patankar, S. V., 1980. *Numerical Heat Transfer and Fluid Flow.* Series in Computational Methods in Mechanics and Thermal Sciences, Hemisphere Publishing Corporation, London.

PHOENICS (Versions 1.4, 1.5.3 and 1.6). Parabolic Hyperbolic Or Elliptic Numerical Integration Code Series, Concentration Heat And Momentum (CHAM) Limited, Bakery House, 40 High Street, Wimbledon, London, SW19 5AU, England, UK.

POPLOG (Versions 13.6, 14.1), Integral Solutions Limited, University of Sussex.

Prat, A., Lores, J., Fletcher, P., and Catot, J. M., 1990. Back-end manager: an interface between a knowledge-based front end and its application subsystems. *Knowledge-Based Systems, Vol 3, No 4, December 1990, 225-229.*

Ramirez, M. R., and Belytschko, T., 1989. An Expert System for Setting Time Steps in Dynamic Finite Element Programs. *Engineering with Computers 5, 205-219.*

Ramsay, A., 1984. *ALVEY IKBS Research Theme Workshop: Intelligent Front Ends 2,* University of Sussex, 10/11 July 1984. IEE, Hitchin, 48-50.

Reddy, M., and O'Hare, G. M. P., 1991. The blackboard model: a survey of its application. *Artificial Intelligence Review, 5, 169-186.*

Reid, I., 1990. Interfaces to numerical packages. *Computer Physics Communications, 61 (1/2), 141-149.*

Roberts, G. O., 1971. Computational Meshes for Boundary Layer Problems. *Proceedings of the Second International Conference on Numerical Methods in Fluid Dynamics, Lecture Notes in Physics, Vol 8, 171-177.*

Ross, P., 1984. The Virtues and Problems of User Modelling. *IEEE Colloquium on Intelligent Knowledge Based Systems. 5/1-5/4.*

Ryskin, G., and Leal, L. G., 1984. Numerical solution of free-boundary problems in fluid mechanics. Part 1: The finite-difference technique. *Journal of Fluid Mechanics, Vol 148, 1-17.*

Ryskin, G., and Leal, L. G., 1983. Orthogonal Mapping. *Journal of Computational Physics, 50, 71-100.*

Schildt, H., 1990. *C: The Complete Reference, Second Edition,* Osbourne McGraw-Hill, California, USA.

Shortliffe, E. H., 1975. *MYCIN: A computer-based computer program for advising physicians regarding antimicrobial therapy selection.* PhD Thesis, Stanford University, California, USA.

Steel, Jr., G. L., 1990. *COMMON LISP, The Language. Second Edition.* Digital Press.

Strandhagen, J. O., 1989. Expert Systems in Manufacturing Simulation. *In: Browne, J., (Ed), Knowledge Based Production Management Systems, 83-93.*

Tangen, K., and Wretling, U., 1986. Intelligent Front Ends to Numerical Simulation Programs. *In: Brammer, M. A., (Ed), Research and Development in Expert Systems III, 254-265.*

Taylor, C., and Hughes, T. G., 1981. *Finite Element programming of the Navier-Stokes equations.* Pineridge Press, Swansea.

Teresko, J., 1985. Artificial Intelligence: More fact than fantasy. *Industry week, 21 January 1985, 53-60.*

Thomas, G. B., Thomas, R. C., and Lai, C. C., 1988. An Expert System Interface to a suite of rotordynamic programs. *Institute of Mechanical Engineers Conference Proceedings "Vibrations in Rotating Machinery", 621-626.*

Thomas, R. C., Thomas, G. B., and Littler, J. G., 1990. The cognitive role of an engineer in a diagnostic task. *Proceedings of the UK IT 1990 Conference, Conference Publication Number 316, Southampton, UK, 19/22 March 1990, 259-263.*

Tong, S. S., 1985. Design of Aerodynamic bodies using Artificial Intelligence/Expert System technique. *AIAA paper 85-0112, American Institute of Aeronautics and Astronautics, Aerospace Sciences Meeting, 23rd Reno, NV, 14/17 January 1985, 1-6.*

Tou, F. N., Williams, M. D., Fikes, R., Henderson, A., and Malone, T., 1982. RABBIT: An Intelligent Database Assistant. *Proceedings of the American Association for Artificial Intelligence, Part 82, 314-318.*

TR100, 1989. *The PHOENICS Beginner's Guide.* CHAM Ltd., Wimbledon, London, SW19 5AU

TR200, 1989. *The PHOENICS Reference Manual.* CHAM Ltd., Wimbledon, London, SW19 5AU

Uschold, M., Harding, N., Muetzelfeldt, R., and Bundy, A., 1984. *An Intelligent Front End for Ecological Modelling.* Research paper 223, Department of Artificial Intelligence, Edinburgh University, Edinburgh, UK.

Uzel, A. R., Edwards, R. J., and Button, B. L., 1988. A study into the feasibility of an Intelligent Knowledge Based System (IKBS) in Computational Fluid Mechanics (CFM). *Engineering Applications of Artificial Intelligence, Vol 1, September 1988, 187-193.*

Vogel, A. A., 1989. *A knowledge-based approach to automated flow field zoning for computational fluid dynamics.* PhD Thesis, Stanford University, UMI-8912946.

Wager, D. M., 1984. *Expert systems and the construction industry. A study of expert systems, their nature, possible use and implications for the construction industry.* Construction Industry Computing Association.

Wang, M., and Georgiadis, J. G., 1989. Numerical Generation of Orthogonal Boundary-Fitted Grids for Heat and Fluid Flow. *National Heat Transfer Conference, HTD Vol 110, Numerical Heat Transfer With Personal Computers and Supercomputing, 65-71.*

Waterman, D. A., 1986. How do Expert Systems differ from conventional programs? *Expert Systems, January 1986, Vol 3, No 1, 16-19.*

Webster, R., and Miner, L., 1982. Expert Systems Programming Problem-Solving. *Technology/2, January/February 1982, 62-91.*

Weiss, S., and Kulikowski, C., 1979. EXPERT: A System for Developing Consultation Models. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, 942-947.*

Weiss, S., Kulikowski, C., Apte, C., Uschold, M., Patchett, J., Brigham, R., and Spitzer, B., 1982. Building Expert Systems for Controlling Complex Programs. *Proceedings of the American Association for Artificial Intelligence, AAAI-82, 322-326.*

Whitmore, R. W. H., 1991. KASTLE: A KBFE for a Software Library. *Proceedings of the Teaching Company Scheme Expert Systems Seminar, Vol 1, No 6, 20 November 1991.*

Whittkowski, K. M., 1991. A structured visual language for a knowledge-based front-end to statistical analysis systems in biomedical research. *Computer Methods and Programs in Biomedicine, 35, Issue 1, May 1991, 59-67.*

Winston, P. H., 1984. *Artificial Intelligence. Second Edition.* Addison-Wesley.

Winston, P. H., and Horn, B. K. P., 1989. *LISP, Third Edition.* Addison-Wesley.

Wolstenholme, D. E., and Nelder, J. A., 1986. A front end for GLIM. *In: Haux, R., (Ed), Expert Systems in Statistics, Gustav Fischer, 155-177.*

Wong, F. S., Dong, W. M., and Blanks, M., 1988. Coupling of Symbolic and numerical computations on a microcomputer. *Artificial Intelligence in Engineering, 1988, Vol 3, No 1, 32-38.*

Xuesi, J., and Zhengzhong, W., 1988. Simulation Integration Algorithms Selecting Expert System. *Proceedings of the 1988 IEEE International Conference on Systems, Man, and Cybernetics (IEEE Cat. No. 88CH2556-9), Vol 2, 1068-1070.*

Yu, T., Dai, G., and Zhu, Z., 1988. Development of an expert system for computer-aided control systems design. *Proceedings of the Fourth IFAC Computer Aided Design in Control Systems Symposium, 1988, 353-358.*

Yuasa, T. and Hagiya, M., 1986. *Introduction to Common Lisp.* Academic Press Inc.

APPENDIX A

Fourier series coefficients for the profile function shown in Figure 3.15

$$
\begin{aligned}
a_0 = \frac{1}{\pi}\Bigg\{ & \frac{\pi}{2} - \frac{\alpha_3}{2\pi}(2\pi - \alpha_3) + \bar{\psi}(\alpha_3 + \alpha_2) \\
& - \frac{m_1}{2}(\alpha_3^2 + \alpha_2^2 - 2\alpha_2\alpha_3) \\
& + \frac{m_2}{2}(\alpha_2^2 + \alpha_1^2 - 2\alpha_1\alpha_2) \\
& + \frac{m_3}{2}(2\alpha_2\alpha_4 - \alpha_2^2 - \alpha_4^2)\Bigg\}
\end{aligned}
\tag{A1}
$$

$$
\begin{aligned}
a_n = \frac{1}{\pi}\Bigg\{ & \frac{(\alpha_3 - \pi)}{\pi n}\sin(n\alpha_3) + \frac{\cos(n\alpha_3) - \cos(n\pi)}{\pi n^2} \\
& + \frac{m_1}{n}(\alpha_2\sin(n\alpha_2) - \alpha_3\sin(n\alpha_3)) + \frac{m_1}{n^2}(\cos(n\alpha_2) - \cos(n\alpha_3)) \\
& + \frac{m_1\alpha_2 + \bar{\psi}}{n}(\sin(n\alpha_3) - \sin(n\alpha_2)) \\
& + \frac{\bar{\psi}}{n}(\sin(n\alpha_1) + \sin(n\alpha_2)) + \frac{m_2}{n}(\alpha_2\sin(n\alpha_2) - \alpha_1\sin(n\alpha_1)) \\
& + \frac{m_2}{n^2}(\cos(n\alpha_2) - \cos(n\alpha_1)) + \frac{\bar{\psi} - m_2\alpha_1}{n}(\sin(n\alpha_2) - \sin(n\alpha_1)) \\
& + \frac{m_3}{n}(\alpha_4\sin(n\alpha_4) - \alpha_2\sin(n\alpha_2)) + \frac{m_3}{n^2}(\cos(n\alpha_4) - \cos(n\alpha_2)) \\
& - \frac{m_3\alpha_4}{n}(\sin(n\alpha_4) - \sin(n\alpha_2))\Bigg\}
\end{aligned}
\tag{A2}
$$

$$
\begin{aligned}
b_n = \frac{1}{\pi}\Bigg\{ & \frac{\alpha_3 - \pi}{n\pi}\cos(n\alpha_3) - \frac{\sin(n\alpha_3)}{\pi n^2} \\
& + \frac{m_1}{n}(\alpha_2\cos(n\alpha_2) - \alpha_3\cos(n\alpha_3)) + \frac{m_1}{n^2}(\sin(n\alpha_3) - \sin(n\alpha_2)) \\
& + \frac{m_1\alpha_2 + \bar{\psi}}{n}(\cos(n\alpha_3) - \cos(n\alpha_2)) + \frac{\bar{\psi}}{n}(\cos(n\alpha_2) - \cos(n\alpha_1)) \\
& + \frac{m_2}{n}(\alpha_1\cos(n\alpha_1) - \alpha_2\cos(n\alpha_2)) + \frac{m_2}{n^2}(\sin(n\alpha_2) - \sin(n\alpha_1)) \\
& + \frac{\bar{\psi} - m_2\alpha_1}{n}(\cos(n\alpha_1) - \cos(n\alpha_2)) + \frac{m_3}{n}(\alpha_2\cos(n\alpha_2) - \alpha_4\cos(n\alpha_4)) \\
& + \frac{m_3}{n^2}(\sin(n\alpha_4) - \sin(n\alpha_2)) - \frac{m_3\alpha_4}{n}(\cos(n\alpha_2) - \cos(n\alpha_4))\Bigg\}
\end{aligned}
\tag{A3}
$$

APPENDIX B

Finite volume aspect ratio dependent C code

```
#include <stdio.h>
#include <math.h>

#define odd(i) (fmod((i), 2) == 1.0 ? 1 : 0)
#define even(i) (fmod((i), 2) == 0.0 ? 1 : 0)


/* Declare the functions */


float F(float theta);
void get_A0(float PI, float psi, float alpha1, float alpha2,
            float alpha3, float alpha4, float m1, float m2, float m3);
void get_AN(float PI, float psi, float alpha1, float alpha2,
            float alpha3, float alpha4, float m1, float m2,
            float m3);
void get_BN(float PI, float psi, float alpha1, float alpha2,
            float alpha3, float alpha4, float m1, float m2,
            float m3);

float PSI_N(float value, float alpha);

/* Declare GLOBAL variables */

float a0, an[101], bn[101];

main()
{
        float AR, delta, h, eta, Y[1000];
        float T;
        float lambda;
        float theta, f, f_last, last_f_last;
        float PI = 3.141592654;
        float alpha1 = 0.0, alpha2 = 0.0, alpha3 = PI, alpha4 = PI;
        float beta = 0.0, psi = 1.0;
        float m1 = 1.0 / PI;
        float m2 = -1.0 / PI;
        float m3 = -1.0 / PI;
        float psi_N, dummy, alpha;

        int i = 0, reset_psi = 'N', j, k, alpha1_i = -1;

        printf("\nEnter the overall height of the region: ");
        scanf("%f", &h);
        printf("\nEnter the maximum allowed aspect ratio: ");
        scanf("%f", &AR);
        printf("\nEnter the minimum cell size: ");
        scanf("%f", &delta);
        printf("\nEnter alpha (0.0, 0.5, or 1.0): ");
        scanf("%f", &eta);

        T = delta / h;        /* T is the datum metric */
```

```
if (eta == 1.0)
   alpha = 0.0;
else
   alpha = eta;

Y[1] = alpha - 1.0;
f_last = 0.0;
f = 0.0;
get_A0(PI, psi, alpha1, alpha2, alpha3, alpha4, m1, m2, m3);
get_AN(PI, psi, alpha1, alpha2, alpha3, alpha4, m1, m2, m3);
get_BN(PI, psi, alpha1, alpha2, alpha3, alpha4, m1, m2, m3);


while ((alpha - Y[i]) >= (0.1 * T)) {

   theta = PI*Y[i] / (1.0 - alpha);
   last_f_last = f_last;
   f_last = f;
   f = F(theta);
   lambda = pow(10.0, (f * log10(AR)));
   Y[i+1] = Y[i] + lambda * T;

   if (i == alpha1_i) {
      alpha1 = fabs(2.0*PI*Y[i]);
      alpha4 = fabs(2.0*PI*(0.5-T));
      m2 = (psi - beta) / (alpha1 - alpha2);
      m3 = beta / (alpha2 - alpha4);
      get_A0(PI, psi, alpha1, alpha2, alpha3, alpha4,
            m1, m2, m3);
      get_AN(PI, psi, alpha1, alpha2, alpha3, alpha4,
            m1, m2, m3);
      get_BN(PI, psi, alpha1, alpha2, alpha3, alpha4,
            m1, m2, m3);
   }

   if (Y[i+1]>0.0 && Y[i]<0.0 && reset_psi == 'N') {
      alpha1_i = i;
      --i;
      alpha2 = fabs(PI*Y[i] / (1.0 - alpha));
      alpha3 = fabs(PI*Y[i-1] / (1.0 - alpha));
      beta = last_f_last;
      psi_N = PSI_N(alpha2/(PI*AR*T), alpha);
      psi = log10(alpha2/(PI*psi_N*T)) / log10(AR);
      m1 = (psi - beta) / (alpha3 - alpha2);
      get_A0(PI, psi, alpha1, alpha2, alpha3, alpha4,
            m1, m2, m3);
      get_AN(PI, psi, alpha1, alpha2, alpha3, alpha4,
            m1, m2, m3);
      get_BN(PI, psi, alpha1, alpha2, alpha3, alpha4,
            m1, m2, m3);
      reset_psi = 'Y';
      --i;
   }
```

```
        ++i;
    }

    Y[i] = alpha;

    if (eta == 0.0)
        for(j=1; j <= i; j++)
            Y[j] += 1.0;
    else if (eta == 0.5)
            for(j=1; j <= i; j++)
                Y[j] += 0.5;
    else {
        j = 0;
        k = i;
        while (k >= j) {
            dummy = fabs(Y[j]);
            Y[j] = fabs(Y[k]);
            Y[k] = dummy;
            j++;
            k--;
        }
    }
    for(j=1; j <= i; j++) Y[j] = h * Y[j+1];
}

float F(float theta)
{
    float f = a0 / 2.0;
    int i = 1, j;

    if (fabs(theta) == 3.141592654)
        f = 0.0;
    else
        for(i=1; i <= 100; i++)
            f += an[i]*cos(i*theta) + bn[i]*sin(i*theta);
    return f;
}


float PSI_N(float value, float alpha)
{
    float integral_part, fractional_part;

    fractional_part = modf(value, &integral_part);

    if (fractional_part > 0.0) ++integral_part;
    if even(integral_part) integral_part += 2.0 * alpha;

    return integral_part;
}
```

```
void get_A0(float PI, float psi_bar, float alpha1, float alpha2,
        float alpha3, float alpha4, float m1, float m2, float m3)
{
    a0  = PI / 2.0;
    a0 -= alpha3 * (2.0*PI - alpha3) / (2.0*PI);
    a0 += psi_bar * (alpha3 + alpha2);
    a0 -= m1 * (pow(alpha3, 2.0) + pow(alpha2, 2.0) -
        2.0*alpha2*alpha3) / 2.0;
    a0 += m2 * (pow(alpha2, 2.0) + pow(alpha1, 2.0) -
        2.0*alpha1*alpha2) / 2.0;
    a0 += m3 * (2.0*alpha2*alpha4 - pow(alpha2, 2.0) -
        pow(alpha4, 2.0)) / 2.0;
    a0 /= PI;
}

void get_AN(float PI, float psi_bar, float alpha1, float alpha2,
        float alpha3, float alpha4, float m1, float m2, float m3)
{
    int i;
    float n;

    for(i=1; i <= 100; i++) {

      n = i * 1.0;
      an[i]  = (alpha3 - PI) * sin(n*alpha3) / (PI*n);
      an[i] += (cos(n*alpha3) - cos(n*PI)) / (PI*pow(n, 2.0));
      an[i] += m1*(alpha2*sin(n*alpha2) -
          alpha3*sin(n*alpha3)) / n;
      an[i] += m1*(cos(n*alpha2) - cos(n*alpha3)) /
          pow(n, 2.0);
      an[i] += (m1*alpha2 + psi_bar)*(sin(n*alpha3) -
          sin(n*alpha2))/n;
      an[i] += psi_bar*(sin(n*alpha1) + sin(n*alpha2)) / n;
      an[i] += m2*(alpha2*sin(n*alpha2) -
          alpha1*sin(n*alpha1)) / n;
      an[i] += m2*(cos(n*alpha2) - cos(n*alpha1)) /
          pow(n, 2.0);
      an[i] += (psi_bar - m2*alpha1)*(sin(n*alpha2) -
          sin(n*alpha1))/n;
      an[i] += m3*(alpha4*sin(n*alpha4) -
          alpha2*sin(n*alpha2)) / n;
      an[i] += m3*(cos(n*alpha4) - cos(n*alpha2)) /
          pow(n, 2.0);
      an[i] -= m3*alpha4*(sin(n*alpha4) - sin(n*alpha2)) / n;
      an[i] /= PI;

    }
}
```

```
void get_BN(float PI, float psi_bar, float alpha1, float alpha2,
        float alpha3, float alpha4, float m1, float m2, float m3)
{
    int i;
    float n;

    for(i=1; i <= 100; i++) {
      n = i * 1.0;
      bn[i]  = (alpha3 - PI) * cos(n*alpha3) / (PI*n);
      bn[i] -= sin(n*alpha3) / (PI*pow(n, 2.0));
      bn[i] += m1*(alpha2*cos(n*alpha2) -
          alpha3*cos(n*alpha3)) / n;
      bn[i] += m1*(sin(n*alpha3) - sin(n*alpha2)) /
          pow(n, 2.0);
      bn[i] += (m1*alpha2 + psi_bar)*(cos(n*alpha3) -
          cos(n*alpha2))/n;
      bn[i] += psi_bar*(cos(n*alpha2) - cos(n*alpha1)) / n;
      bn[i] += m2*(alpha1*cos(n*alpha1) -
          alpha2*cos(n*alpha2)) / n;
      bn[i] += m2*(sin(n*alpha2) - sin(n*alpha1)) /
          pow(n, 2.0);
      bn[i] += (psi_bar - m2*alpha1)*(cos(n*alpha1) -
          cos(n*alpha2))/n;
      bn[i] += m3*(alpha2*cos(n*alpha2) -
          alpha4*cos(n*alpha4)) / n;
      bn[i] += m3*(sin(n*alpha4) - sin(n*alpha2)) /
          pow(n, 2.0);
      bn[i] -= m3*alpha4*(cos(n*alpha2) - cos(n*alpha4)) / n;
      bn[i] /= PI;
    }
}
```

APPENDIX C

Command sequence program, COMSEQ.FOR, used for the data file checker.

```
      Program Preleo

      Integer Line,Finish,NCommands
      Character Statements*50(500),Word*75,Uppercase*75

      Common /Commands/ Statements
      Common /word/ Word
      Common /Integers/ Number,NCommands

      Open(Unit=21,File='Q1.DAT',Status='UNKNOWN')
      Open(Unit=22,File='$$LEOOUT.DAT',Status='UNKNOWN')
      Open(Unit=23,File='Q11.DAT',Status='UNKNOWN')

      Number=Nelem(Statements)
      NCommands=0
      Line=0
10    Read(21,15,End=30) Word
15    Format(A75)
      Finish=Length(Word)
      Word=Uppercase(Word)
      Line=Line+1
      if (Word(1:2).eq.'  ') then
         Write(23,15) Word
         goto 20
      else
         if (Word(1:1).eq.' ') Word=Word(2:Finish)
         Write(23,15) Word
      endif
      Call Check(Line)
20    goto 10
30    Call Sort
      Call Print
      Close(21)
      Close(22)
      Close(23)
      end

      Subroutine Sort
      Character*50 Statements(500),Dummy
      Integer flag
      Common /Commands/ Statements
      Common /Integers/ Number,NCommands

15    flag=0
      Do 20 i=1,NCommands
         if (ichar(St..:ments(i)(2:2)).eq.32) then
         j=length(Statements(i))
         Do 25 k=?,j-1
25          Statements(i)(k:k)=Statements(i)(k+1:k+1)
         Statements(i)(j:j)=char(32)
         endif
         if (i.eq.NCommands) goto 20
```

```
            if (ichar(Statements(i+1)(1:1)).gt.
     1      ichar(Statements(i)(1:1))) then
            Dummy=Statements(i+1)
            Statements(i+1)=Statements(i)
            Statements(i)=Dummy
            flag=1
            endif
20      Continue
        if (flag.eq.1) goto 15
        return
        end


        Subroutine Print
        Integer Number
        Character Statements*50(500)
        Character*10 Forchar(0:2),String,IntegerToText
        Common /Commands/ Statements
        Common /Integers/ Number,NCommands

        Forchar(0)='(A1,'
        Forchar(2)=',A1)'
        i=length(Statements(NCommands))
        Statements(NCommands)(i:i)=char(32)
        j=0
        Do 10 i=1,NCommands
10      j=j+Length(Statements(i))
        String=IntegerToText(j)
        i=Length(String)
        Forchar(1)(1:i)=String
        Forchar(1)(i+1:i+2)='A1'
        write(22,Forchar) char(39),((Statements(i)(j:j),
     1  j=2,length(Statements(i))),i=1,NCommands),
     1  char(39)

        end

        Subroutine Check(Line)
        Integer Line,LPPosition,EQPosition,Position,CLength,
     1  Number,WFinish
        Character Statements*50(500),Word*75
        Common /Commands/ Statements
        Common /word/ Word
        Common /Integers/ Number,NCommands

        LPPosition=Locate(1,1,40,Word)
        if (LPPosition.eq.0) LPPosition=1000000
        EQPosition=Locate(1,1,61,Word)
        if (EQPosition.eq.0) EQPosition=1000000
        if (EQPosition.eq.LPPosition) then
           WFinish=length(Word)
           goto 4
```

```
      endif
      WFinish=Min(LPPosition,EQPosition)-1


4     Do 10 i=1,Number
        Position=Locate(1,1,44,Statements(i)) ! Locate comma
        CLength=Position-1                    ! StringFINISH=location-1
        if (CLength.gt.0) goto 5
        CLength=Length(Statements(i))  ! SFinish=Statement length
5       if (Statements(i)(3:CLength).eq.Word(1:WFinish)) then
        if (Word(1:WFinish).eq.'REAL'.or.Word(1:WFinish).eq.
     1     'INTEGER') then
          j=LPPosition
          Last=j
          k=Locate(1,LPPosition,41,Word)
6         Number=Number+1
          n=Len(Statements(Number))
          Do 7 m=1,n
7         Statements(Number)(m:m)=char(32)
          Statements(Number)(1:1)='|'
          Statements(Number)(2:2)='*'
          j=Locate(1,j,44,Word)
          if (j.eq.0) then
          j=Locate(-1,k,44,Word)
          if (j.gt.LPPosition)
     1      Statements(Number)(3:3+k-LPPosition-2)=
     1        Word(j+1:k-1)
          if (j.eq.0)
     1      Statements(Number)(3:3+k-LPPosition-2)=
     1        Word(LPPosition+1:k-1)
            goto 8
          endif
          Statements(Number)(3:3+j-Last-2)=
     1        Word(Last+1:j-1)
          Last=j
          j=j+1
          goto 6
        endif
8       Call AppendLines(i,Line)
        return
        endif
10    Continue

      end

      Subroutine AppendLines(i,Line)
      Character*50 Insert,Statements(500),Dummy
      Character*5 LineNumber,Occurances,IntegerToText
      Integer Line,NOccurances,Start,CLength,Finish,TextToInteger
      Integer NCommands
      Common /Commands/ Statements
      Common /Integers/ Number,NCommands
```

```
C Locate first comma for Occurances
      Start=Locate(1,1,44,Statements(i))

      LineNumber=IntegerToText(Line)
      CLength=length(Statements(i))      ! Get total length
      if (Start.eq.0) then     ! First time located
         NCommands=NCommands+1
         Dummy=Statements(i)
         Statements(i)=Statements(NCommands)
         Statements(NCommands)=Dummy
         i=NCommands
         Statements(i)(CLength+1:CLength+1)=',' ! Add a comma
         Statements(i)(CLength+2:Clength+3)='1,' ! 1 occurance
      else

C Locate next comma
         Finish=Locate(1,Start+1,44,Statements(i))
         NOccurances=TextToInteger(Start+1,Finish-1,
     1      Statements(i))+1
         Occurances=IntegerToText(NOccurances)
         Statements(i)=Insert(Occurances,Start+1,Finish-1,
     1         Statements(i))
      endif
      CLength=Length(Statements(i))
      Statements(i)=Insert(LineNumber,CLength+1,CLength+1,
     1         Statements(i))
      CLength=Length(Statements(i))
      Statements(i)(CLength+1:CLength+1)=','          ! Add a comma
      return
      end

      Function TextToInteger(Start,Finish,String)
      Character*(*) String
      Character SubString*10
      Integer Start,Finish,SubLength,Power,TextToInteger
      SubString=String(Start:Finish)
      SubLength=Length(SubString)
      Power=SubLength-1
      TextToInteger=0
      if (Power.eq.0) then
         TextToInteger=ichar(SubString(1:1))-48
      else
         Do 10 i=1,Power
10       TextToInteger=TextToInteger+
     1   ((ichar(SubString(i:i))-48)*10**(Power))
      endif
      return
      end

      Function IntegerToText(Number)
      Integer Number,Position,Power,FixNumber
      Character*(*) IntegerToText
```

```
      FixNumber=Number
      Position=1
      Power=3
      j=len(Integertotext)
      Do 5 i=1,j
5     IntegerToText(i:i)=char(32)

10    if (FixNumber-(10**(Power-1))) 20,15,15
15       i=Number/(10**(Power-1))
         IntegerToText(Position:Position)=char(i+48)
         Number=Number-i*10**(Power-1)
         Position=Position+1
20    Power=Power-1
      if (Power.ge.1) goto 10
      Number=FixNumber
      return
      end

      Function Insert(SubString,Start,Finish,String)
      Character*(*) SubString,String,Insert
      Integer Start,Finish,SubLength,StrLength

      j=len(insert)
      Do 10 i=1,j
10    Insert(i:i)=char(32)
      if (Start.gt.1) Insert(1:Start-1)=String(1:Start-1)
      SubLength=Length(SubString)
      StrLength=Length(String)
      Insert(Start:Start+SubLength-1)=SubString(1:SubLength)
      if (Finish+1.gt.StrLength) return
      Insert(Start+SubLength:Start+StrLength+SubLength-Finish-1)=
     1    String(Finish+1:StrLength)
      return
      end

      Function Locate(Direction,Start,ascii,String)
      Integer Start,Finish,ascii,Direction
      Character*(*) String
      Finish=1
      if (Direction.eq.1) Finish=Length(String)
      Do 10 i=Start,Finish,Direction
         if (ichar(String(i:i)).eq.ascii) then
         Locate=i
         return
         endif
10    Continue
      Locate=0
      return
      end
```

```
      Function Nelem(Array)
      Character*(*) Array(500)
      Do 10 i=500,1,-1
         j=length(Array(i))
         if (j.ne.0) then
         Nelem=i
         return
         endif
10    Continue
      end

      Function Uppercase(String)
      Character*(*) String,Uppercase
      Integer ascii,Finish
      Finish=Length(String)
      Do 10 i=1,Finish
         ascii=ichar(String(i:i))
         if (ascii.ge.97.and.ascii.le.122) String(i:i)=char(ascii-32)
10    Continue
      Uppercase=String
      return
      end

      Function Length(String)
      Character*(*) String
      Integer Length,j,ascii
      Length=len(String)
      j=Length
      Do 10 i=j,1,-1
         ascii=ichar(String(i:i))
         if (ascii.ne.32.and.ascii.ne.0) return
         Length=Length-1
10    Continue
      Length=0
      return
      end


      BlockData PHOENICS
      Character*50 Statements(500)
      Common /Commands/ Statements

C     The following statements, 1 to 74, are deemed to be the most popular

      Data (Statements(i),i=1,74) /' TALK',' STOP',' RUN',
     1' VDU',
     1'} INTEGER','} REAL',' TEXT','8 GRDPWR','_ SOLUTN','^ SOLVE',
     1' STORE',' TERMS',' CONPOR',' INIT','7 PATCH',
     1' RESTRT','6 COVAL',' RELAX',' OUTPUT',' PLOT',' STEADY',
     1' CARTES',' ONEPHS',' XCYCLE',' USEGRD',' USEGRX',
     1' ECHO',' AUTOPS',' NOWIPE',' SAVE',' LSTEP',
     1'9*NX','9*NY','9*NZ',' FSWEEP',' LSWEEP',' IXMON',
```

```
1' IYMON',
1' IZMON',' NPRMNT',' NPRMON',' TSTSWP',' IPLTF',
1' IPLTL',
1' NPRINT',' NXPRIN',' NYPRIN',' NZPRIN',' LITER',
1' TFIRST',' TLAST','9*XULAST','9*YVLAST','9*ZWLAST',
1' RINNER',' DIFCUT',' HUNIT','~*ENUL','~*ENUT','~*RHO1',
1'8 TFRAC','8 XFRAC','8 YFRAC','8 ZFRAC',' PRNDTL',' PRT',
1' FIINIT',' RESREF',' ENDIT',' VARMAX',' VARMIN',
1'9 NAME',' NAMFI',' NAMGRD'/

   Data (Statements(i),i=75,147) /' CLEAR',' LOAD',' SATRUN',
1' DOMAIN',' FIXDOM',' MAGIC',' READCO',' SEEPTS',
1' SETLIN',' SETPT',' VIEW',' RADIAT',' TURMOD',
1' PARAB',' BFC',' LIJ',' LIK',' LJK',' NONORT',
1' RSTGEO',' SAVGEO',' SLIDE',' SLIDH',' SLIDL',' SLIDN',
1' SLIDS',' SLIDW',' UGEOM',' UUP',' VUP',' WUP',
1' ADDDIF',' BLOCKZ',' DONACC',' EQDVDP',' GALA',
1' NEWRH1',
1' NEWENL',' NEWENT',' NEWRH2',' UCORR',' UDIFF',
1' UDIFNE',
1' UCONNE',' UCONV',' UCORCO',' USOLVE',' USOURC',
1' INIADD',
1' PICKUP',' DARCY',' DUDX',' DUDY',' DUDZ',
1' DVDX',' DVDY',' DVDZ',' DWDX',' DWDY',' DWDZ',
1' GENK',
1' LSG1',' LSG2',' LSG3',' LSG4',' LSG5',' LSG6',
1' LSG7',
1' LSG8',' LSG9',' LSG10',' DISTIL',' NULLPR'/

   Data (Statements(i),i=148,228) /' INIFLD',' SUBWGR',
1' XZPR',
1' YZPR',' LIBREF',' IMON',' JMON',' KMON',' LITXC',
1' LITYC',' LITZC',' MSWP',' NCRT',' DEN1',' DEN2',
1' EPOR',' HPOR',' IMB1',' IMB2',' INTFRC',' INTMDT',
1' LEN1',' LEN2',' NPOR',' PCOR',' TEMP1',' TEMP2',
1' VISL',' VIST',' VPOR',' ICHR',
1' IURINI',' IURVAL',' KELIN',' IPARAB',
1' ISWC1',' ISWR1',' ISWR2',' ITHC1',' LITC',' LITFLX',
1' LITYHD',' ISG1',' ISG2',' ISG3',' ISG4',
1' ISG5',' ISG6',' ISG7',' ISG8',' ISG9',' ISG10',
1' ISG11',' ISG12',' ISG13',' ISG14',' ISG15',' ISG16',
1' ISG17',' ISG18',' IZW1',' IURPRN',
1' IPROF',' ISTPRF',' ISTPRL',' ISWPRF',
1' ISWPRL',' ITABL',' IXPRL',' IXPRF',' IYPRF',
1' IYPRL',' IZPRF',' IZPRL',' NCOLCO',' NCOLPF',' NPLT',
1' NROWCO',' NTPRIN',' NTZPRF',' NUMCLS'/

   Data (Statements(i),i=229,314)/' LG',' IG',' AZXU',
1' AZYV',' AZDZ',' ZWADD',' TMP2',' TMP2A',
1' FIXCOR',' RELXC',' RELYC',' RELZC',
1' RLOLIM',' RUPLIM',' U1AD',' U2AD',' V1AD',
1' V2AD',' W1AD',' W2AD',' ZDIFAC',' DRH1DP',' DRH2DP',
```

```
1' EL1',' EL1A',' EL1B',' EL1C',' EL2',' EL2A',' EL2B',
1' EL2C',' ENULA',' ENULB',' ENULC',
1' ENUTA',' ENUTB',' ENUTC',' PRESSO',' PHNH1A',
1' PHNH1B',
1' PHNH1C',' PRLC1A',' PRLC1B',' PRLC1C',' PRLC2A',
1' PRLC2B',' PRLC2C',' PRLC3C',' PRLC4A',' PRLC4B',
1' PRLC4C',' PRLH1A',' PRLH1B',' PRLH1C',' RHO1A',
1' RHO1B',' RHO1C',' RHO2',' RHO2A',' RHO2B',' RHO2C',
1' TEMPO',' TMP1',' TMP1A',' TMP1B',' TMP1C',
1' TMP2B',' TMP2C',' CFIPS',' CFIPA',' CFIPB',
1' CFIPC',' CFIPD',' CINH1A',' CINH1B',' CINH1C',
1' CINH2A',
1' CINH2B',' CINH2C',' CMDOT',' CMDTA',' CMDTB',' CMDTC',
1' CMDTD',' HEATBL',' AZPH'/

   Data (Statements(i),i=315,377) /' PBAR',' CONMDT',
1' OVRRLX',
1' AZW1',' BZW1',' CZW1',' DZW1',' RSG1',' RSG2',
1' RSG3',
1' RSG4',' RSG5',' RSG6',' RSG7',' RSG8',' RSG9',
1' RSG10',
1' RSG11',' RSG12',' RSG13',' RSG14',' RSG15',' RSG16',
1' RSG17',' RSG18',' RSG19',' RSG20',' RSG21',' RSG22',
1' RSG23',' RSG24',' RSG25',' RSG26',' RSG27',' RSG28',
1' RSG29',' RSG30',' DSTTOL',' ABSIZ',' ORSIZ',
1' PHINT',' CINT',' RG',' EX',' CG',' CSG1',
1' CSG2',' CSG3',' CSG4',' CSG5',' CSG6',' CSG7',
1' CSG8',' CSG9',' CSG10',' NAMSAT',
1' UCRT',' U2CR',' VCRT',' V2CR',' WCRT',' W2CR',
1' SKIP'/

   end
```

APPENDIX D

FORTRAN mathematical parsing code, EVALUATE.FOR

```
      Program Evaluate

      Integer*4 DP,Start,Finish,llength
      Real TextToNumber,rvalue
      Character*80 Expression,uppercase,Values,String,Calculate,
     1 Insert,IntegerToText
      Character*10 Forchar(0:2)
      Character*1 RP,LP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
     1 COMMA,POINT
      Character*4 CLOG,CLOGE,CEXP

      Common /Signs/ LP,RP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
     1 COMMA,POINT,CLOG,CLOGE,CEXP
      Common /Integer/ DP

      Open(Unit=10,File='$$LEOINP.DAT',Status='UNKNOWN')
      Open(Unit=11,File='$$LEOOUT.DAT',Status='UNKNOWN')


      Do 1 i=0,2
         k=len(Forchar(i))
         Do 2 j=1,k
2           Forchar(i)(j:j)=char(32)
1     continue

      Forchar(0)(1:4)='(A1,'
      Forchar(2)(1:4)=',A1)'

      read(10,*) Expression,Values

      Expression=uppercase(Expression)
      Values=uppercase(Values)
      Finish=locate(1,1,COMMA,Values)-1
      String=Values(1:Finish)
      rvalue=int(TextToNumber(String))
      DP=int(rvalue)
      Call SortExpression(Expression,Values)
10    llength=length(Expression)
      Start=locate(-1,llength,LP,Expression)
      if (Start.eq.0) then
         Expression=Calculate(Expression)
      else
         Finish=locate(1,Start,RP,Expression)
         String=Expression(Start+1:Finish-1)
         String=Calculate(String)
         Expression=Insert(Start,Finish,String,Expression)
         goto 10
      endif

      llength=length(Expression)
      String=IntegerToText(llength)
      i=length(String)
```

```
      Forchar(1)(1:i)=String(1:i)
      Forchar(1)(i+1:i+2)='A1'
      write(11,Forchar) char(39),(Expression(i:i),i=1,llength),
     1                  char(39)
      close(10)
      close(11)

      end

C     *************** Subroutines and Functions ************

      BlockData Symbols
      Character*1 LP,RP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
     1  COMMA,POINT
      Character*4 CLOG,CLOGE,CEXP
      Common /Signs/ LP,RP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
     1  COMMA,POINT,CLOG,CLOGE,CEXP
      Data LP,RP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,COMMA,POINT,
     1 CLOG,CLOGE,CEXP
     1  /'(',')','+','-','*','/','^',',','.','LOG','LOGE','EXP'/
      end

      Function IntegerToText(Number)
      Integer Number,Position,Power,FixNumber
      Character*(*) IntegerToText
      FixNumber=Number
      Position=1
      Power=3
      j=len(IntegerToText)
      Do 5 i=1,j
5     IntegerToText(i:i)=char(32)
10    if (FixNumber-(10**(Power-1))) 20,15,15
15       i=Number/(10**(Power-1))
         IntegerToText(Position:Position)=char(i+48)
         Number=Number-i*10**(Power-1)
         Position=Position+1
20    Power=Power-1
      if (Power.ge.1) goto 10
      Number=FixNumber
      return
      end

      Subroutine SortExpression(String,Values)
      Integer*4 ascii,VStart,VFinish,DP,llength,SubLength,Position

      Character String*(*),Values*(*),Substring*80,Insert*80
      Character*1 RP,LP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
     1  COMMA,POINT
      Character*4 CLOG,CLOGE,CEXP

      Common /Signs/ LP,RP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
     1  COMMA,POINT,CLOG,CLOGE,CEXP
```

```
      Common /Integer/ DP

      i=1
10    llength=length(String)
15    if (i.gt.llength) then
        return
      endif
      ascii=ichar(String(i:i))
      if (ascii.ge.65.and.ascii.le.90) then
        if (ascii.eq.69.and.
     1      (ichar(String(i+1:i+1)).eq.43.or.
     1       ichar(String(i+1:i+1)).eq.45).and.
     1      (ichar(String(i-1:i-1)).ge.48.and.
     1       ichar(String(i-1:i-1)).le.57)) then
          i=i+1
          goto 15
          elseif (String(i:i+3).eq.CLOGE) then
                i=i+4
                goto 15
                elseif (String(i:i+2).eq.CLOG.or.
     1                  String(i:i+2).eq.CEXP) then
                        i=i+3
                        goto 15
        endif
        j=i+1
20      jascii=ichar(String(j:j))
        if (jascii.eq.40.or.jascii.eq.41.or.jascii.eq.42.
     1      or.jascii.eq.43.or.jascii.eq.45.or.jascii.eq.
     1      47.or.jascii.eq.94.or.j.ge.llength) then
          Substring=String(i:j-1)
          if (j.eq.llength.and.jascii.ne.40.and.jascii.ne.41)
     1      then
            Substring=String(i:j)
            j=j+1        ! Increment j by 1 for correct insertion
          endif
          SubLength=length(SubString)
          Position=0
30        VStart=locate(1,Position+1,Substring,Values)
          if (Values(VStart-1:VStart-1).ne.COMMA.and.
     1      Values(VStart+1:VStart+1).ne.COMMA) then
            Position=VStart
            goto 30
          endif
          VStart=VStart+SubLength+1
          VFinish=locate(1,VStart,COMMA,Values)-1
          if (VFinish.lt.0) VFinish=length(Values)
          SubString=Values(VStart:VFinish)
          String=Insert(i,j-1,SubString,String)
          i=i+length(SubString)
          goto 10
        else
          j=j+1
```

```
            goto 20
          endif
        endif
        i=i+1
        goto 15
        end

        Function Calculate(String)
        Integer*4 DP,Start,Finish,Position
        real number1,number2
        Character String*(*),Calculate*(*)
        Character*80 NumberToText,Insert,SubString
        Character*4 CLOG,CLOGE,CEXP
        Character*1 RP,LP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
    1   COMMA,POINT

        Common /Signs/ LP,RP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
    1   COMMA,POINT,CLOG,CLOGE,CEXP
        Common /Integer/ DP

10      Position=locate(1,1,CLOGE,String)
        if (Position.ne.0) then
          Call SingleNumbers(Number1,Position+4,String,Finish)
          SubString=NumberToText(LOG(Number1))
          String=Insert(Position,Finish,SubString,String)
          goto 10
        else
20      Position=locate(1,1,CLOG,String)
        if (Position.ne.0) then
          Call SingleNumbers(Number1,Position+3,String,Finish)
          SubString=NumberToText(LOG10(Number1))
          String=Insert(Position,Finish,SubString,String)
          goto 20
        else
30      Position=locate(1,1,CEXP,String)
        if (Position.ne.0) then
          Call SingleNumbers(Number1,Position+3,String,Finish)
          SubString=NumberToText(EXP(Number1))
          String=Insert(Position,Finish,SubString,String)
          goto 30
        else
40      Position=locate(1,1,POWER,String)
        if (Position.ne.0) then
          Call DoubleNumbers(Number1,Number2,Position,String,
    1                        Start,Finish)
          SubString=NumberToText(Number1**Number2)
          String=Insert(Start,Finish,SubString,String)
          goto 40
        else
50    Position=locate(1,1,MULTIPLY,String)
        if (Position.ne.0) then
          Call DoubleNumbers(Number1,Number2,Position,String,
```

```
     1                    Start,Finish)
                SubString=NumberToText(Number1*Number2)
                String=Insert(Start,Finish,SubString,String)
                goto 50
            else
60          Position=locate(1,1,DIVIDE,String)
            if (Position.ne.0) then
                Call DoubleNumbers(Number1,Number2,Position,String,
     1                    Start,Finish)
                SubString=NumberToText(Number1/Number2)
                String=Insert(Start,Finish,SubString,String)
                goto 60
            endif
            endif
            endif
            endif
            endif
            endif
c   Perform the addition and subtraction
70          Position=0
80          Position=locate(1,Position+1,PLUS,String)
            if (Position.ne.0) then
                if (String(Position-1:Position-1).eq.'E') goto 80
                Call DoubleNumbers(Number1,Number2,Position,String,
     1                    Start,Finish)
                SubString=NumberToText(Number1+Number2)
                String=Insert(Start,Finish,SubString,String)
                goto 80
            else
90          Position=0
100         Position=locate(1,Position+1,MINUS,String)
            if (Position.ne.0) then
                if ((Position.gt.1.and.String(Position-1:Position-1).
     1          eq.'E').or.Position.eq.1) goto 100
                Call DoubleNumbers(Number1,Number2,Position,String,
     1                    Start,Finish)
                SubString=NumberToText(Number1-Number2)
                String=Insert(Start,Finish,SubString,String)
                goto 100
            endif
            endif
            Calculate=String
            return
            end


            Function uppercase(String)
            Character String*(*),uppercase*(*)
            Integer ascii,llength
            llength=length(String)
            Do 10 j=1,llength
                ascii=ichar(String(j:j))
```

```
            if (ascii.ge.97.and.ascii.le.122) then
                String(j:j)=char(ascii-32)
            else
                String(j:j)=char(ascii)
            endif
10          continue
            uppercase=String
            return
            end


            Function length(String)
            Character*(*) String
            length=len(String)
            llength=length
            Do 10 j=llength,1,-1
               if (ichar(String(j:j)).ne.32) return
               length=length-1
10          continue
            length=0
            return
            end

            Function locate(direction,start,SubString,String)
            Integer*4 direction,start,finish,SubLength
            Character SubString*(*),String*(*)
            SubLength=length(SubString)
            finish=1
            if (direction.eq.1) finish=length(String)
            Do 10 i=start,finish,direction
               if (String(i:i+SubLength-1).eq.Substring) then
                  locate=i
                  return
               endif
10          continue
            locate=0
            return
            end

            Function Insert(istart,ifinish,SubString,String)
            Integer*4 istart,ifinish,llength,tlength
            Character SubString*(*),String*(*),Insert*(*)
            j=len(Insert)
            Do 10 i=1,j
10          Insert(i:i)=char(32)
            if (istart.gt.1) Insert(1:istart-1)=String(1:istart-1)
            llength=length(SubString)
            tlength=istart-1+llength
            Insert(istart:istart+llength-1)=SubString(1:llength)
            llength=length(String)
            if (ifinish+1.gt.llength) return
            Insert(tlength+1:tlength+llength-ifinish+1)=
```

```
     1        String(ifinish+1:llength)
              return
              end


              Function LocateMaths(Direction,Start,String)
              Integer*4 Direction,Start,Finish

              Character*(*) String
              if (Direction-0) 10,10,20
10               Finish=1
              goto 30
20               Finish=length(String)
30            Do 40 i=Start,Finish,Direction
                 ascii=ichar(String(i:i))
                 if ((ascii.eq.42.or.ascii.eq.43.or.ascii.eq.45.
     1              or.ascii.eq.47)) then
                    if (i.eq.start.and.ichar(String(i:i)).eq.45) goto 40
                    if (i.gt.1.and.ichar(String(i-1:i-1)).eq.69) goto 40
                    if (Direction.eq.1) then
                            LocateMaths=i-1
                    else
                            if (i.eq.1) then
                               LocateMaths=i
                               return
                            endif
                            LocateMaths=i+1
                    endif
                    return
                 endif
40            continue
              LocateMaths=Finish
              return
              end

              Function TextToNumber(String)
              Integer*4 llength,DP,EPos,Ipower,Start,PSign,Sign
              Character String*(*),E*1
              Character*1 RP,LP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
     1        COMMA,POINT
              Character*4 CLOG,CLOGE,CEXP
              Real TextToNumber
              Common /Signs/ LP,RP,PLUS,MINUS,MULTIPLY,DIVIDE,POWER,
     1        COMMA,POINT,CLOG,CLOGE,CEXP
              Common /Integer/ DP
              E='E'
              Ipower=0
              Sign=1
              TextToNumber=0
              llength=length(String)
              if (String(1:1).eq.MINUS) then
                    Sign=-1
```

```
              String=String(2:llength)
              llength=llength-1
          endif
          EPos=locate(-1,llength,E,String)
          if (EPos.gt.0) then
             j=llength-EPos-1
             Start=EPos+1
             PSign=1
             if (String(EPos+1:EPos+1).eq.PLUS.or.
     1           String(EPos+1:EPos+1).eq.MINUS) then
                j=llength-EPos-2
                Start=EPos+2
             endif
             if (String(EPos+1:EPos+1).eq.MINUS) PSign=-1
                Do 5 i=Start,llength
                   Ipower=Ipower+((ichar(String(i:i))-48)*(10**j))
                   j=j-1
5               continue
             Ipower=PSign*Ipower
             String=String(1:Epos-1)
             llength=length(String)
          endif
          if (locate(1,1,POINT,String).eq.0) then
             j=llength-1
          else
             j=locate(1,1,POINT,String)-2
          endif
             Do 20 i=1,llength
                if (ichar(String(i:i)).eq.46) goto 20
                TextToNumber=TextToNumber+
     1                       ((float(ichar(String(i:i)))-48.0)
     1        j=j-1          *(10.0**float(j)))
10        continuej=j-1
20           continue
          TextToNumber=Sign*TextToNumber*(10.0**float(Ipower))
          return
          end

          Function NumberToText(Number)
          Character*80 NumberToText,DummyScreen
          Real Number,Inumber
          Integer DP,Sign,Power,Start
          Common /Integer/ DP
          Start=1
          Do 5 i=1,80
5         NumberToText=char(32)
          if (number-0.0) 7,8,9
7         NumberToText(1:1)='-'
          Start=2
          Number=abs(Number)
          goto 9
8         NumberToText='0'
```

```
        return
9       if (number-1.0) 10,20,30
10      Sign=1
        goto 35
20      NumberToText='1'
        return
30      Sign=-1

C   Get the number into a standard form

35      Power=0
40      Inumber=number*(10.0**(float(Sign)*float(Power)))
        write(DummyScreen,110) Inumber      ! DO NOT remove this line
        if (Inumber.ge.1.0.and.Inumber.lt.10.0) goto 50
        Power=Power+1
        goto 40
50      Inumber=(float(int((Inumber*(10.0**float(DP+1)))+0.5)))/
    1         (10.0**float(DP+1))
c       jnumber=int(inumber*(1.0+1.0*10.0**(-1*(DP))))
        write(DummyScreen,110) Inumber      ! DO NOT remove this line
        NumberToText(Start:Start)=char(int(inumber)+48)
        NumberToText(Start+1:Start+1)=char(46)
        Inumber=(Inumber-float(int(Inumber))+0.5*10.0**(-1*DP))*10.0
        Do 60 i=1,DP-1
          write(DummyScreen,110) Inumber  ! DO NOT remove this line
          NumberToText(i+Start+1:i+Start+1)=char((int(Inumber))+48)
          Inumber=(Inumber-float(int(Inumber))+
    1                 0.5*10.0**(-1*DP))*10.0
60      continue
        llength=length(NumberToText)
65      if (ichar(NumberToText(llength:llength)).ne.48) goto 70
        NumberToText(llength:llength)=char(32)
        llength=llength-1
        goto 65
70      if (Power.eq.0) then
          if (ichar(NumberToText(llength:llength)).eq.46)
    1     NumberToText(llength:llength)=char(32)
          return
        else
          NumberToText(llength+1:llength+1)=char(69)
          if (0-Sign) 80,80,90
80        NumberToText(llength+2:llength+2)=char(45)
          goto 100
90        NumberToText(llength+2:llength+2)=char(43)
100       if (Power.lt.10) then
            NumberToText(llength+3:llength+3)=char(Power+48)
          elseif (Power.lt.100) then
              NumberToText(llength+3:llength+3) =
    1         char(int(float(Power)/10.0)+48)
              Power=Power-(int(float(Power)/10.0))*10
              NumberToText(llength+4:llength+4)=
    1         char(Power+48)
```

```
          endif
        endif

        return
110     Format(E20.15)
        end

        Subroutine DoubleNumbers(Number1,Number2,Position,String,
     1                  Start1,Finish2)
        Real Number1,Number2,TextToNumber
        Integer*4 Position,Start1,Finish2
        Character*(*) String
        Start1=LocateMaths(-1,Position-1,String)
        Finish2=LocateMaths(1,Position+1,String)
        Number1=TextToNumber(String(Start1:Position-1))
        Number2=TextToNumber(String(Position+1:Finish2))
        return
        end

        Subroutine SingleNumbers(Number,Start,String,Finish)
        Real Number,TextToNumber
        Integer*4 Start,Finish
        Character*(*) String
        Finish=LocateMaths(1,Start,String)
        Number=TextToNumber(String(Start:Finish))
        return
        end
```

APPENDIX E

LISP KBFE Objects and Rulebases

OBJECTS

```
(set-object TargetUserModel
            :Type 'text
            :AllowedValues '(novice experienced advanced)
            :DefaultValue 'novice
            :Prompt "What type of PHOENICS user to you consider yourself to be ?")

(set-object UserModel
            :Type 'text
            :AllowedValues '(novice experienced advanced)
            :DefaultValue 'novice
            :Prompt "What kind of KBFE user do you consider yourself to be ?")

(set-object fact)

(set-object boundary-names
            :type 'list
            :Prompt 'never)

(set-object trace
            :Type 'text
            :Value 'off)

(set-object inlet-flow-area
            :Type 'real
            :Rulebase t)

(set-object number-of-bindings
          :Type 'integer)

(set-object x-min
            :type 'real)

(set-object x-max
            :type 'real)

(set-object y-min
            :type 'real)

(set-object y-max
            :type 'real)

(set-object z-min
            :type 'real)

(set-object z-max
            :type 'real)

(set-object minimum-region-size
```

```
            :type 'real
            :ComputeValue '(set-minimum-region-size))

(set-object porosity-definition
        :Type 'text
        :Preface '(For the blockages you can define a default porosity -
                0 for a solid - 1 for no obstruction or greater
                than 1 for simulating expanded cells. The default
                which you can predefine will be applied to all
                obstructions. Alternatively you can individually
                specify obstruction porosities.)
        :DefaultValue 'constant-0.0
        :AllowedValues '(constant-0.0 constant-predefined
                        individually-defined)
        :Prompt "Porosity definition :"
        :Rulebase t)

(set-object porosity
        :Type 'real
        :Status 'volatile
        :DefaultValue 0.0
        :AllowedValues '(>= 0.0 <= 10.0)
        :Prompt "Porosity value")

(set-object x-grid
        :Type 'list)

(set-object y-grid
        :Type 'list)

(set-object z-grid
        :Type 'list)

(set-object dimensional-units
        :Type 'text
        :DefaultValue 'mm
        :AllowedValues '(m mm)
        :Preface '(What are the coordinate units)
        :Prompt "What are the dimensional units ?")

(set-object conversion-factor
        :Type 'real
        :RuleBase t)

(set-object number-of-dimensions
        :Type 'integer
        :DefaultValue 2
          :FixedValue 2
        :AllowedValues '(>= 1 <= 3)
        :Preface '(The geometry can be either 2 or 3 dimensional.)
        :Prompt "Number of dimensions")
```

```
(set-object axis-1
        :Type 'Text
        :AllowedValues '(unused x circumferential))


(set-object axis-2
        :Type 'Text
        :AllowedValues '(unused y radial))


(set-object axis-3
        :Type 'Text
        :AllowedValues '(unused z axial))

(set-object store-variables
        :Type 'list)

(set-object delta
        :Type 'real
          :Prompt "Enter the minimum cell size"
        :RuleBase t)

(set-object boundary-layer-thickness
        :Type 'real
        :ComputeValue '(get-boundary-layer-thickness))

(set-object slab-wise-variables
        :Type 'list)

(set-object whole-field-variables
        :Type 'list
        :Value '(p1))

(set-object target-file
        :Description '(Target PHOENICS data file)
        :Type 'string
        :Preface '(Enter the target ^PHOENICS data file.  Please include
                the file extension - ie target.file)
        :DefaultValue "Q1.DAT"
        :Prompt "PHOENICS target data file")

(set-object laminar-prndtl
        :Description '(Laminar Prandtl number)
        :Type 'real
          :Units "Dimensionless"
        :Prompt "Enter the Prandtl number for the fluid")

(set-object viscosity
        :Type 'real
          :Units "m^2 / s"
        :Prompt "Enter the kinematic viscosity")
```

```
(set-object viscosity-thermal-dependence
        :Type 'text
        :AllowedValues '(required not-required)
        :DefaultValue 'not-required
        :Preface '(If you wish to simulate the change of viscosity within
                the domain depending upon the calculated local
                temperatures then enter ^ required at the prompt.)
        :Prompt "Viscosity thermal dependence required or not-required ?"
        :RuleBase t)

(set-object viscosity-equation
        :Type 'string
        :AllowedValues '("A+BT" "A+BT+BT**2")
        :DefaultValue "A+BT"
        :Prompt "Enter the viscosity equation which is appropriate")

(set-object enula
        :Type 'real
        :DefaultValue 0.0
        :Preface '((Viscosity equation - (1)) viscosity-equation)
        :Prompt "Enter coefficient A of the viscosity equation")

(set-object enulb
        :Type 'real
        :DefaultValue 1.0
        :Preface '((Viscosity equation - (1)) viscosity-equation)
        :Prompt "Enter coefficient B of the viscosity equation")

(set-object enulc
        :Type 'real
        :DefaultValue 0.0
        :Preface '((Viscosity equation - (1)) viscosity-equation)
        :Prompt "Enter coefficient C of the viscosity equation")

(set-object tmp1-equation
        :Type 'string
        :AllowedValues '(constant "A+BH")
        :DefaultValue "A+BH"
        :Preface '(Temperature can be expressed a a function of enthalpy.
                Enter the equation which expresses the relationship of
                temperature as a function of enthalpy.)
        :Prompt "T = f(H)"
        :RuleBase t)

(set-object tmp1a
        :Type '-  '
        :DefaultValue 0.0
        :Preface '((T--perature/enthalpy relationship - (1))
                tmp1-equation)
        :Prompt "Enter coefficient A in the temperature/enthalpy relationship")
```

```
(set-object tmp1b
        :Type 'real
        :DefaultValue 1.0
        :Preface '((Temperature/enthalpy relationship - (1))
                tmp1-equation)
        :Prompt "Enter coefficient B in the temperature/enthalpy relationship")



(set-object density
        :Type 'real
        :DefaultValue 1.0
          :Units "kg / m^3"
        :Prompt "Enter the density")

(set-object density-thermal-dependence
        :Type 'text
        :AllowedValues '(not-required enthalpy temperature)
        :DefaultValue 'not-required
        :Preface '(If you wish to simulate the change of density within
                the domain depending upon the localised thermal
                conditions then enter the appropriate value.)
        :Prompt "Density thermal dependence"
        :Rulebase t)

(set-object density-equation
        :Type 'string
        :AllowedValues '("A+BT" "1/(A+BH)" "C+A(P)**B" "A+BT" "B(P)/T")
        :DefaultValue "A+BT"
        :Prompt "Enter the density equation which is appropriate")

(set-object rho1a
        :Type 'real
        :DefaultValue 0.0
        :Preface '((Density equation - (1)) density-equation)
        :Prompt "Enter coefficient A of the density equation")

(set-object rho1b
        :Type 'real
        :DefaultValue 1.0
        :Preface '((Density equation - (1)) density-equation)
        :Prompt "Enter coefficient B of the density equation")

(set-object rho1c
        :Type 'real
        :DefaultValue 0.0
        :Preface '((Density equation - (1)) density-equation)
        :Prompt "Enter coefficient C of the density equation")
```

```
(set-object presso
        :Type 'real
        :DefaultValue 0.0
        :Preface '(The datum static pressure is the pressure that needs to
                be added to the pheonics solution pressure field so
                that the pressure dependent physical quantities can be
                calculated.)
        :Prompt "Enter the datum static pressure")

(set-object coordinates
        :Type 'text
        :AllowedValues '(cylindrical cartesian)
        :DefaultValue 'cartesian
        :Preface '(^Phoenics essentially uses two types of coordinate
                systems - cartesian and cylindrical. For two
                dimensional configurations which this system can
                initially develop the ^XY plane will be utilised. This
                will be automatically translated into the respective
                ^XY or ^YZ planes which phoenics requires depending upon
                your choice of either cartesian of cylindrical
                coordinates.)
        :Prompt "Are the coordinates cartesian or cylindrical ?")

(set-object analysis-title
        :Type 'string
        :Preface '(The analysis title cannot be more than 40
                characters long. The main purpose of this is
                to be able to identify the analysis.)
          :Help '((Novice (This is the novice help))
                (Experienced (This is the experienced help))
                (Advanced (This is the advanced help)))
        :Prompt "What is the analysis title ?")


(set-object thermal-requirements
        :Type 'text
        :AllowedValues '(thermal isothermal)
        :DefaultValue 'Isothermal
        :Prompt "Is the analysis thermal or isothermal ?")

(set-object number-of-inlets
        :Type 'integer
        :AllowedValues '(> 0)
        :DefaultValue 1
        :Prompt "How many inlets are within the domain ?")

(set-object number-of-outlets
        :Type 'integer
        :AllowedValues '(>= 1)
        :DefaultValue 1
        :Prompt "How many outlets are within the domain ?")
```

```
(set-object number-of-obstructions
        :Type 'integer
        :AllowedValues '(>= 0)
        :DefaultValue 0
        :Prompt "How many obstructions are within the domain ?")

(set-object flow-regime
        :Type 'text
        :AllowedValues '(Laminar Turbulent)
        :DefaultValue 'laminar
        :Prompt "Is the flow to be laminar or turbulent ?")

(set-object fluid-compressibility
        :Type 'text
        :AllowedValues '(compressible incompressible)
        :DefaultValue 'incompressible
          :Fixedvalue 'incompressible
        :Preface '(Gases such as air act as incompressible fluids for
                mach numbers less than 0.3.)
        :Prompt "Is the fluid compressible or incompressible ?")

(set-object aspect-ratio
        :Type 'integer
          :AllowedValues '(>= 1 <= 10)
          :Units "Dimensionless"
        :DefaultValue 5
          :Prompt "Enter the maximum allowed aspect ratio")
```

RULEBASES

Rulebase: BC-RB
Network: ((1 ())
(2 ((5 ((6 ()) (7 ()) (8 ()) (14 ())))
(12 ((13 ()) (14 ()) (15 ())))
(16 ())
(17 ())
(18 ())
(19 ())
(20 ())
(21 ())
(22 ())
(23 ())
(24 ())
(25 ())
(26 ())))
(3 ((12 ((13 ()) (14 ()) (15 ())))))
(4 ((5 ((6 ()) (7 ()) (8 ()) (14 ())))))
(9 ((5 ((6 ()) (7 ()) (8 ()) (14 ())))
(12 ((13 ()) (14 ()) (15 ())))
(16 ())
(17 ())
(18 ())
(19 ())
(20 ())
(21 ())
(22 ())
(23 ())
(24 ())
(25 ())
(26 ())))
(10 ((16 ())))
(11 ((17 ()) (18 ()) (19 ()) (20 ()) (21 ()) (22 ()) (23 ()))))
Rules:

```
(rule-1 (if (dependent-variables are uninstantiated))
      (then (instantiate dependent-variables)))

(rule-2 (for all dependent-variables
           (if ((u1 v1 w1) includes |$value|)
              (boundary name for |$type| |$identity| |$nodes| is |$name|)
              (|$type| is inlet)
              (|$value| condition is |$condition|)
              (|$value| value is |$quantity|))
           (then (assert |$value| at |$type| boundary |$name| is |$condition| at
                    |$quantity|)))))

(rule-3 (if (dependent-variables includes ke)
           (boundary name for |$type| |$identity| |$nodes| is |$name|)
           (|$type| is inlet)
```

```
        (ke value is |$quantity|))
    (then (assert ke at inlet boundary |$name| is constant at |$quantity|))))

(rule-4 (if (dependent-variables includes ep)
        (boundary name for inlet |$identity| |$nodes| is |$name|)
        (ep value is |$quantity|))
    (then (assert ep at inlet boundary |$name| is constant at
            |$quantity|)))

(rule-5 (if (ke at inlet boundary |$name| is constant at |$ke|)
        (cardinal for surface |$nodes| is |$cardinal|)
        (characteristic length for |$name| = |$gmixl|))
    (then (ep value is ((|$ke| |^| 1.5) * 0.1643 / (|$Gmixl| * 0.09)))))

(Rule-6 (if (coordinates are cartesian)
        ((north south) includes |$cardinal|))
    (then (characteristic length for |$name| = (abs ((xc_2) - (xc_1))))))

(rule-7 (if (coordinates are cylindrical)
        ((north south) includes |$cardinal|))
    (then (characteristic length for |$name| = (abs ((zc_2) - (zc_1))))))

(rule-8 (if (or ((coordinates are cartesian)
            ((west east) includes |$cardinal|))
          ((coordinates are cylindrical)
            ((low high) includes |$cardinal|))))
    (then (characteristic length for |$name| = (abs ((yc_2) - (yc_1))))))

(rule-9 (for all dependent-variables
        (if ((u1 v1 w1 p1 ep ke) excludes |$value|)
          (boundary name for |$type| |$identity| |$nodes| is |$name|)
          (|$type| is inlet)
          (|$value| condition is |$condition|)
          (|$value| value is |$quantity|))
        (then (assert |$value| at |$type| boundary |$name| is |$condition| at
            |$quantity|)))))

(rule-10 (if (dependent-variables includes p1)
        (boundary name for |$type| |$identity| |$nodes| is |$name|)
        (|$type| is outlet)
        (p1 value is |$quantity|))
    (then (assert p1 at |$type| boundary |$name| is constant at |$quantity|)))

(rule-11 (if (dependent-variables includes h1)
        (boundary name for |$type| |$identity| |$nodes| is |$name|)
        (|$type| is wall)
        (h1 condition is |$condition|)
        (h1 value is |$quantity|))
    (then (assert h1 at |$type| boundary |$name| is |$condition| at |$quantity|)))

(rule-12 (if (u1 for |$name| is |$u1-value|)
        (v1 for |$name| is |$v1-value|)
```

```
            (w1 for |$name| is |$w1-value|)
            (turbulence intensity for |$name| is |$ti|))
        (then (ke value is
                ((|$ti| / 6) *
                ((|$u1-value| |^| 2) + ($v1-value| |^| 2) + (|$w1-value| |^| 2))))))

 (rule-13 (if (|$velocity| at inlet boundary |$name| is constant is
            uninstantiated))
        (then (|$velocity| for |$name| is 0.0)))

 (Rule-14 (if (|$velocity| at inlet boundary |$name| is constant at
            |$velocity-value|))
        (then (|$velocity| for |$name| is |$velocity-value|)))

 (rule-15 (if (flow-regime is turbulent))
        (then (turbulence intensity for |$name| is
                (ask turbulence intensity at |$name|
                        (|$type| |$identity| nodes |$nodes|)
                        ((type real)
                         (allowedvalues (> 0 < 0.5))
                         (Defaultvalue 0.01)
                         (Help ((default (enter the turbulent intensity as a percentage of
the root mean squared velocity at |$name| - your entered value should be between 0 and
1 this is usually in the range of 1 percent - ie 0.01)))))))))

 (Rule-16 (if (|$type| is outlet))
        (then (p1 condition is constant)
            (p1 value is
                (ask outlet pressure at |$name|
                    ((type real)
                     (units "n / m^2")
                     (defaultvalue 0.0)
                     (Allowedvalues (>= 0.0))
                     (Help ((default (fluid velocity fields are driven by |pressure.|
|^Phoenics| requires that at least one |^outlet| pressure is |fixed.| This is usually at
atmospheric gauge |pressure.| Please enter the known value - if this is unknown then
accept the default |value.|)))))))))

 (Rule-17 (if ((u1 v1 w1 ep ke) includes |$value|))
        (then (|$value| condition is constant)))

 (rule-18 (if (|$type| is inlet))
        (then (h1 condition is
                (ask thermal condition at |$name|
                    (|$type| |$identity| nodes |$nodes|)
                    ((type text)
                     (defaultvalue isothermal)
                     (allowedvalues (isothermal constant-heat-flux)))))))

 (rule-19 (if (|$type| is inlet) (|$condition| is isothermal))
        (then (h1 value is
                (ask temperature at |$name|
```

```
                (|$type| |$identity| nodes |$nodes|)
                ((type real) (units "degrees celsius"))))))

(rule-20 (if (|$type| is inlet) (|$condition| is constant-heat-flux))
        (then (h1 value is
                (ask heat flux at |$name|
                    (|$type| |$identity| nodes |$nodes|)
                    ((type real)
                     (allowedvalues (> 0))
                     (units "w / m^2"))))))

(rule-21 (if (|$type| is wall))
        (then (h1 condition is
                (ask thermal condition at |$name|
                    (|$type| |$identity| nodes |$nodes|)
                    ((type text)
                     (allowedvalues
                      (isothermal adiabatic constant-heat-flux)))))))

(rule-22 (if (|$type| is wall) (|$condition| is isothermal))
        (then (h1 value is
                (ask temperature at |$name|
                    (|$type| |$identity| nodes |$nodes|)
                    ((type real) (units "degrees celsius"))))))

(rule-23 (if (|$type| is wall) (|$condition| is constant-heat-flux))
        (then (h1 value is
                (ask heat flux at |$name|
                    (|$type| |$identity| nodes |$nodes|)
                    ((type real) (units "w / m^2"))))))

(rule-24 (if (|$type| is inlet))
        (then (u1 value is
                (ask u1 velocity at |$name|
                    ((type real)
                     (units "m / s")
                     (help ((default (axis-1 velocity)))))))))

(rule-25 (if (|$type| is inlet))
        (then (v1 value is
                (ask v1 velocity at |$name|
                    ((type real)
                     (units "m / s")
                     (help ((default (axis-2 velocity)))))))))

(rule-26 (if (|$type| is inlet))
        (then (w1 value is
                (ask w1 velocity at |$name|
                    ((type real)
                     (units "m / s")
                     (help ((default (axis-3 velocity)))))))))
```

Rulebase: DEPENDENT-VARIABLES-RB
Network: ((1 ()) (2 ()) (3 ()) (4 ()) (5 ()) (6 ()) (7 ()) (8 ()))
Rules:

```
(rule-1  (if ((laminar turbulent) includes flow-regime))
         (then (dependent-variables includes p1)))

(rule-2  (if (axis-1 is-not unused))
         (then (dependent-variables includes u1)
               (slab-wise-variables includes u1)))

(rule-3  (if (axis-2 is-not unused))
         (then (dependent-variables includes v1)
               (slab-wise-variables includes v1)))

(rule-4  (if (axis-3 is-not unused))
         (then (dependent-variables includes w1)
               (slab-wise-variables includes w1)))

(rule-5  (if (thermal-requirements is thermal))
         (then (dependent-variables includes h1)
               (whole-field-variables includes h1)))

(rule-6  (if (or ((density-thermal-dependence is temperature))
                 ((density-thermal-dependence is enthalpy))
                 ((thermal-requirements is isothermal)
                  (fluid-compressibility is compressible))))
         (then (store-variables includes rho1)))

(rule-7  (if (viscosity-thermal-dependence is required))
         (then (store-variables includes enul)))

(rule-8  (if (flow-regime is turbulent))
         (then (dependent-variables includes ke)
               (dependent-variables includes ep)
               (store-variables includes enut)))
```

Rulebase: CONVERSION-FACTOR-RB
Network: ((1 ()) (2 ()))
Rules:

```
(rule-1  (if (dimensional-units are m)) (then (conversion-factor = 1.0)))
(rule-2  (if (dimensional-units are mm)) (then (conversion-factor = 0.001)))
```

Rulebase: DELTA-RB
Network: ((1 ((2 ()) (3 ()))))
Rules:

```
(rule-1  (if (recommended smallest cell size = |$delta|))
         (then (delta defaultvalue |$delta|)
               (delta allowedvalues
                     (>= |$delta| < (minimum-region-size / 2)))
               (delta preface
                     (the recommended minimum cell size has been
                        calculated as
```

```
                    (|$delta| / conversion-factor)
                    dimensional-units according to the geometry and
                    existing boundary |conditions.| You can accept
                    this default value by pressing return - or you
                    can enter a new |value.|))
            (Ask delta)))

 (rule-2  (if (minimum-region-size < (2 * boundary-layer-thickness)))
       (then (recommended smallest cell size =
            (0.1 * Minimum-region-size))))


 (rule-3  (if (minimum-region-size >= (2 * boundary-layer-thickness)))
       (then (recommended smallest cell size =
            (boundary-layer-thickness / 3.0))))

Rulebase: FLUID-RB
 Network: ((1 ())
       (2 ())
       (3 ())
       (4 ())
       (5 ())
       (6 ())
       (7 ())
       (8 ())
       (9 ())
       (10 ())
       (11 ())
       (12 ())
       (13 ())
       (14 ()))
  Rules:
 (rule-1  (if (thermal-requirements is thermal)
          (laminar-prndtl is uninstantiated))
       (then (ask laminar-prndtl)))

 (rule-2  (if (or ((thermal-requirements is isothermal)
               (viscosity is uninstantiated))
              ((thermal-requirements is thermal)
               (viscosity-thermal-dependence is-not required))))
       (then (ask viscosity)))

 (rule-3  (if (flow-regime is laminar)
          (thermal-requirements is thermal)
          (viscosity-thermal-dependence is required)
          (viscosity-equation is "a+bt"))
       (then (ask enula) (ask enulb) (ask tmp1-equation)))

 (rule-4  (if (flow-regime is laminar)
          (thermal-requirements is thermal)
          (viscosity-thermal-dependence is required)
          (viscosity-equation is "a+bt+ct**2"))
```

```
        (then (ask enula) (ask enulb) (ask enulc) (ask tmp1)))

(rule-5  (if (or ((fluid-compressibility is incompressible)
                  (thermal-requirements is isothermal)
                  (density is uninstantiated))
                 ((thermal-requirements is thermal)
                  (density-thermal-dependence is-not required))))
        (then (ask density)))

(rule-6  (if (thermal-requirements is isothermal)
             (fluid-compressibility is compressible))
        (then (density-equation is "c+a(p)**b")))


(rule-7  (if (thermal-requirements is thermal)
             (density-thermal-dependence is enthalpy))
        (then (density-equation allowedvalues ("a+bh" "1/a+bh"))))

(rule-8  (if (thermal-requirements is thermal)
             (density-thermal-dependence is temperature)
             (fluid-compressibility is compressible))
        (then (density-equation is "b(p)/t")))

(rule-9  (if (thermal-requirements is thermal)
             (density-thermal-dependence is temperature)
             (fluid-compressibility is incompressible))
        (then (density-equation is "a+bt")))

(rule-10 (if (thermal-requirements is thermal)
             (density-thermal-dependence is enthalpy)
             (density-equation is "a+bh"))
        (then (ask rho1a) (ask rho1b)))

(rule-11 (if (thermal-requirements is thermal)
             (density-thermal-dependence is enthalpy)
             (density-equation is "1/(a+bh)"))
        (then (ask rho1a) (ask rho1b)))

(rule-12 (if (thermal-requirements is isothermal)
             (fluid-compressibility is compressible)
             (density-equation is "c+a(p)**b"))
        (then (ask rho1a) (ask rho1b) (ask rho1c) (ask presso)))

(rule-13 (if (thermal-requirements is thermal)
             (density-thermal-dependence is temperature)
             (density-equation is "a+bt"))
        (then (ask rho1a) (ask rho1b) (ask tmp1-equation)))

(rule-14 (if (thermal-requirements is thermal)
             (density-thermal-dependence is temperature)
             (fluid-compressibility is compressible)
             (density-equation is "b(p)/t"))
```

```
        (then (ask rho1b) (ask presso) (ask tmp1-equation)))

Rulebase: G1-RB
 Network: ((1 ()) (2 ()) (3 ()))
  Rules:
 (rule-1  (if nothing)
        (then (->q1 message |^group| 1 run identifiers and other
                  preliminaries)))

 (rule-2  (if (analysis-title is instantiated))
        (then (->q1 |?[]| Text analysis-title)))

 (rule-3  (if nothing) (then (->screen group 1 complete)))

Rulebase: G2-RB
 Network: ((1 ()) (2 ()))
  Rules:
 (rule-1  (if nothing)
        (then (->q1 message |^group| 2 transience - time step
                  specification)))

 (rule-2  (if nothing) (then (->screen group 2 complete)))

Rulebase: G3-RB
 Network: ((1 ()) (2 ()) (3 ()) (4 ()) (5 ()) (6 ()) (7 ()) (8 ()))
  Rules:
 (rule-1  (if nothing)
        (then (->q1 message |^group| 3 x-direction grid specification)))

 (rule-2  (if (coordinates are cylindrical)) (then (->q1 |?=| Cartes f)))

 (rule-3  (if (x has |$x| regions)
           (x region |$x| cells |$xfirst| to |$xlast|)
           (y has |$y| regions)
           (y region |$y| cells |$yfirst| to |$ylast|)
           (z has |$z| regions)
           (z region |$z| cells |$zfirst| to |$zlast|)
           ((|$xlast| * |$ylast| * |$zlast|) > 1000))
        (then (->q1 message make sure that the array |^maxfrc| in
                  |^satlit| is at least
                  (|$xlast| * |$ylast| * |$zlast|))))

 (rule-4  (if (x has |$nreg| regions)
           (x region |$nreg| cells |$start| to |$finish|)
           (|$finish| > 1))
        (then (->q1 |?=| Nx |$finish|)))

 (rule-5  (if (x-max > 0.0)) (Then (->q1 |?=| Xulast x-max)))

 (rule-6  (for all x-grid
            (if nothing)
            (then (->q1 |?[]=| Xfrac |$index| |$value|))))
```

```
(rule-7  (if nothing) (then (->screen group 3 complete)))

(rule-8  (if (x has |$n| regions)
             (x region |$n| cells |$first| to |$last|)
             (|$last| > 100))
         (then (->q1 message alter the array nxfr in satlit to be |$last|)))

Rulebase: G4-RB
 Network: ((1 ()) (2 ()) (3 ()) (4 ()) (5 ()))
  Rules:
 (rule-1  (if nothing)
         (then (->q1 message |^group| 4 y-direction grid specification)))

 (rule-2  (if (y has |$nreg| regions)
             (y region |$nreg| cells |$start| to |$finish|)
             (|$finish| > 1))
         (then (->q1 |?=| Ny |$finish|)))

 (rule-3  (if (y-max > 0.0)) (Then (->q1 |?=| Yvlast y-max)))

 (rule-4  (for all y-grid
             (if nothing)
             (then (->q1 |?[]=| Yfrac |$index| |$value|))))

 (rule-5  (if nothing) (then (->screen group 4 complete)))

Rulebase: G5-RB
 Network: ((1 ()) (2 ()) (3 ()) (4 ()) (5 ()))
  Rules:
 (rule-1  (if nothing)
         (then (->q1 message |^group| 5 z-direction grid specification)))

 (rule-2  (if (z has |$nreg| regions)
             (z region |$nreg| cells |$start| to |$finish|)
             (|$finish| > 1))
         (then (->q1 |?=| Nz |$finish|)))

 (rule-3  (if (z-max > 0.0)) (Then (->q1 |?=| Zwlast z-max)))

 (rule-4  (for all z-grid
             (if nothing)
             (then (->q1 |?[]=| Zfrac |$index| |$value|))))

 (rule-5  (if nothing) (then (->screen group 5 complete)))

Rulebase: G6-RB
 Network: ((1 ()) (2 ()))
 (rule-1  (if nothing)
         (then (->q1 message |^group| 6 body fitted coordinates)))

 (rule-2  (if nothing) (then (->screen group 6 complete)))
```

```
Rulebase: G7-RB
 Network: ((1 ()) (2 ()) (3 ()) (4 ()) (5 ()) (6 ()))
  Rules:
 (rule-1  (if nothing)
        (then (->q1 message |^group| 7 variables - including porosities -
                named stored and solved)))

 (rule-2  (if (store-variables is instantiated))
        (then (->q1 |?[]| Store store-variables)))

 (rule-3  (if (flow-regime is turbulent) (store-variables includes enut))
        (then (->q1 |?=| Vist 50) (->q1 |?[]=| Name 50 enut)))

 (rule-4  (for all whole-field-variables
            (if nothing)
            (then (->q1 |?[]| Solutn |$value| y y y n n n))))

 (rule-5  (for all slab-wise-variables
            (if nothing)
            (then (->q1 |?[]| Solutn |$value| y y n n n n))))

 (rule-6  (if nothing) (then (->screen group 7 complete)))

Rulebase: G8-RB
 Network: ((1 ()) (2 ()) (3 ()))
  Rules:
 (rule-1  (if nothing)
        (then (->q1 message |^group| 8 terms - in differential equations -
                and devices)))

 (rule-2  (if (thermal-requirements are thermal))
        (then (->q1 |?[]| Terms h1 n y y n y n)))

 (rule-3  (if nothing) (then (->screen group 8 complete)))

Rulebase: G9-RB
 Network: ((1 ())
        (2 ())
        (3 ())
        (4 ())
        (5 ())
        (6 ())
        (7 ())
        (8 ())
        (9 ())
        (10 ())
        (11 ())
        (12 ())
        (13 ())
        (14 ())
        (15 ()))
```

Rules:

```
(rule-1  (if nothing)
         (then (->q1 message |^group| 9 properties of the medium)))

(rule-2  (if (viscosity-thermal-dependence is not-required))
         (then (->q1 |?=| Enul viscosity)))

(rule-3  (if (viscosity-thermal-dependence is required)
             (viscosity-equation is "a+bt"))
         (then (->q1 |?=| Enul grnd1)
               (->q1 |?=| Enula enula)
               (->q1 |?=| Enulb enulb)))

(rule-4  (if (viscosity-thermal-dependence is required)
             (viscosity-equation is "a+bt+ct**2"))
         (then (->q1 |?=| Enul grnd2)
               (->q1 |?=| Enula enula)
               (->q1 |?=| Enulb enulb)
               (->q1 |?=| Enulc enulc)))

(rule-5  (if (density-thermal-dependence is not-required)
             (fluid-compressibility is incompressible))
         (then (->q1 |?=| Rho1 density)))

(rule-6  (if (density-thermal-dependence is enthalpy)
             (density-equation is "a+bh"))
         (then (->q1 |?=| Rho1 grnd1)
               (->q1 |?=| Rho1a rho1a)
               (->q1 |?=| Rho1b rho1b)))

(rule-7  (if (density-thermal-dependence is enthalpy)
             (density-equation is "1/(a+bh)"))
         (then (->q1 |?=| Rho1 grnd2)
               (->q1 |?=| Rho1a rho1a)
               (->q1 |?=| Rho1b rho1b)))

(rule-8  (if (density-thermal-dependence is not-required)
             (fluid-compressibility is compressible))
         (then (->q1 |?=| Rho1 grnd3)
               (->q1 |?=| Rho1a rho1a)
               (->q1 |?=| Rho1b rho1b)
               (->q1 |?=| Rho1c rho1c)
               (->q1 |?=| Presso presso)))

(rule-9  (if (density-thermal-dependence is temperature)
             (density-equation is "a+bt"))
         (then (->q1 |?=| Rho1 grnd4)
               (->q1 |?=| Rho1a rho1a)
               (->q1 |?=| Rho1b rho1b)))

(rule-10  (if (density-thermal-dependence is temperature)
              (density-equation is "b(p)/t"))
```

```
        (then (->q1 |?=| Rho1 grnd5)
              (->q1 |?=| Rho1b rho1b)
              (->q1 |?=| Presso presso)))

 (rule-11  (if (thermal-requirements are thermal)
               (tmp1-equation is constant))
           (then (->q1 |?=| Tmp1 grnd1) (->q1 |?=| Tmp1a tmp1a)))

 (rule-12  (if (thermal-requirements are thermal) (tmp1-equation is "a+bh"))
           (then (->q1 |?=| Tmp1 grnd2)
                 (->q1 |?=| Tmp1a tmp1a)
                 (->q1 |?=| Tmp1b tmp1b)))

 (rule-13  (if (thermal-requirements is thermal))
           (then (->q1 |?[]=| Prndtl h1 laminar-prndtl)))

 (rule-14  (if (flow-regime is turbulent)) (then (->q1 |?[]| Turmod kemodl)))

 (rule-15  (if nothing) (then (->screen group 9 complete)))

Rulebase: G10-RB
 Network: ((1 ()) (2 ()))
  Rules:
 (rule-1  (if nothing)
          (then (->q1 message |^group| 10 interphase transport processes
                      and properties)))

 (rule-2  (if nothing) (then (->screen group 10 complete)))

Rulebase: G11-RB
 Network: ((1 ())
          (2 ((15 ((14 ()) (16 ()) (17 ()) (18 ()) (19 ())))))
          (3 ())
          (4 ((8 ()) (9 ((10 ()) (11 ()))) (12 ()) (13 ())))
          (5 ())
          (6 ())
          (7 ())
          (20 ()))
  Rules:
 (rule-1   (if nothing)
           (then (->q1 message |^group| 11 initialisation of fields of
                       variables porosities etc)))

 (rule-2   (if (boundary name for obstruction |$number| |$allnodes| is
                     |$name|)
               (|$name| x cells are |$ixf| to |$ixl|)
               (|$name| y cells are |$iyf| to |$iyl|)
               (|$name| z cells are |$izf| to |$izl|))
           (then (porosity prompt (enter the porosity for |$nar e|))
                 (->q1 |?[]| Conpor |$name| porosity cell |$ixf| |$ixl|
                       |$iyf| |$iyl| |$izf| |$izl|)))
```

```
(rule-3   (if (dependent-variables are uninstantiated))
          (then (instantiate dependent-variables)))

(rule-4   (for all dependent-variables
             (if (initial value for |$value| = |$initial-value|))
             (then (->q1 |?[]=| Fiinit |$value| |$initial-value|))))

(rule-5   (if nothing)
          (then (->q1 message *** when restarting deactivate previous
                      |^fiinit| commands and activate the following
                      |^restrt| and |^fiinit| commands)
                (->q1 message "restrt(all)")))

(rule-6   (for all dependent-variables
             (if nothing)
             (then (->q1 message "fiinit(" |$value| ")= readfi"))))



(rule-7   (for all store-variables
             (if (initial value for |$value| = |$initial-value|))
             (then (->q1 |?[]=| Fiinit |$value| |$initial-value|))))

(rule-8   (if (|$value| is h1)
             (h1 at |$type| boundary |$name| is isothermal at
                |$temperature|))
          (then (initial value for h1 = (average temperature from bindings))))

(rule-9   (if ((u1 v1 w1) includes |$value|)
             (|$value| at inlet = |$velocity|))
          (then (initial value for |$value| =
                      (average velocity from bindings))))

(rule-10  (if (or ((|$value| at inlet boundary |$name| is constant is
                         uninstantiated))
                  ((|$value| at inlet boundary |$name| is constant at
                         |$velocity-value|)
                   (|$velocity-value| = 0.0))))
          (Then (|$value| at inlet = 0.1)))

(Rule-11  (if (|$value| at inlet boundary |$name| is constant at
                      |$velocity-value|)
             (|$velocity-value| > 0.0))
          (Then (|$value| at inlet = |$velocity-value|)))

(rule-12  (if (bou  '      .e for inlet |$identity| |$nodes| is |$name|)
             (ke at inlet boundary |$name| is constant at |$tkein|))
          (then (initi  '  '  for ke = (average tkein from bindings))))

(rule-13  (if (boundary name for inlet |$identity| |$nodes| is |$name|)
             (ep at inlet boundary |$name| is constant at |$epin|))
          (then (initial value for ep = (average epin from bindings))))
```

```
(rule-14  (if (surface |$nodes| is part of |$name|)
              (surface |$nodes| is in |$axis| regions |$start| to |$finish|))
          (then (|$name| is in |$axis| regions |$start| to |$finish|)))

(rule-15  (if (|$name| is in |$axis| regions |$start| to |$finish|)
              (|$axis| region |$start| cells |$isf| to |$isl|)
              (|$axis| region |$finish| cells |$iff| to |$ifl|)
              (first |$axis| cell = |$first|)
              (last |$axis| cell = |$last|))
          (then (|$name| |$axis| cells are |$first| to |$last|)))

(rule-16  (if (|$isf| > 1)) (then (first |$axis| cell = (-1 * |$isf|))))

(rule-17  (if (|$isf| = 1)) (then (first |$axis| cell = |$isf|)))

(rule-18  (if (|$axis| has |$n| regions)
              (|$axis| region |$n| cells |$nfirst| to |$nlast|)
              (|$isl| < |$nlast|))
          (then (last |$axis| cell = (-1 * |$ifl|))))

(rule-19  (if (|$axis| has |$n| regions)
              (|$axis| region |$n| cells |$nfirst| to |$nlast|)
              (|$ifl| = |$nlast|))
          (then (last |$axis| cell = |$ifl|)))

(rule-20  (if nothing) (then (->screen group 11 complete)))
```

Rulebase: G12-RB
Network: ((1 ()) (2 ()))
Rules:

```
(rule-1  (if nothing) (then (->q1 message |^group| 12 unused)))

(rule-2  (if nothing) (then (->screen group 12 complete)))
```

Rulebase: G13-RB
Network:

```
((1 ())
 (2 ())
 (3 ((4 ())
 (8 ((5 ()) (6 ()) (7 ())))
 (9 ((4 ()) (5 ()) (6 ()) (7 ())))
 (10 ())
 (11 ())
 (12 ())
 (13 ((5 ()) (6 ()) (7 ())))
 (14 ())
 (15 ())
 (16 ())
 (17 ((5 ()) (6 ()) (7 ())))
 (18 ((5 ()) (6 ()) (7 ())))
 (19 ((5 ()) (6 ()) (7 ())))
 (21 ((22 ()) (23 ()) (24 ())))
 (25 ())
```

```
        (26 ())
        (27 ()))
        (20 ((5 ()) (6 ()) (7 ()))
        (28 ()))
```

Rules:

```
(rule-1   (if nothing)
          (then (->q1 message |^group| 13 boundary and internal conditions
                      and special sources)))

(rule-2   (if (dependent-variables are uninstantiated))
          (then (instantiate dependent-variables)))

(rule-3   (if (boundary name for |$type| |$identity| |$nodes| is |$name|)
            (cardinal for surface |$nodes| is |$cardinal|)
            (patch type for |$name| is |$patch-type|)
            (|$name| x cells are |$ixf| to |$ixl|)
            (|$name| y cells are |$iyf| to |$iyl|)
            (|$name| z cells are |$izf| to |$izl|)
            (consider |$phi|)
            (coval for |$phi| at |$type| is |$co| |$val|))
          (then (fire in block relative to |$name|)
              (->q1 |?[]| Patch |$name| |$patch-type| |$ixf| |$ixl|
                    |$iyf| |$iyl| |$izf| |$izl| 1 1)
              (->q1 |?[]| Coval |$name| |$phi| |$co| |$val|)))

(rule-4   (for all dependent-variables
            (if (or ((|$type| is inlet))
                    ((|$type| is outlet) (|$value| is p1))
                    ((|$type| is wall)
                     ((u1 v1 w1 h1 ke ep) includes |$value|))))
            (then (consider |$value|))))

(rule-5   (if (or ((|$cardinal| is west)) ((|$cardinal| is east))))
          (then (|$name| is on constant x)
              (v1 is parallel to |$name|)
              (w1 is parallel to |$name|)
              (u1 is perpendicular to |$name|)))

(rule-6   (if (or ((|$cardinal| is north)) ((|$cardinal| is south))))
          (then (|$name| is on constant y)
              (w1 is parallel to |$name|)
              (u1 is parallel to |$name|)
              (v1 is perpendicular to |$name|)))

(rule-7   (if (or ((|$cardinal| is high)) ((|$cardinal| is low))))
          (then (|$name| is on constant z)
              (u1 is parallel to |$name|)
              (v1 is parallel to |$name|)
              (w1 is perpendicular to |$name|)))

(rule-8   (if ((west low south) includes |$cardinal|)
            (|$name| is on constant |$axis|)
```

```
        (surface |$nodes| is in |$axis| regions |$start| to |$finish|)
        (|$start| = 1)
        (|$axis| region 1 cells |$if| to |$il|))
      (then (|$name| |$axis| cells are |$if| to |$if|)))

(rule-9  (if (|$name| is on constant |$axis|)
        (surface |$nodes| is in |$axis| regions |$start| to |$finish|)
        (|$start| = |$finish|)
        (|$axis| has |$n-regions| regions)
        (|$axis| region |$n-regions| cells |$if| to |$il|))
      (then (|$name| |$axis| cells are |$il| to |$il|)))

(rule-10  (if (surface |$nodes| is in |$axis| regions |$start| to |$finish|)
        (|$axis| region |$start| cells |$isf| to |$isl|)
        (|$axis| region |$finish| cells |$ilf| to |$ill|))
      (then (|$name| |$axis| cells are |$isf| to |$ill|)))

(rule-11  (if ((inlet outlet) includes |$type|))
      (then (patch type for |$name| is |$cardinal|)))

(rule-12  (if (|$type| is wall))
      (then (patch type for |$name| is
            (join (symbol-split 1 |$cardinal|) wall))))

(rule-13  (if (|$velocity| is perpendicular to |$name|)
        (|$velocity| at inlet boundary |$name| is constant at
         |$magnitude|))
      (then (coval for p1 at inlet is fixflu (|$magnitude| * density))))

(rule-14  (if (|$type| is outlet)
        (p1 at outlet boundary |$name| is constant at |$pressure|))
      (then (coval for p1 at outlet is fixp |$pressure|)))

(rule-15  (if (or (((u1 v1 w1) includes |$phi|)
            (|$phi| at inlet boundary |$name| is constant at |$val|))
           ((|$phi| is h1)
            (h1 at inlet boundary |$name| is isothermal at |$val|))))
      (then (coval for |$phi| at inlet is onlyms |$val|)))

(rule-16  (if (|$phi| is h1)
        (h1 at |$type| boundary |$name| is constant-heat-flux at
          |$val|))
      (then (coval for h1 at |$type| is fixflu |$val|)))

(rule-17  (if (|$phi| is perpendicular to |$name|))
      (then (coval for |$phi| at wall is fixval 0.0)))

(Rule-18  (if (flow-regime is laminar) (|$phi| is parallel to |$name|))
      (then (coval for |$phi| at wall is 1.0 0.0)))

(Rule-19  (if (flow-regime is turbulent) (|$phi| is parallel to |$name|))
      (then (coval for |$phi| at wall is grnd2 0.0)))
```

```
(Rule-20  (if ((u1 v1 w1) includes |$phi|)
              (|$phi| is perpendicular to |$name|)
              (|$phi| at inlet boundary |$name| is constant at |$magnitude|))
          (then (value for |$phi| at |$name| = |$magnitude|)))

(rule-21  (if (h1 at |$type| boundary |$name| is |$condition| at
                 |$thermal-value|)
              (coefficient for h1 at |$name| is |$coefficient|))
          (then (coval for h1 at |$type| is |$coefficient| |$thermal-value|)))

(rule-22  (if (|$condition| is isothermal) (|$type| is inlet))
          (then (coefficient for h1 at |$name| is onlyms)))

(rule-23  (if (|$condition| is isothermal) (|$type| is wall))
          (then (coefficient for h1 at |$name| is fixval)))

(rule-24  (if (|$condition| is constant-heat-flux))
          (then (coefficient for h1 at |$name| is fixflu)))

(rule-25  (if (flow-regime is turbulent) ((ke ep) includes |$phi|))
          (then (coval for |$phi| at wall is grnd2 grnd2)))

(rule-26  (if (flow-regime is turbulent)
              (ke at inlet boundary |$name| is constant at |$quantity|))
          (then (coval for ke at inlet is onlyms |$quantity|)))

(rule-27  (if (flow-regime is turbulent)
              (ep at inlet boundary |$name| is constant at |$quantity|))
          (then (coval for ep at inlet is onlyms |$quantity|)))

(rule-28  (if nothing) (then (->screen group 13 complete)))

Rulebase: G14-RB
 Network: ((1 ()) (2 ()))
  Rules:
(rule-1   (if nothing)
          (then (->q1 message |^group| 14 down stream pressure - for free
                    parabolic |flow.|)))

(Rule-2   (if nothing) (then (->screen group 14 complete)))

Rulebase: G15-RB
 Network: ((1 ())
          (2 ())
          (3 ((4 ((6 ((7 ()) (8 ()) (9 ()))) (10 ())))
          (5 ((6 ((7 ()) (8 ()) (9 ())))))))
          (11 ()))
  Rules:
(rule-1   (if nothing)
          (then (->q1 message |^group| 15 termination criteria for sweeps
                    and outer interations)))
```

```
(rule-2   (if nothing) (then (->q1 |?=| Lsweep 100)))

(rule-3   (for all dependent-variables
            (if (residual reference for |$value| = |$resref|))
            (then (->q1 |?[]=| Resref |$value| |$resref|))))

(rule-4   (if (|$value| is p1)
            (inlet-flow-area > 0)
            (total inlet velocity = |$velocity|)
            (initial fluid-density = |$density|))
          (then (residual reference for |$value| =
                      (|->1.0E???|
                       (0.01 * |$Density| * |$velocity| *
                            inlet-flow-area)))))

(rule-5   (if (inlet-flow-area > 0) (total inlet velocity = |$velocity|))
          (then (residual reference for |$value| =
                      (|->1.0E???|
                       (0.01 * |$Velocity| * inlet-flow-area)))))

(rule-6   (if (boundary name for inlet |$identity| |$nodes| is |$name|)
            (cardinal for surface |$nodes| is |$cardinal|)
            (|$phi| is perpendicular to |$name|)
            (|$phi| at inlet boundary |$name| is constant at
                  |$velocity-value|))
          (then (total inlet velocity = (sum velocity-value from bindings))))


(rule-7   (if ((low high) includes |$cardinal|))
          (then (w1 is perpendicular to |$name|)))

(rule-8   (if ((north south) includes |$cardinal|))
          (then (v1 is perpendicular to |$name|)))

(rule-9   (if ((east west) includes |$cardinal|))
          (then (u1 is perpendicular to |$name|)))

(rule-10  (if (fluid-compressibility is incompressible)
            (thermal-requirements are isothermal))
          (then (initial fluid-density = density)))

(rule-11  (if nothing) (then (->screen group 15 complete)))

Rulebase: G16-RB
 Network: ((1 ()) (2 ()))
   Rules:
(rule-1   (if nothing)
          (then (->q1 message |^group| 16 termination criteria for inner
                  iterations)))

(rule-2   (if nothing) (then (->screen group 16 complete)))
```

```
Rulebase: G17-RB
 Network: ((1 ()) (2 ((3 ()) (4 ()) (5 ()) (6 ()))) (7 ()))
  Rules:
 (rule-1   (if nothing)
         (then (->q1 message |^group| 17 under-relaxation and related
                  sources)))

 (rule-2   (for all dependent-variables
             (if (relaxation method for |$value| is |$method|)
                (relaxation factor for |$value| is |$factor|))
             (then (->q1 |?[]| Relax |$value| |$method| |$factor|))))

 (rule-3   (if (|$value| is p1))
         (then (relaxation method for |$value| is linrlx)
             (relaxation factor for |$value| is 0.8)))

 (Rule-4   (if ((u1 v1 w1) includes |$value|))
         (then (relaxation method for |$value| is falsdt)
             (relaxation factor for |$value| is 0.5)))

 (Rule-5   (if ((ke ep) includes |$value|))
         (then (relaxation method for |$value| is falsdt)
             (relaxation factor for |$value| is 0.01)))

 (Rule-6   (if (|$value| is h1))
         (then (relaxation method for |$value| is linrlx)
             (relaxation factor for |$value| is 1.0)))

 (Rule-7   (if nothing) (then (->screen group 17 complete)))

Rulebase: G18-RB
 Network: ((1 ()) (2 ()))
  Rules:
 (rule-1   (if nothing)
         (then (->q1 message |^group| 18 limits on variable values or
                  increments to them)))

 (rule-2   (if nothing) (then (->screen group 18 complete)))

Rulebase: G19-RB
 Network: ((1 ()) (2 ()))
  Rules:
 (rule-1   (if nothing)
         (then (->q1 message |^group| 19 data comminucated by |^satellite|
                  to |^ground|)))

 (rule-2   (if nothing) (then (->screen group 19 complete)))
```

```
Rulebase: G20-RB
 Network: ((1 ()) (2 ()))
   Rules:
 (rule-1   (if nothing)
         (then (->q1 message |^group| 20 control of preliminary printout)))

 (rule-2   (if nothing) (then (->screen group 20 complete)))

Rulebase: G21-RB
 Network: ((1 ()) (2 ()) (3 ()))
   Rules:
 (rule-1   (if nothing)
         (then (->q1 message |^group| 21 frequency and extent of field
                  printout)))

 (rule-2   (for all dependent-variables
             (if nothing)
             (then (->q1 |?[]| Output |$value| y y y y y y))))

 (rule-3   (if nothing) (then (->screen group 21 complete)))

Rulebase: G22-RB
 Network: ((1 ()) (2 ((5 ()) (6 ()))) (3 ((5 ()) (6 ()))) (4 ((5 ()) (6 ()))) (7 ()))
   Rules:
 (rule-1   (if nothing)
         (then (->q1 message group 22 location of spot-value and frequency
                  of residual printout)))

 (rule-2   (if (x monitor cell = |$ixmon|))
         (then (->q1 |?=| Ixmon |$ixmon|)))

 (rule-3   (if (y monitor cell = |$iymon|))
         (then (->q1 |?=| Iymon |$iymon|)))

 (rule-4   (if (z monitor cell = |$izmon|))
         (then (->q1 |?=| Izmon |$izmon|)))

 (rule-5   (if (|$axis| has 1 regions) (|$axis| region 1 cells 1 to 1))
         (then (|$axis| monitor cell = 1)))

 (rule-6   (if (boundary name for outlet 1 |$nodes| is |$name|)
             (surface |$nodes| is in |$axis| regions |$first| to |$last|)
             (|$axis| region |$first| cells |$ff| to |$fl|)
             (|$axis| region |$last| cells |$lf| to |$ll|))
         (then (|$axis| monitor cell =
                    (max 1 (int (|$ff| + ((|$ll| - |$ff|) / 2)))))))

 (rule-7   (if nothing) (then (->screen group 22 complete)))
```

```
Rulebase: G23-RB
 Network: ((1 ()) (2 ()))
  Rules:
 (rule-1   (if nothing)
          (then (->q1 message |^group| 23 variable-by-variable field
                    printout)))

 (rule-2   (if nothing) (then (->screen group 23 complete)))

Rulebase: G24-RB
 Network: ((1 ()) (2 ()))
  Rules:
 (rule-1   (if nothing)
          (then (->q1 message |^group| 24 preparations for continuation
                    |runs.|)))

 (Rule-2   (if nothing) (then (->screen group 24 complete)))

Rulebase: GEOMETRY-RB
 Network: ((1 ()) (2 ()) (3 ()) (4 ()) (5 ()) (6 ()) (7 ()))
  Rules:
 (rule-1 (if (analysis-title is uninstantiated)) (then (ask analysis-title)))

 (rule-2 (if (number-of-dimensions = 2) (coordinates are cartesian))
        (then (axis-1 is x) (axis-2 is y) (axis-3 is unused)))

 (rule-3 (if (number-of-dimensions = 3) (coordinates are cartesian))
        (then (axis-1 is x) (axis-2 is y) (axis-3 is z)))

 (rule-4 (if (number-of-dimensions = 2) (coordinates are cylindrical))
        (then (axis-1 is unused) (axis-2 is radial) (axis-3 is axial)))

 (rule-5 (if (number-of-dimensions = 3) (coordinates are cylindrical))
        (then (axis-1 is circumferential)
             (axis-2 is radial)
             (axis-3 is axial)))

 (rule-6 (if (conversion-factor > 0))
        (then (run enter-nodes) (run geometry) (use name-walls-rb)))

 (rule-7 (if (trace is on)) (then (->screen <nl> <- geometry-rb <nl>)))


Rulebase: GRID-RB
 Network: ((1 ()))
  Rules:
 (rule-1 (if (delta > 0) (aspect-ratio > 0))
        (then (run mesh-regions) (run assert-grid-information)))
```

Rulebase: INITIAL-RB
Network: ((1 ()))
Rules:
(rule-1 (if nothing)
(then (instantiate targetusermodel) (instantiate usermodel)))

Rulebase: POROSITY-DEFINITION-RB
Network: ((1 ()) (2 ()))
Rules:
(rule-1 (if (porosity-definition is |constant-0.0|))
(Then (porosity status fixed) (porosity = 0.0)))

(Rule-2 (if (porosity-definition is constant-predefined)
(porosity is uninstantiated))
(then (porosity status fixed)))

Rulebase: DENSITY-THERMAL-DEPENDENCE-RB
Network: ((1 ()) (2 ()))
Rules:
(rule-1 (if (thermal-requirements is isothermal))
(then (density-thermal-dependence is not-required)))

(rule-2 (if (thermal-requirements is thermal)
(density-thermal-dependence is uninstantiated))
(then (ask density-thermal-dependence)))

Rulebase: VISCOSITY-THERMAL-DEPENDENCE-RB
Network: ((1 ()) (2 ()))
Rules:
(rule-1 (if (thermal-requirements is isothermal))
(then (viscosity-thermal-dependence is not-required)))

(rule-2 (if (thermal-requirements is thermal)
(viscosity-thermal-dependence is uninstantiated))
(then (ask viscosity-thermal-dependence)))

Rulebase: TMP1-EQUATION-RB
Network: ((1 ((3 ()))) (2 ((3 ()))))
Rules:
(rule-1 (if (density or viscosity thermal dependence is temperature)
(tmp1-equation is constant))
(then (ask tmp1a)))

(rule-2 (if (density or viscosity thermal dependence is temperature)
(tmp1-equation is "a+bh"))
(then (ask tmp1a) (ask tmp1b)))

(rule-3 (if (or ((density-thermal-dependence is temperature))
((viscosity-thermal-dependence is temperature))))
(then (density or viscosity thermal dependence is temperature)))

```
Rulebase: INLET-FLOW-AREA-RB
 Network: ((1 ()) (2 ((3 ()) (4 ()) (5 ()) (6 ()) (7 ()))))
   Rules:
 (rule-1   (if nothing) (then (inlet-flow-area = 0.0)))

 (Rule-2   (if (boundary name for inlet |$identity| |$nodes| is |$name|)
             (cardinal for surface |$nodes| is |$cardinal|)
             (inlet area for |$nodes| = |$area|))
           (then (inlet-flow-area += |$area|)))

 (rule-3   (if (coordinates are cylindrical)
             ((high low) includes |$cardinal|))
           (then (inlet area for |$nodes| =
                    (0.5 * (Abs (((yc_2) |^| 2) - ((yc_1) |^| 2)))))))

 (rule-4   (if (coordinates are cylindrical) (|$cardinal| is north))
           (then (inlet area for |$nodes| =
                    (abs ((yc_1) * ((zc_2) - (zc_1)))))))

 (rule-5   (if (coordinates are cartesian)
             (axis-1 is x)
             ((north south) includes |$cardinal|))
           (then (inlet area for |$nodes| = (1 * (abs ((xc_2) - (xc_1)))))))

 (rule-6   (if (coordinates are cartesian)
             (axis-3 is z)
             ((north south) includes |$cardinal|))
           (then (inlet area for |$nodes| = (1 * (abs ((zc_2) - (zc_1)))))))

 (rule-7   (if (coordinates are cartesian)
             ((west east low high) includes |$cardinal|))
           (then (inlet area for |$nodes| = (1 * (abs ((yc_2) - (yc_1)))))))

Rulebase: NAME-WALLS-RB
 Network: ((1 ()) (2 ()) (3 ()) (4 ()) (5 ((6 ()) (7 ()))) (8 ()) (9 ()))
   Rules:
 (rule-1   (if (boundary name for |$type| |$identity| |$nodes| is |$name|))
           (then (boundary-names includes |$name|)))

 (rule-2   (if nothing) (then (define surfaces :type 'list)))

 (rule-3   (if (cardinal for surface |$nodes| is |$cardinal|))
           (then (surfaces includes |$nodes|)))

 (rule-4   (if (boundary name for |$type| |$identity| |$nodes| is |$name|)
             ((inlet ... cludes |$type|)
             (surfaces includes |$nodes|))
           (then (surfaces ...cludes |$nodes|)))

 (rule-5   (if (coordinates are cylindrical)
             (number-of-dimensions = 2)
             (cardinal for surface |$nodes| is |$cardinal|)
```

```
        (|$cardinal| is south)
        (first node y coordinate is |$y1|)
        (second node y coordinate is |$y2|)
        (|$y1| = 0)
        (|$y2| = 0))
      (then (surfaces excludes |$nodes|)))

(rule-6  (if nothing) (then (first node y coordinate is (yc_1))))

(rule-7  (if nothing) (then (second node y coordinate is (yc_2))))

(rule-8  (if (surface |$nodes| is part of |$obstruction|)
        (surfaces includes |$nodes|))
      (then (surfaces excludes |$nodes|)))

(rule-9  (for all surfaces
        (if nothing)
        (then (ask boundary name for wall surface |$value|
                ((type text)
                 (disallowedvalues boundary-names)
                 (consequent boundary-names includes answer)))))))))
```

APPENDIX F

LISP Inference Engine - Detail flowcharts

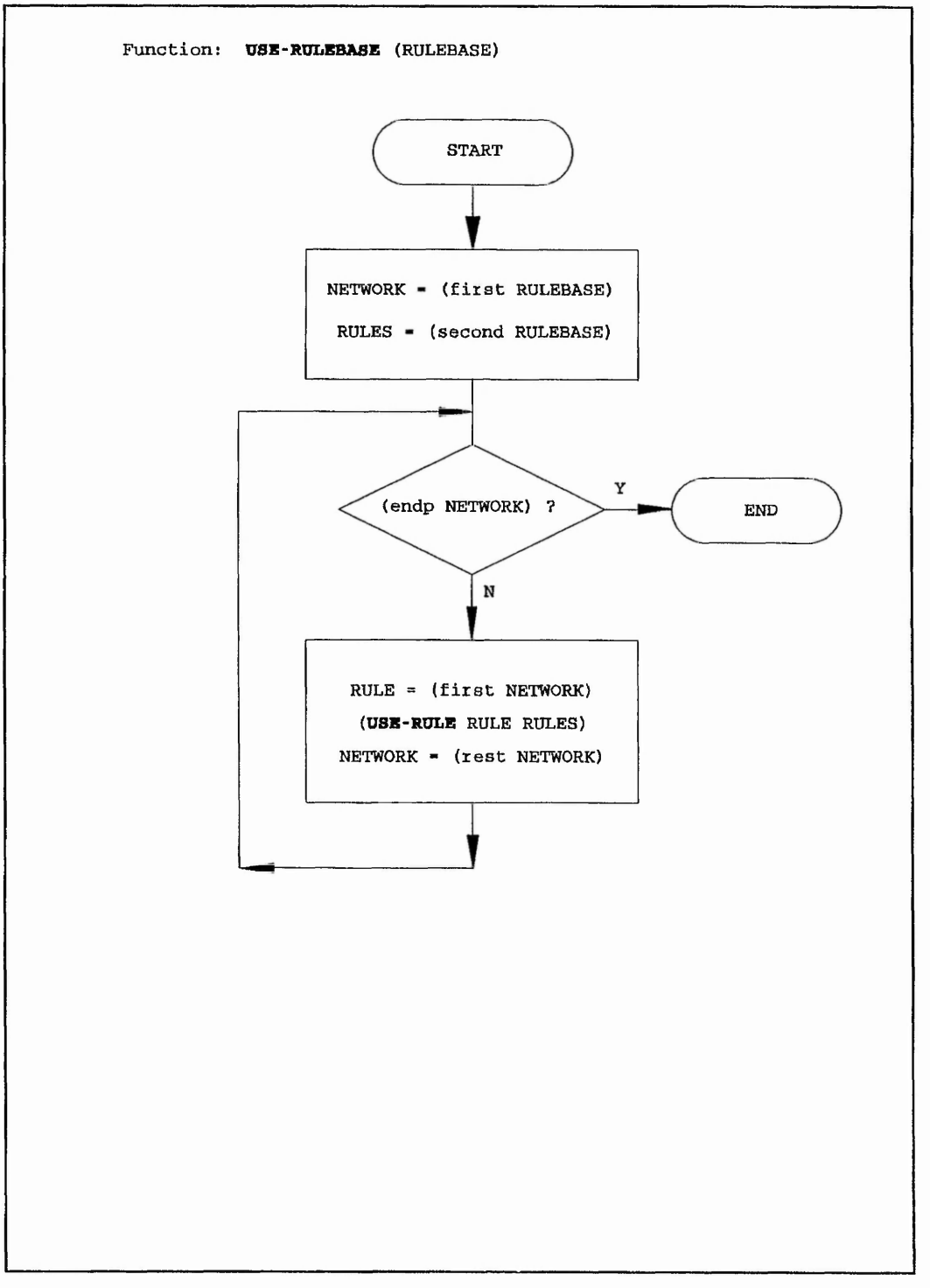


Figure F.1: USE-RULEBASE

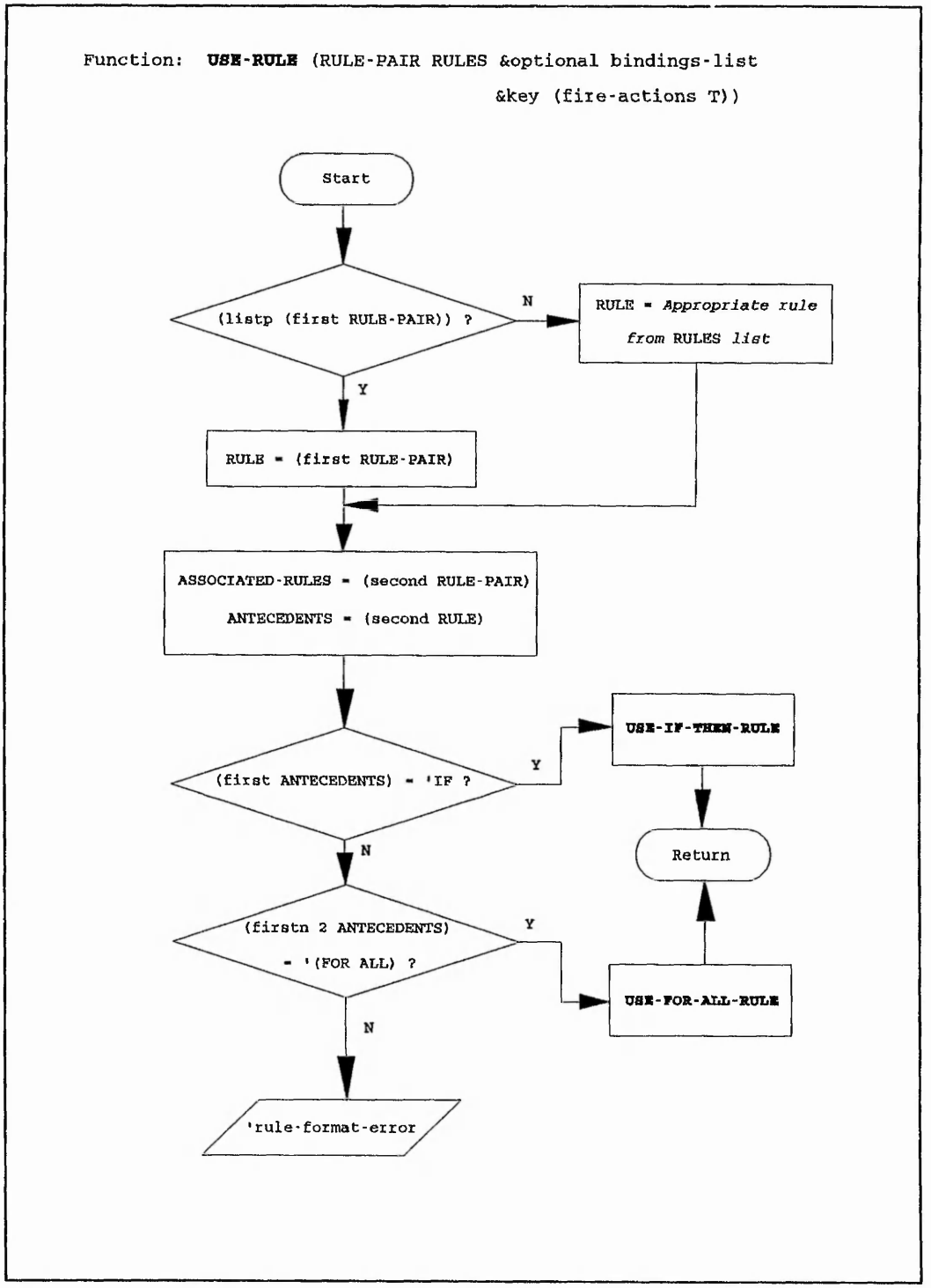


Figure F.2: USE-RULE

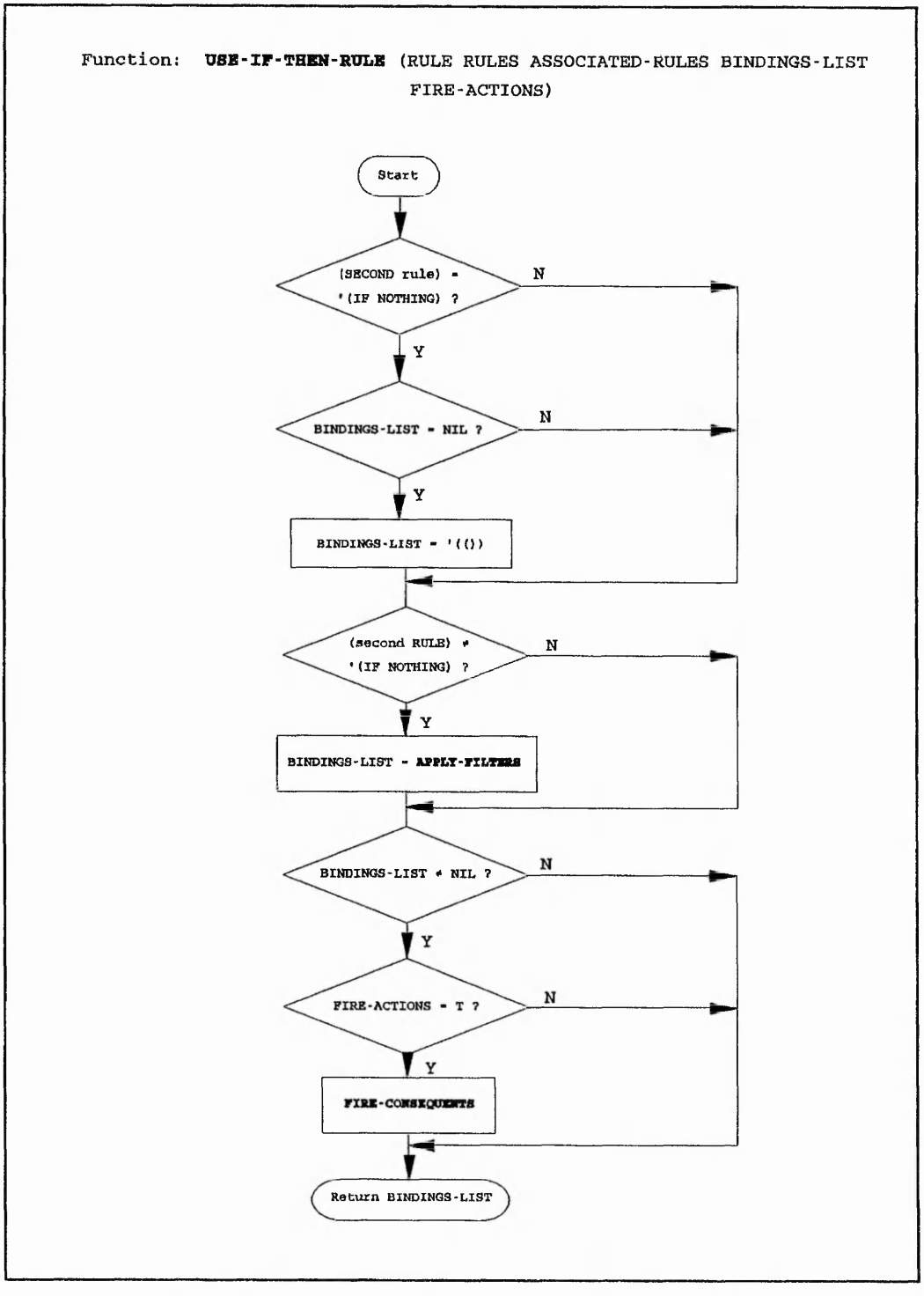


Figure F.3: USE-IF-THEN-RULE

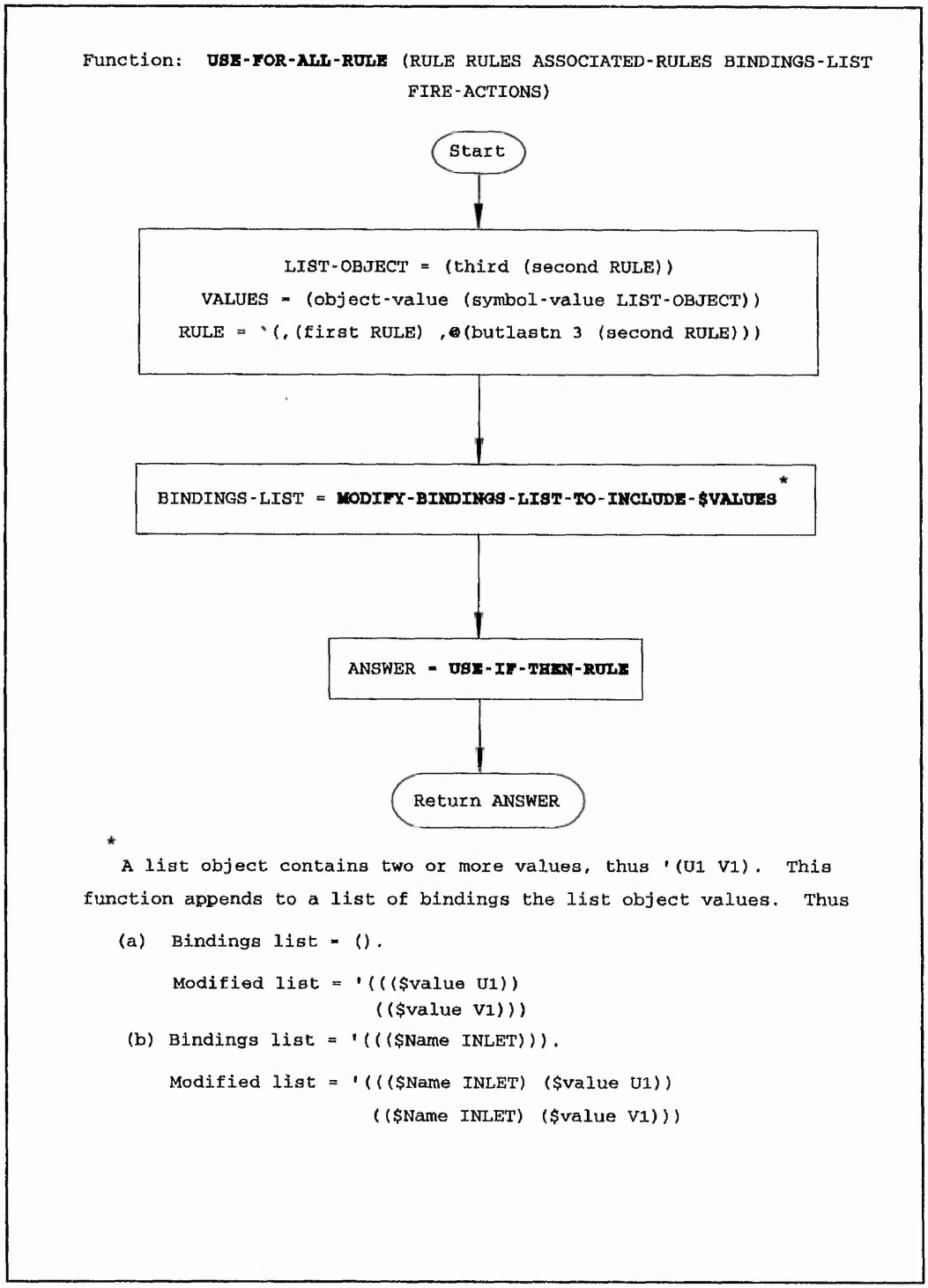


Figure F.4: USE-FOR-ALL-RULE

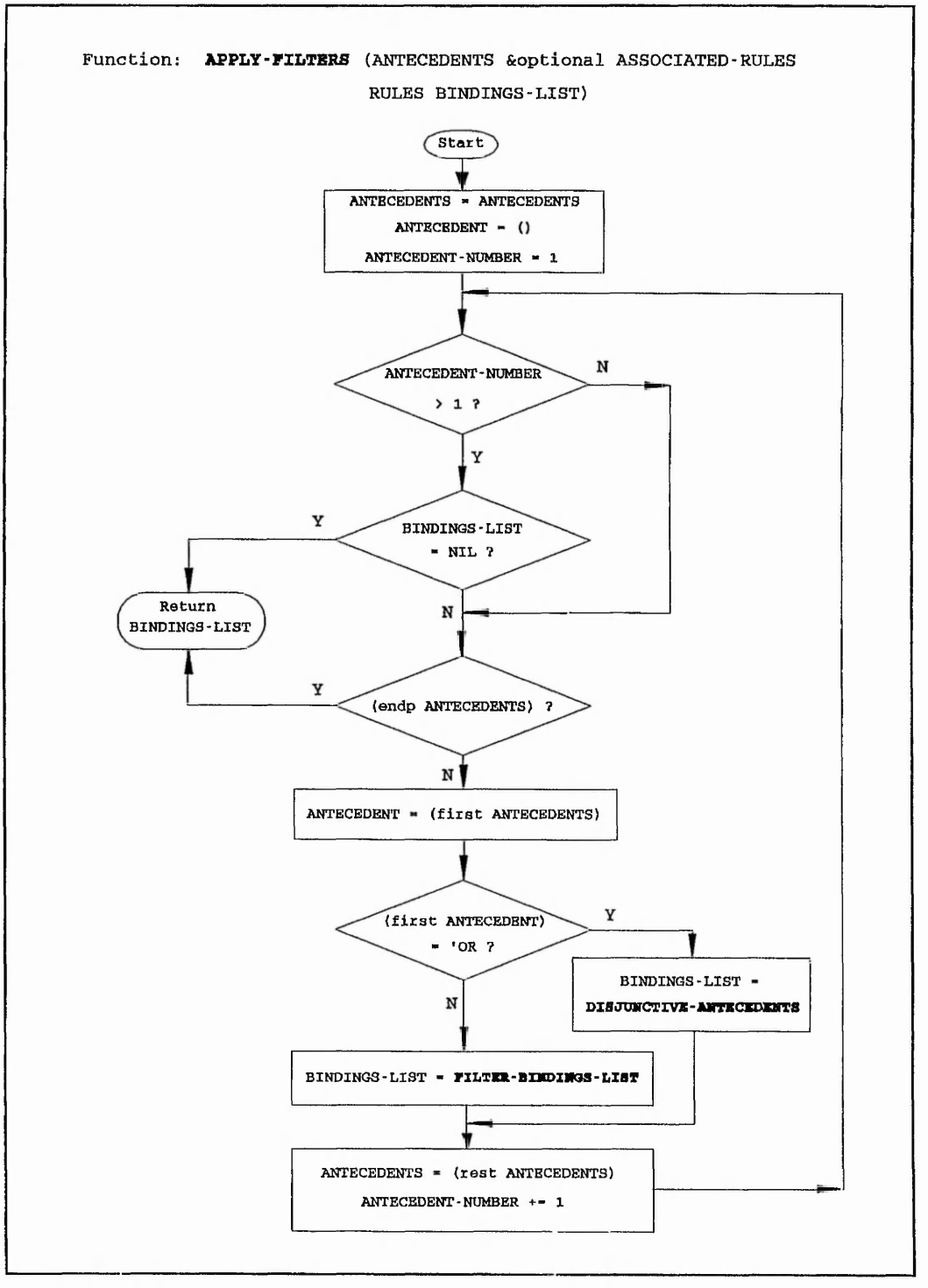


Figure F.5: APPLY-FILTERS

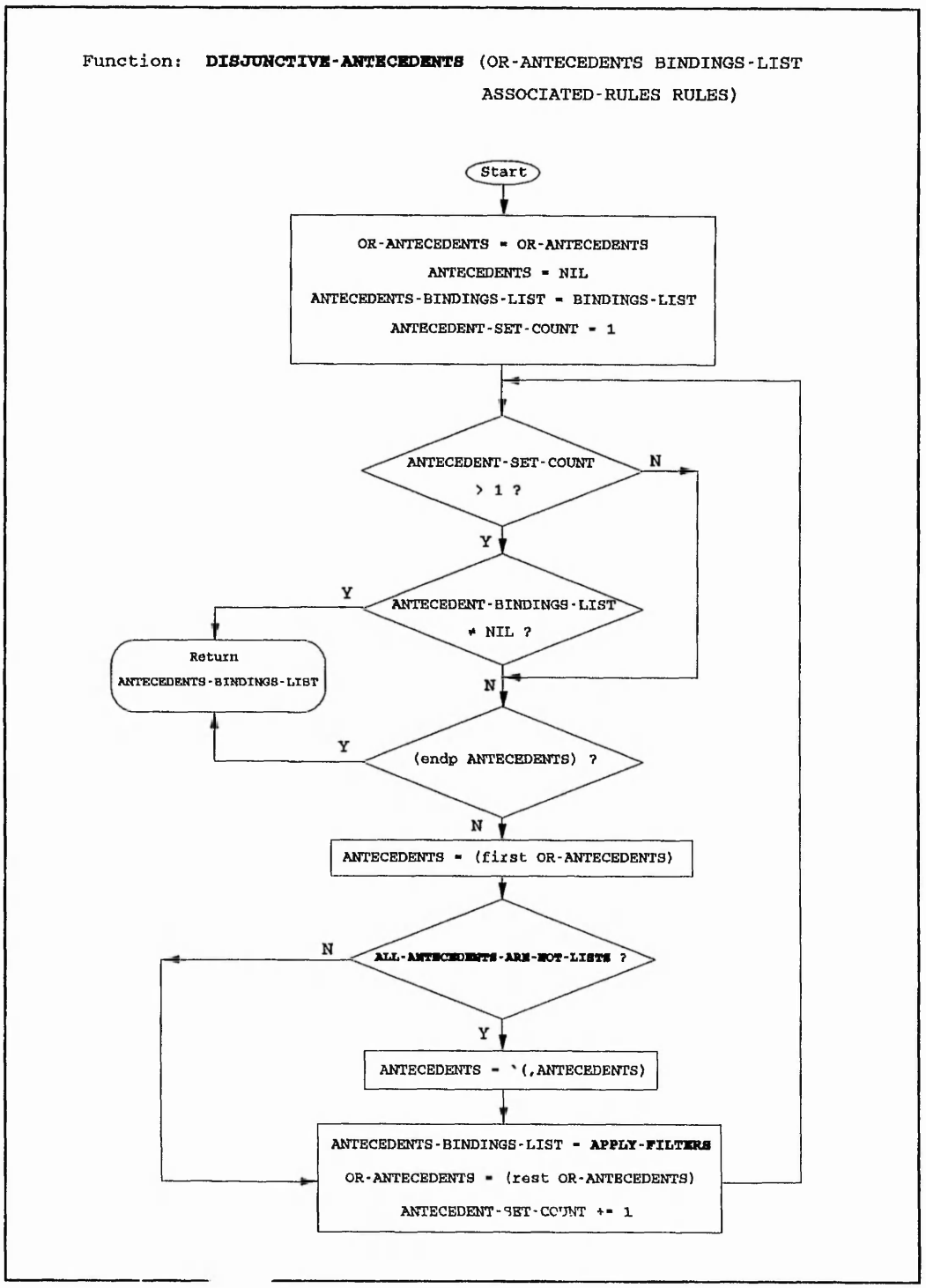


Figure F.6: DISJUNCTIVE-ANTECEDENTS

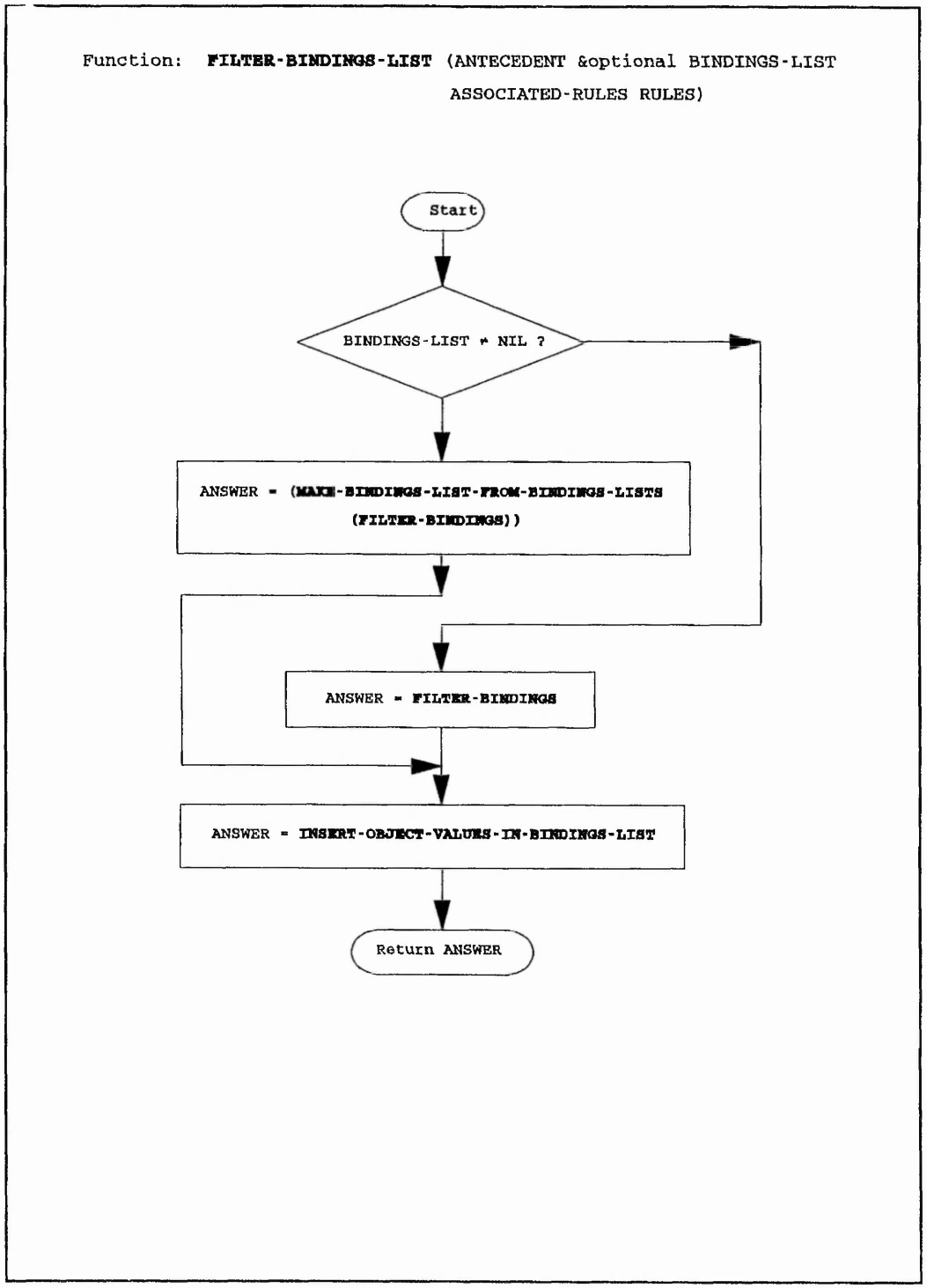


Figure F.7: FILTER-BINDINGS-LIST

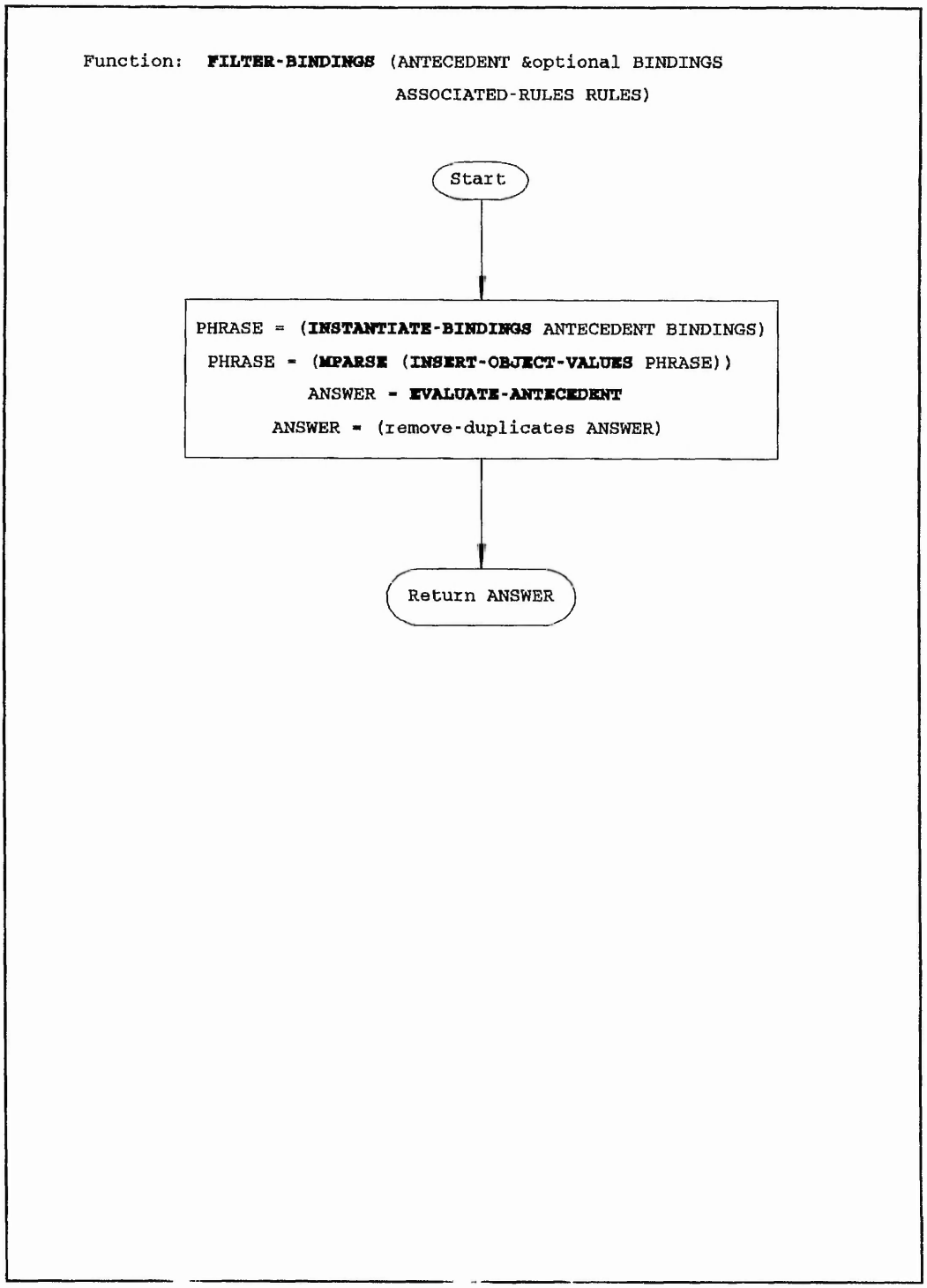


Figure F.8: FILTER-BINDINGS

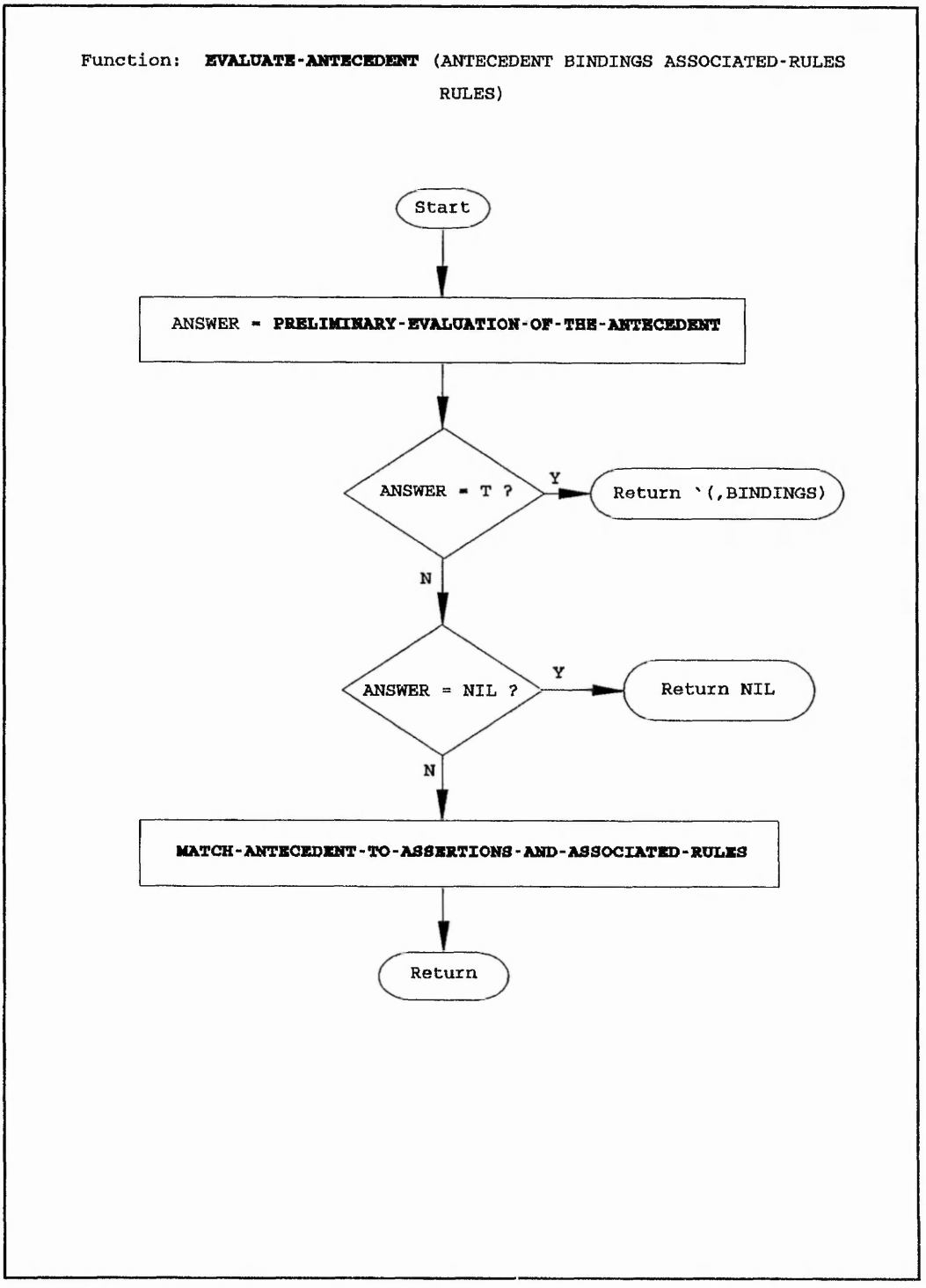
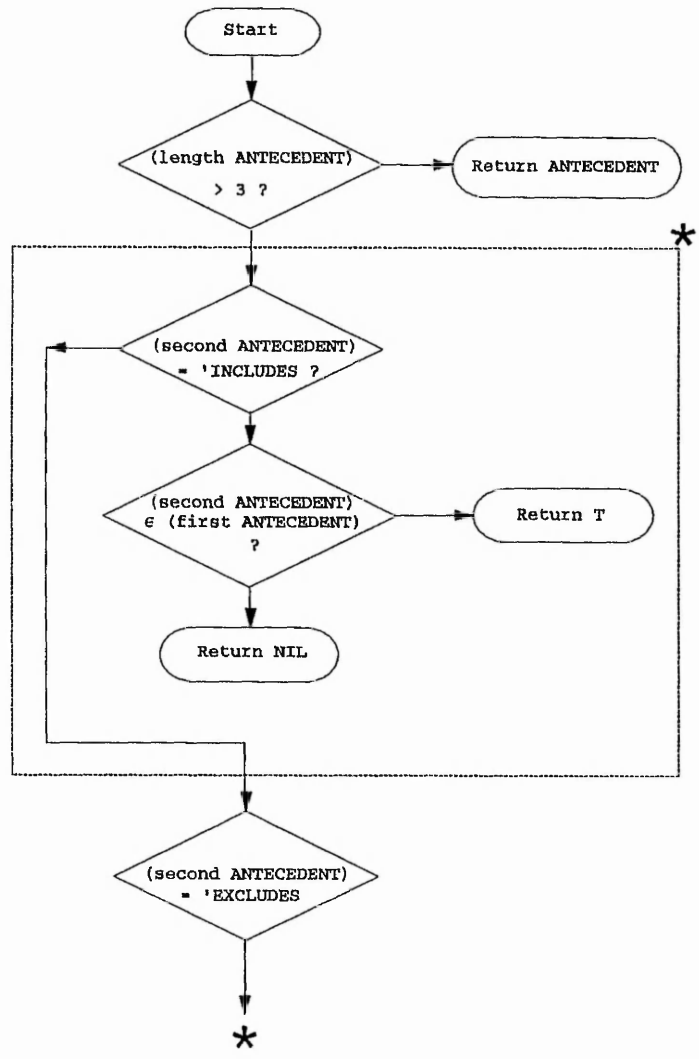


Figure F.9: EVALUATE-ANTECEDENT

Function: **PRELIMINARY-EVALUATION-OF-THE-ANTECEDENT** (ANTECEDENT)



*
The sectioned area of the flow chart is repeated, and modified accordingly, to facilitate the following operators: EXCLUDES, OVERLAPS, =, IS, ARE, <> (not equal to), IS-NOT, ARE-NOT, >=, <=, > and <.

Figure F.10: PRELIMINARY-EVALUATION-OF-THE-ANTECEDENT

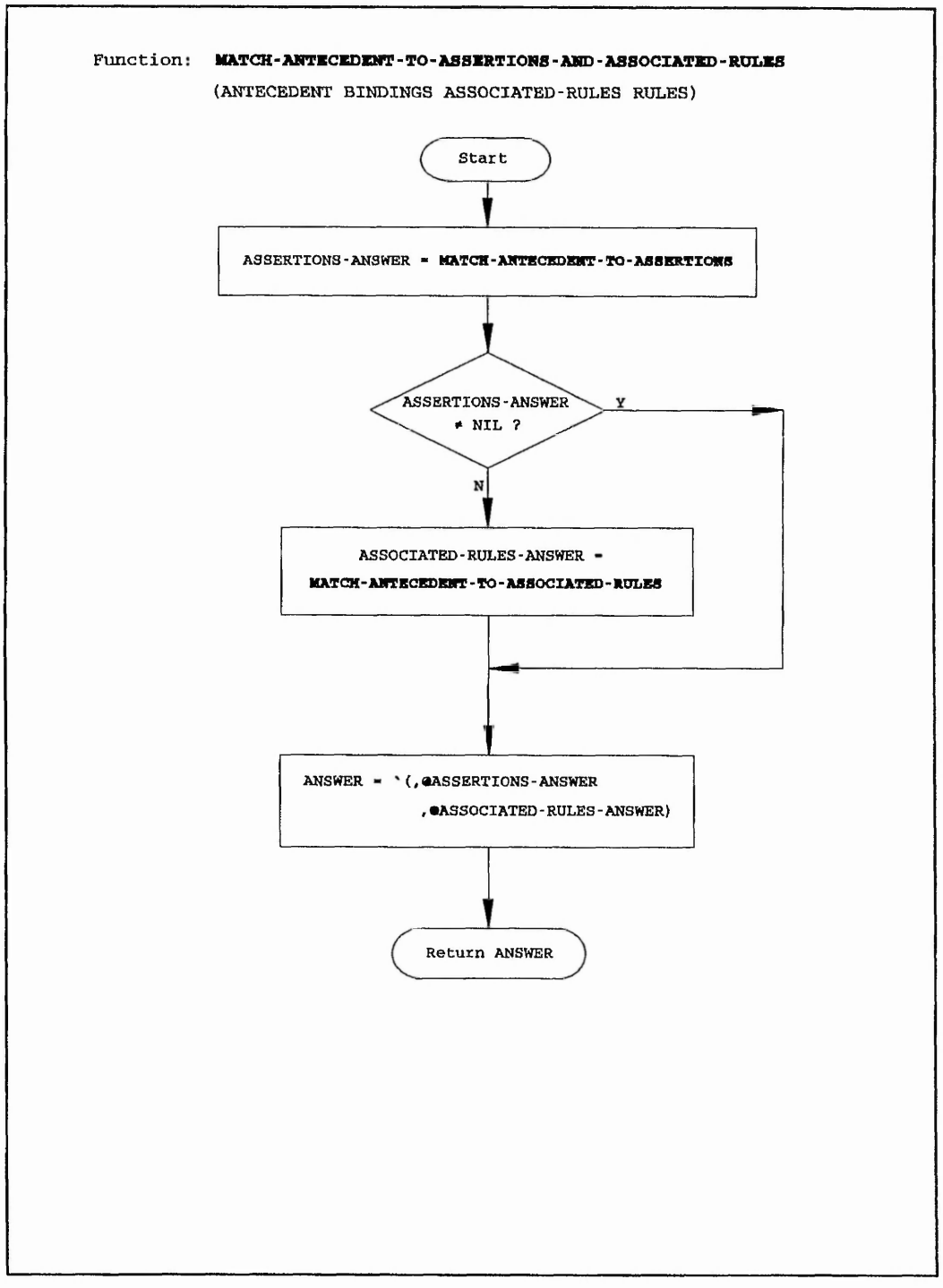


Figure F.11: MATCH-ANTECEDENT-TO-ASSERTIONS-AND-ASSOCIATED-RULES

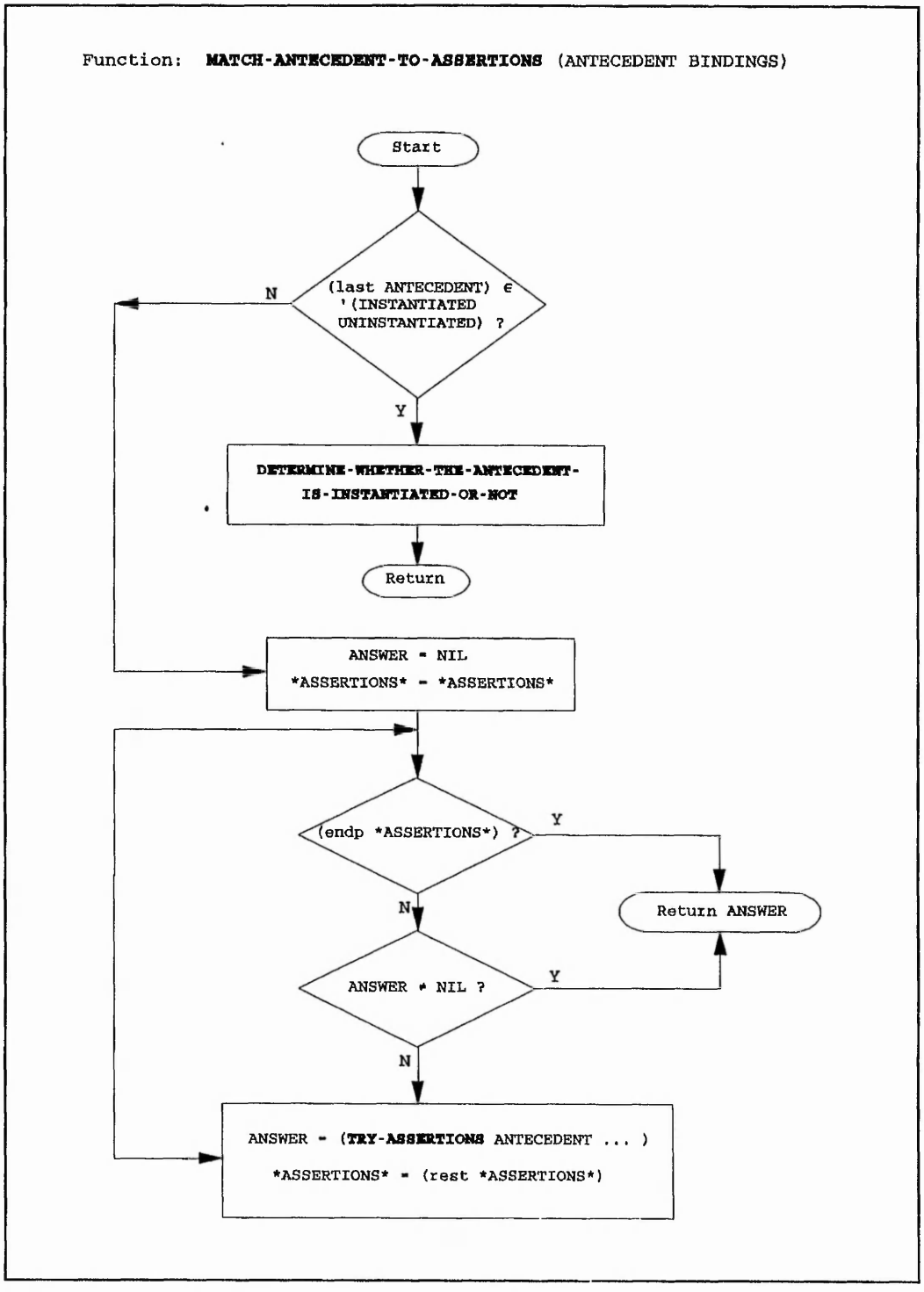


Figure F.12: MATCH-ANTECEDENT-TO-ASSERTIONS

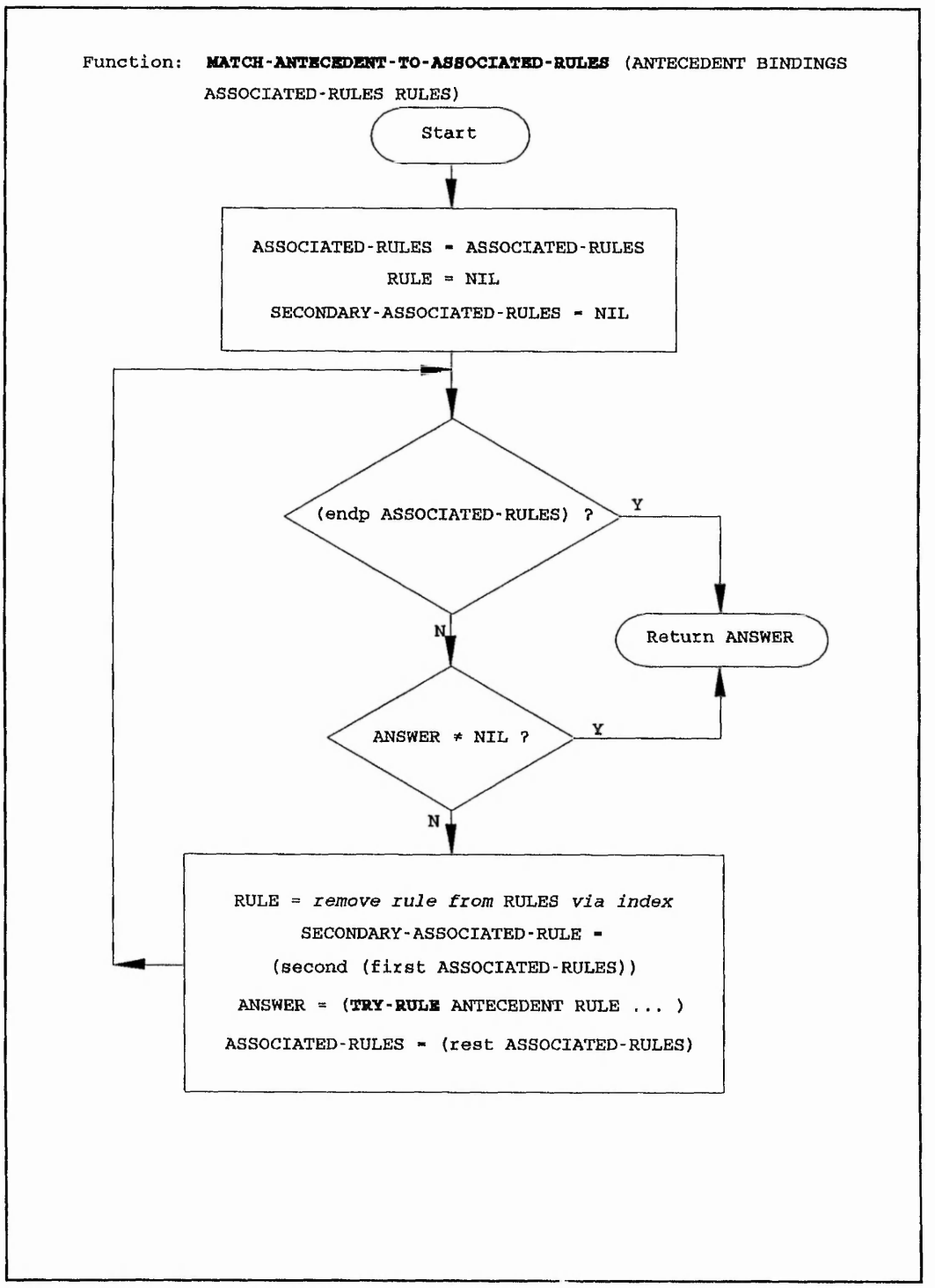


Figure F.13: MATCH-ANTECEDENT-TO-ASSOCIATED-RULES

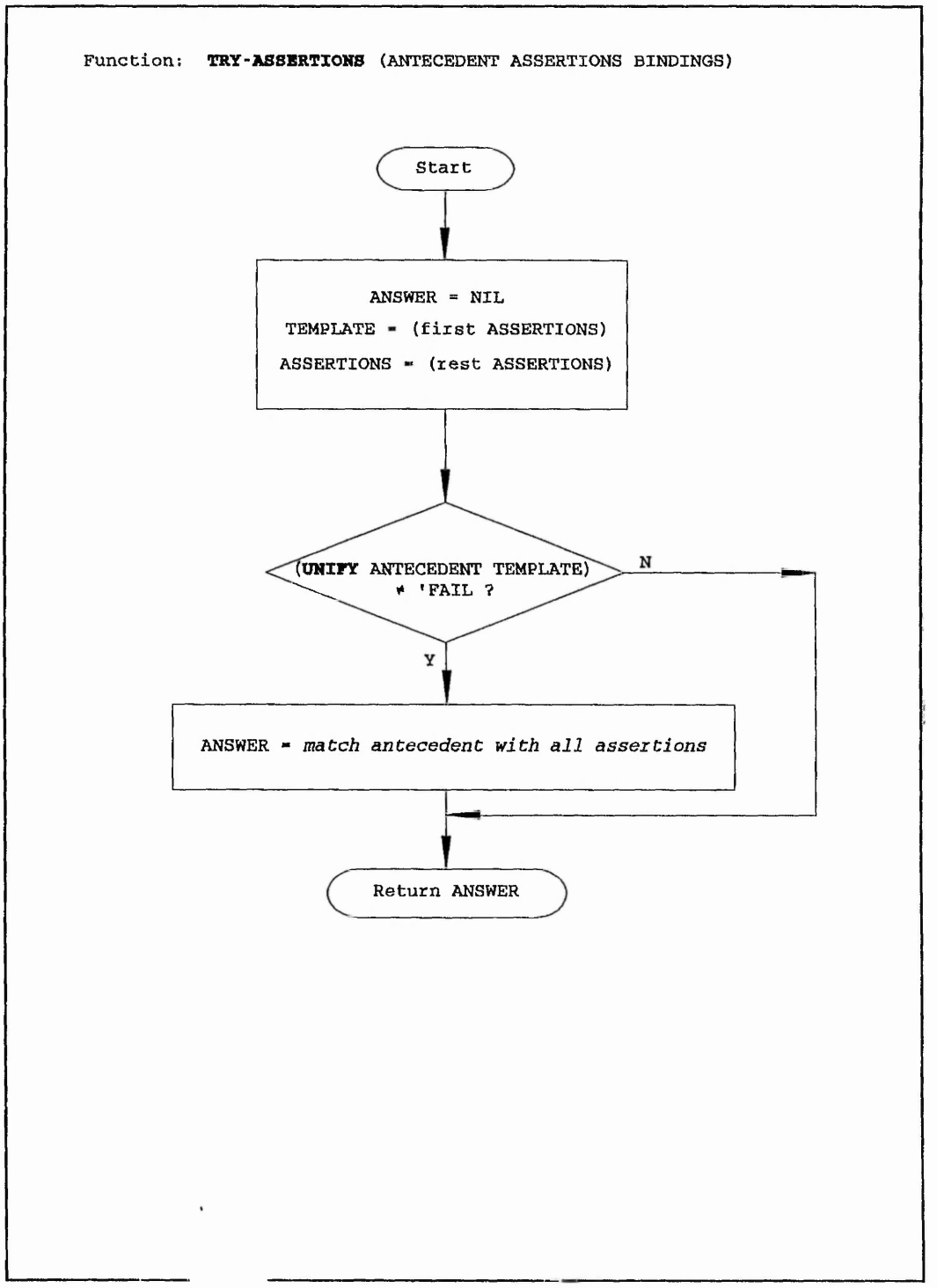


Figure F.14: TRY-ASSERTIONS

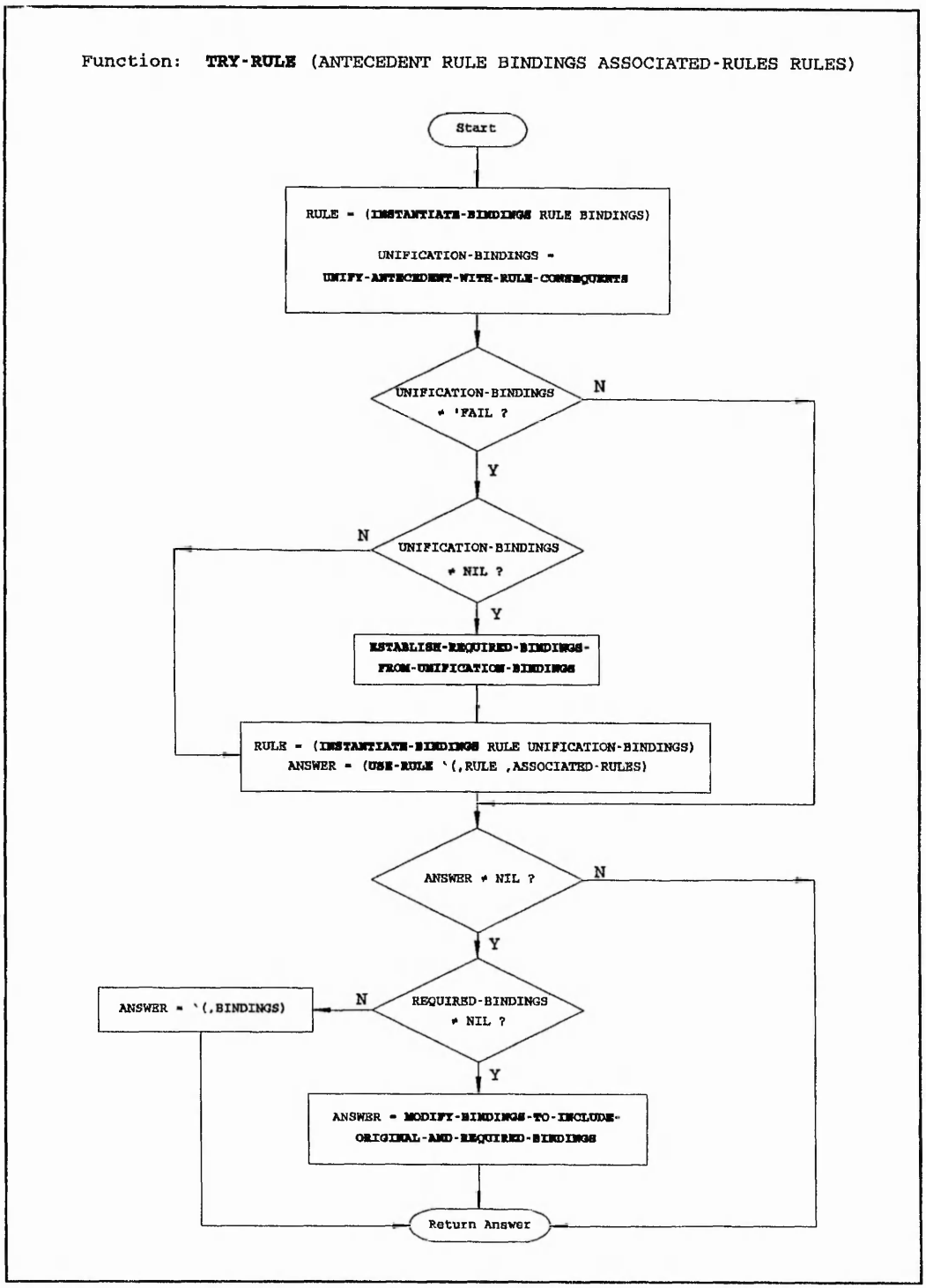


Figure F.15: TRY-RULE

Page intentionally left blank

APPENDIX G

LISP KBFE Code

A copy of the LISP source code

can be obtained on disk by writing to

The author or Dr K Jambunathan

at

The Nottingham Trent University
Faculty of Engineering and Computing
Department of Mechanical Engineering
Burton Street
Nottingham
NG1 4BU
England

Loading the system is carried out by loading one file, **IFE.LSP**. This is performed by entring at the LISP prompt, ==, **(load "ife.lsp")**. The contents of **IFE.LSP** are ...

Filename: **IFE.LSP**

```
(load "initialise.lsp")      ;Initialisation file
(load "macro.lsp")             ;Macros

(load "external.lsp")          ;External Functions
(load "control.lsp")           ;Control file
(load "inference.lsp")         ;Inferencing code

(load "geom.lsp")              ;Geometry specification file

(load "object.lsp")           ;Object file
(load "rules.lsp")            ;Rules
```

End-Of-File

The contents of the other files are listed in the same order as they appear in IFE.LSP, with the exception of OBJECT.LSP and RULES.LSP, which are given in Appendix E.

Filename: **INITIALISE.LSP**

```
;;;; --- INITIALISATIONS ---

(setq pop11::popmemlim 1000000)

(defstruct object
   (Description       nil)
   (Type              nil)
   (Preface           nil)
   (FixedValue        nil)
   (DisallowedValues  nil)
   (AllowedValues     nil)
   (DefaultValue      nil)
   (ComputeValue      nil)
   (Units             nil)
   (Value             nil)
   (Prompt            nil)
   (Help              nil)
   (status            nil)
   (RuleBase          nil))

(defvar Q1 nil)
(defvar *symbol-counter* 0)
(defvar *debug* nil)

(defvar *objects* nil)
(setf *objects* nil)

(defvar TargetUserModel ())
```

```
(setf TargetUserModel 'Experienced)
(defvar UserModel ())
(setf UserModel 'novice)

(defvar *boundaries*       nil)
(defvar *regions*          '((x ()) (y ()) (z ())))
(defvar *nodes*            nil)
(defvar *rule-count*       0)

; The following variables need to be SPECIAL variables as opposed to
; LEXICAL variables because the function MAKE-REGIONS uses the LISP
; function SYMBOL-VALUE.  The function cannot access LEXICAL variables.

(defvar axis ())
(defvar x ())
(defvar x1 ())
(defvar x2 ())
(defvar y ())
(defvar y1 ())
(defvar y2 ())
(defvar z ())
(defvar z1 ())
(defvar z2 ())

(defvar *assertions* ())
(defun reset-assertions ()
(setf *assertions*
      '(((boundary name for $type $identity $nodes is $name))
        ((cardinal for surface $nodes is $cardinal))
        ((surface $surface is part of $obstruction))
        (($dependent-variable at $type boundary $name is $condition at $quantity))
        (($axis has $n regions))
        (($axis region $No cells $first to $last))
        (($axis region $No co-ordinates $first to $last))
        ((surface $nodes is in $axis regions $start to $finish))
        ((surface $nodes interfaces $axis regions $start and $last)))))
(reset-assertions)

(defvar *ife-functions* ())
(setf *ife-functions* '(join ask concat-symbol xc_1 xc_2 yc_1 yc_2
                        zc_1 zc_2 run fetch ->q1 symbol-split abs
                        ->Screen ->1.0e??? int max use define instantiate))

(defvar general-completion  '(stop end finish quit exit complete))

(defvar *manipulation-templates* ())
(setf *manipulation-templates* '((average $variable from bindings)
                  (sum $variable from bindings)
                  ($variable prompt $prompt)
                  ($variable status $status)))
```

End-Of-File

Filename: **MACRO.LSP**

```
(defmacro set-object (name &key
        (Description        nil)
        (Type               nil)
        (Preface            nil)
        (FixedValue         nil)
        (DisAllowedValues   nil)
        (AllowedValues      nil)
        (DefaultValue       nil)
        (ComputeValue       nil)
        (Units          nil)
        (Value              nil)
        (Prompt             nil)
        (Help               nil)
        (Status             nil)
        (RuleBase           nil)
        &allow-other-keys)
   `(progn
      (unless (member ',name '(archive restore target-file fact surfaces))
         (setf *objects* (cons
        (cons ',name '(( :Description        ,Description
                         :Type               ,Type
                         :Preface            ,Preface
                         :FixedValue         ,FixedValue
                         :DisAllowedValues   ,DisallowedValues
                         :AllowedValues      ,AllowedValues
                         :DefaultValue       ,DefaultValue
                         :ComputeValue       ,ComputeValue
                         :Units          ,Units
                         :Value              ,Value
                         :Prompt             ,Prompt
                         :Status             ,(if status status "fixed)
                         :Help               ,Help
                         :RuleBase           ,RuleBase)))
                         *objects*)))
        (setf ,name
           (make-object :Description        ,Description
                         :Type               ,Type
                         :Preface            ,Preface
                         :FixedValue         ,FixedValue
                         :DisAllowedValues   ,DisallowedValues
                         :AllowedValues      ,AllowedValues
                         :DefaultValue       ,DefaultValue
                         :ComputeValue       ,ComputeValue
                         :Units          ,Units
                         :Value              ,Value
                         :Prompt             ,Prompt
                         :Status             ,(if status status "fixed)
                         :Help               ,Help
                         :RuleBase           ,RuleBase
                         :Allow-other-keys t))))
```

```
(defmacro remember-rule (rulebase rule)
    ; 11 April 1992
    ;
    ; Calling Function:  All rulebase files

    `(let ((macro-rule (re-structure-rule-to-rule-template ,rule)))
      (unless (member macro-rule ,rulebase)
        (setf ,rulebase (append ,rulebase (list macro-rule))))))

(defmacro <> (i1 i2) `(not (equal ,i1 ,i2)))

(defmacro modify-list (original-list)
    `(dolist (item ,original-list t)
      (when (y-or-n-p (concatenate 'string "Remove "
                (write-to-string item)
                " "))
        (setf ,original-list (remove item ,original-list :count 1)))))

(defmacro augment-rulebase (rulebase)
    ; 11 April 1992
    ;
    ; Calling Function:  Control structure or == prompt
    ;
    ; This macro take a rulebase that has been built using REMEMBER-RULE and
    ; restructures the list to contain (NETWORK RULES).

    `(let ((clist (do ((rulebase ,rulebase (rest rulebase))
                (count 1 (incf count)) (lst ()))
               ((endp rulebase) (reverse lst))
               (setf lst (cons (list count) lst)))))
      (do ((rules ,rulebase (rest rules))
          (rule ())
          (prc 1 (incf prc))
          (antecedents ())
          (consequents ())
          (answer ()))
         ((endp rules) clist)

         (setf rule (first rules)
           consequents (rule-consequents rule))

         (do ((rulebase ,rulebase (rest rulebase))
             (src 1 (incf src)))
            ((endp rulebase))

            (setf answer ())

            (dolist (consequent consequents t)
              (when (<> (rule-antecedents (first rulebase))
                  '(nothing))
                (setf answer (unify-consequent-with-antecedents
                      consequent
```

```
                (rule-antecedents (first rulebase)))))
            (when answer
              (rplacd (assoc prc clist)
                '(,(remove-duplicates
                    (cons src
                      (second (assoc prc clist))))))))))
      (setf ,rulebase '(,(create-inference-network clist) ,,rulebase))
      t))

(defmacro -> (object value)
    '(setf (object-value ,object) ,value))

(defmacro ? (object)
    '(object-value ,object))

(defmacro >> (object value)
    '(setf (object-value ,object) ',value))
```

End-Of-File

Filename: **EXTERNAL.LSP**

```
; 15 April 1992
;
; This file contains all of the external functions that need to be called
; for use within the IFE code.

(require 'external)
(external-load-files "mesh.obj" '(("grid" generate-mesh)))
```

End-Of-File

Filename: **CONTROL.LSP**

```
(defvar inference-network ())
(defvar preliminary ())
(defvar synthesise ())

(setf preliminary '(initial-rb Geometry-rb Fluid-rb BC-rb Grid-rb))
(setf synthesise '(g1-rb g2-rb g3-rb g4-rb g5-rb g6-rb g7-rb g8-rb g9-rb
                   g10-rb g11-rb g12-rb g13-rb g14-rb g15-rb
                   g16-rb g17-rb g18-rb g19-rb g20-rb g21-rb g22-rb
                   g23-rb g24-rb))

(defvar inference-chain ())
(setf inference-chain ())

(defun phoenics (&optional bypass)

    (setf inference-chain synthesise)

    (unless bypass
```

```
      (reset)
      (setf inference-network '(,@preliminary ,@synthesise)))

    (setf inference-chain ())
    (dolist (rulebase-name inference-network 'network-complete)
      (use-rulebase rulebase-name))

    (write-to-file))

(defun write-to-file ()

    (fetch 'target-file)

    (with-open-file (file-stream (object-value target-file)
        :direction :output
        :if-does-not-exist :create
        :if-exists :rename)
      (format file-stream "~%Talk=F; Run(1, 1); VDU=TTY"))

    (dolist (statement q1 (format t "~%File has been written~%"))
      (write->q1 statement))

    (with-open-file (file-stream (object-value target-file)
        :direction :output
        :if-does-not-exist :create
        :if-exists :append)
      (format file-stream "Stop")))
```

Filename: **INFERENCE.LSP**

```
(defvar DefaultValueError ())
(setf DefaultValueError "~%Sorry, no Default Value, enter definite value.~%")
(defvar HelpErrorMessage ())
(setf HelpErrorMessage "~%Sorry, no help available.~%")

;;;;; --- INFERENCE FUNCTIONS ---

(defun add-binding (bindings-variable bindings-value bindings)
    ; 3 April 1992
    ;
    ; This takes a variable $BINDINGS-VARIABLE, a datum
    ; $BINDINGS-VALUE and a list
    ; of bindings (()()()...()), and adds the pair
    ; ($BINDINGS-VARIABLE BINDINGS-VALUE). Thus giving
    ; (()()()...()($BINDINGS-VARIABLE BINDINGS-VALUE)).

  (if (eq '$_ bindings-variable)
      bindings
      (cons (make-binding bindings-variable bindings-value) bindings)))

(defun all-antecedents-are-not-lists (antecedents)
    ; 6 April 1992
```

```
      ;
      ; Calling Function:  DISJUNCTIVE-ANTECEDENTS
      ; Return: T or NIL
      ;
      ; This function takes a list, antecedents, which can be a single antecedent,
      ; or a list of multiple antecedents.  If the list consists of all
      ; antecedents it will return NIL, alternatively, if it is only one antecedent
      ; it will return T.

      (do ((antecedents antecedents (rest antecedents))
           (answer ()))
          ((or (endp antecedents) answer) answer)
          (when (not (listp (first antecedents)))
                (setf answer t))))

(defun all-integerp (lst)
   (if (member nil (mapcar #'integerp lst)) nil t))

(defun all-realp (lst)
      ; 14 April 1992

      (if (member nil (mapcar #'floatp lst)) nil t))

(defun all-textp (lst)
      ; 14 April 1992

      (if (member t (mapcar #'numberp lst)) nil t))

(defun all-numberp (lst)
      ; 14 April 1992

      (if (member t (mapcar #'(lambda (item)
                                (if (numberp item) nil t))
                            lst))
          nil   t))

(defun antecedent-is-instantiated (antecedent &aux instantiations)
      ; 2 April 1992
      ;
      ; Calling Function:  DETERMINE-WHETHER-THE-ANTECEDENT-
      ; IS-INSTANTIATED-OR-NOT
      ; Returns: T or NIL
      ;
      ; This function takes an antecedent and appends to it the instantiations one
      ; at a time and checks to see if there exists a match with the assertions.
      ; The variable 'instantiated' is T if there exists a template and one or more
      ; assertions associated with that template.

      (setf instantiations '((= $value) (is $value) (is-not $value) (are $value)
         (are-not $value) (at $value))
         antecedent (butlastn 2 antecedent))
```

```
  (do* ((tails instantiations (rest tails))
        (tail (first tails) (first tails))
        (pattern (append antecedent tail) (append antecedent tail))
        (instantiated ()))
       ((or (endp tails) instantiated)
        (if (eq instantiated 'assertion-present) t nil))

     (do* ((*assertions* *assertions* (rest *assertions*))
           (assertions (first *assertions*) (first *assertions*))
           (template (first assertions) (first assertions)))
          ((or (endp *assertions*) instantiated) instantiated)

          (when (<> (unify pattern template) 'fail)
             (if (and (rest assertions)
                      (antecedent-matches-with-template-assertions
                            pattern (rest assertions)))
                 (setf instantiated 'assertion-present)
                 (setf instantiated 'no-assertions))))))

(defun antecedent-matches-with-template-assertions (antecedent assertions)
    ; 16 April 1992
    ;
    ; Calling Function:  ANTECEDENT-IS-INSTANTIATED
    ; Returns: T or NIL

    (do ((assertions assertions (rest assertions))
         (answer ()))
        ((or (endp assertions) answer) answer)
        (when (<> (match antecedent (first assertions)) 'fail)
           (setf answer t))))

(defun apply-filters (antecedents &optional Associated-rules rulebase-name
                      bindings-list)
    ; 6 April 1992
    ;
    ; Calling Functions:  USE-IF-THEN-RULE.
    ; Returning:  ((()()()) (()()()) .... (()()()))
    ;
    ; This function is the first level of the inferencing process.  The filters
    ; are the antecedents to the rule under consideration and these are applied
    ; to the bindings-list.  Each antecedent is considered one at a time through
    ; the use of subsequent functions, FILTER-BINDINGS-LIST and
    ; FILTER-BINDINGS.
    ; The list of bindings within the bindings-list expands and contracts
    ; relative to the infomation gathered from the assertions and objects.

  (when (intersection '(all apply-filters) *debug*)
      (format t "~%> APPLY-FILTERS.")
      (format t "~%Antecedents: ~a" antecedents)
      (format t "~%Bindings-list ~a" bindings-list)
      (pause))
```

```
    (do ((antecedents antecedents (rest antecedents))
         (antecedent ())
         (antecedent-number 1 (incf antecedent-number)))
        ((or (and (> antecedent-number 1)
                  (not bindings-list))
             (endp antecedents)) bindings-list)

      (setf antecedent (first antecedents))

      (if (eq (first antecedent) 'or)
          (setf bindings-list
                (disjunctive-antecedents (rest antecedent) bindings-list
                    associated-rules rulebase-name))
          (progn
            (update-inference-chain antecedent-number)
            (setf bindings-list
                  (filter-bindings-list antecedent bindings-list
                      associated-rules rulebase-name))))))

(defun ask-fact (pattern &aux (slots (car (last pattern)))
      prompt type assignment consequent units)

    (set-object fact)

    (setf (object-prompt fact) (make-prompt '(enter the ,@(butlast pattern)))
      (object-type fact) (second (assoc 'type slots))
      (object-Description fact) (string-upcase (make-prompt (butlast pattern)))
      (object-allowedvalues fact) (second (assoc 'allowedvalues slots))
      (object-defaultvalue fact) (second (assoc 'defaultvalue slots))
      (object-disallowedvalues fact)
          (second (insert-object-values (assoc 'disallowedvalues slots)))
      (object-units fact) (second (assoc 'units slots))
      (object-help fact) (second (assoc 'help slots))
      consequent (rest (assoc 'consequent slots))
      assignment (instantiate-assignment-according-to-type (object-type fact)))

    (ask-object 'fact)

    (remember-assertion '(,@(butlast pattern) ,assignment ,(object-value fact)))

    (when consequent
      (fire-consequent '(,@(butlast consequent) ,(object-value fact))))

    (object-value fact))

(defun instantiate-assignment-according-to-type (type)
    (cond ((member type '(text string logical) :test #'equal) 'is)
      ((member type '(integer-list real-list list text-list) :test #'equal) 'are)
      ((member type '(real integer) :test #'equal) '=)))
```

```
(defun ask-object (object
        &aux (type (object-type (symbol-value object)))
        (value ()))
    ; 14 April 1992
    ;
    ; Calling Functions:  CONTINUE-TO-INSERT-OBJECT-VALUES,
    ;           FETCH, MAKE-ASK-FORM

    (when (not (object-prompt (symbol-value object)))
      (setf (object-prompt (symbol-value object))
        (concatenate 'string (string-upcase
            (write-to-string object)) " ")))

    (cond ((and (member type '(text string))
          (object-allowedvalues (symbol-value object)))
       (setq value (menu object)))
      ((eq type 'text)
       (setq value (enter-text object)))
      ((member type '(integer real))
       (setq value (enter-numeric object)))
      ((member type '(integer-list text-list real-list list))
       (setq value (enter-list object)))
      ((eq type 'string)
       (setq value (enter-string object)))
      (t t))
    (setf (object-value (symbol-value object)) value))

(defun average-variable-from-bindings (variable bindings-list &aux answer)
    ; 2 June 1992
    ;
    ; Calling Function:  MANIPULATE-BINDINGS

    (do ((bindings bindings-list (rest bindings))
       (count 1 (incf count))
       (sum 0)
       (bindings-alist ()))
      ((endp bindings) (setf answer (float (/ sum (- count 1)))))

      (setf sum (+ sum (second (assoc variable (first bindings))))))

    (setf bindings-list
      (remove-variable-pair-from-bindings-list
        variable bindings-list)
      variable
      (intern (coerce (cddr (butlast (coerce (write-to-string variable)
                  'list)))
          'string)))

    (subst answer '(average ,variable from bindings)
      bindings-list :test #'equal))
```

```
(defun binding-is-a-required-binding (binding required-bindings)
    ; 3 April 1992
    ;
    ; Calling Function:  MODIFY-BINDINGS-TO-INCLUDE-ORIGINAL-
    ; AND-REQUIRED-BINDINGS
    ; Returns:  T or NIL
    ;
    ; This function determines whether the binding ($variable value) is a
    ; required-binding.  This is true if the $variable appears as the second
    ; argument in any one of the required-bindings.

    (do ((required-bindings required-bindings (rest required-bindings))
         (answer ()))
        ((or (endp required-bindings) answer) answer)
        (when (equal (second (first required-bindings)) (first binding))
            (setf answer t))))

(defun bindings-manipulation-template (phrase)
    ; 11 April 1992
    ;
    ; Calling Functions:  ESTABLISH-REQUIRED-BINDINGS-FROM-
    ; UNIFICATION-BINDINGS,
    ; MANIPULATE-CURRENT-BINDINGS-LIST
    ; Returns: T or NIL

    (manipulation-template-? phrase))

(defun bindings-p (p &aux lst)
    ; 3 April 1992
    ;
    ; Determines whether the symbol P is a bindings-variable by looking at the
    ; first element within the coerced list.  If this is a $ then the symbol is a
    ; bindings-variable.

   (setf lst (coerce (write-to-string p) 'list))

   (if (and (> (length lst) 2)
            (eq (intern (coerce '(,(second lst)) 'string)) '$))
       t
       nil))

(defun bindings-p-in-antecedent (antecedent &aux answer)
    ;
    ; 14 July 1993
    ;
   (do ((element ()))
       ((or (endp antecedent) answer) answer)
       (setf element (first antecedent)
             antecedent (rest antecedent))
       (when (bindings-p element) (setf answer t))))
```

```
(defun butfirstn (number lst &optional (length-of-original-list (length lst)))
   (cond ((>= number length-of-original-list) nil)
         ((= (length lst) (- length-of-original-list number)) lst)
         (t (butfirstn number (rest lst) length-of-original-list))))

(defun butlastn (number lst &optional (original-lst lst))
   (cond ((>= number (length original-lst)) nil)
         ((eq (length lst) (- (length original-lst) number)) lst)
         (t (butlastn number (butlast lst) original-lst))))

(defun check-value (object value
         &optional (range nil range-s)
         &aux (type (object-type
                (symbol-value object)))
         (AV (if range-s
                range
                (object-allowedvalues
                   (symbol-value object))))
         (front-operator (first (firstn 2 AV)))
         (front-operand (second (firstn 2 AV)))
         (back-operator (first (lastn 2 AV)))
         (back-operand (second (lastn 2 AV)))
         (form nil))
      ; 14 April 1992
      ;
      ; Calling Functions:  ENTER-NUMERIC, ENTER-TEXT
      ;
      ; This function CHECKS the VALUES of objects with their ALLOWEDVALUES
      ; slot.

      (when (intersection '(all check-value) *debug*)
        (format t "~%> CHECK-VALUE.")
        (format t "~%Variable: ~a" object)
        (format t "~%Value: ~a" value)
        (format t "~%AllowedValues: ~a" av)
        (format t "~%Front operator: ~a" front-operator)
        (Format t "~%Front operand: ~a" front-operand)
        (format t "~%Back operator: ~a" back-operator)
        (format t "~%Back operand: ~a" back-operand)
        (pause))

      (cond ((not AV) (setq form t))
        ((member type '(integer real))
         (cond ((eq (length AV) 2)
              (setq form `(,front-operator ,value ,front-operand)))
             (t
               (setq form
                  `(and (,front-operator ,value ,front-operand)
                     (,back-operator ,value ,back-operand))))))
        (t (setq form `(member ',value '(,@AV)))))

      (when (intersection '(all check-value) *debug*)
```

```
        (format t "~%Form: ~a" form)
        (pause))

      (cond ((eval form) t)
        ((equal object 'fact)
         (format t "~%Allowed values for ~aare ~a.~%"
           (object-Description (symbol-value object)) AV))
        (t (format t "~%Allowed values for ~a are ~a.~%" object AV)
          NIL)))

(defun concat-symbol (&rest parts)
    ; 11 April 1992
    ;
    ; Calling Function:  JOIN
    ; Returns: A symbol which is joined together ie (this is an example) becomes
    ;          THISISANEXAMPLE

   (if (listp (first parts))
      (intern (apply #'concatenate 'string (mapcar #'string (car parts))))
      (intern (apply #'concatenate 'string (mapcar #'string parts)))))

(defun continue-to-insert-object-values (phrase)
    ; 11 April 1992
    ;
    ; Calling Function:  INSERT-OBJECT-VALUES
    ; Returns:  PHRASE

   (cond ((endp phrase) nil)
      ((listp (first phrase))
       (cons (continue-to-insert-object-values (first phrase))
          (continue-to-insert-object-values (rest phrase))))
      ((eval '(object-p ,(first phrase)))
       (when (or (not (object-value (symbol-value (first phrase))))
              (eq (object-status (symbol-value (first phrase)))
                 'volatile))
          (when (eq (object-status (symbol-value (first phrase)))
                 'volatile)
             (setf (object-value (symbol-value (first phrase))) ()))
          (when (not (try-object-slots (first phrase)))
             (ask-object (first phrase))))
       (cons (object-value (symbol-value (first phrase)))
          (continue-to-insert-object-values (rest phrase))))
      (t
        (cons (first phrase)
          (continue-to-insert-object-values (rest phrase))))))

(defun create-inference-network (nodal-list &aux (branch-nodes ()))
    ; 11 April 1992
    ;
    ; Calling Function:  AUGMENT-RULEBASE

    (setf branch-nodes (find-network-branches () nodal-list))
```

```
    (create-network nodal-list branch-nodes))

(defun create-network (nodal-list branch-nodes &aux node node-alist
      aux-nodal-list)
    ; 11 April 1992
    ;
    ; Calling Function:  CREATE-INFERENCE-NETWORK

    (setf node (first branch-nodes)
      node-alist (assoc node nodal-list))

    (cond ((endp branch-nodes) ())
      ((member node (second (assoc node nodal-list)))
       (setf aux-nodal-list (delete node-alist nodal-list)
          node-alist '(,node ,(remove node (second node-alist))))
       (cons (cons node
             '(((,node
               ,(create-network '(,@aux-nodal-list ,node-alist)
                  (find-network-branches (first branch-nodes)
                    '(,@aux-nodal-list ,node-alist)))))))
          (create-network '(,@aux-nodal-list ,node-alist)
            (rest branch-nodes))))
      (t
        (cons (cons node
              '(,(create-network nodal-list
                  (find-network-branches
                    (first branch-nodes)
                    nodal-list))))
          (create-network nodal-list (rest branch-nodes))))))

(defun debug (&rest fnctns)
    ; 14 April 1992
    ;
    ; Calling Functions:  Any that have diagnostic requirements.

   (when (not fnctns)
      (format t "~%--- Debug ---~%")
      (format t "~%~a~%" *debug*)
      (format t "~%function_name")
      (format t "~%ALL~%PAUSE~%RESET")
      (format t "~%REMOVE function_1 function_2 ... function_n")
       (format t "~%DEBUG> ? ")
      (setf fnctns (read-sentence)))
     (if (not fnctns) *debug*
     (cond ((eq (first fnctns) 'reset)
           (setf *debug* nil))
          ((eq (first fnctns) 'remove)
           (setf *debug* (set-difference *debug* fnctns))
           (debug))
          ((eq (first fnctns) 'all)
           (setf *debug* '(all))
           (debug))
```

```
        (t
          (setf *debug* (union *debug* fnctns))
          (debug)))))

(defun determine-whether-the-antecedent-is-instantiated-or-not
    (antecedent bindings)
    ; 2 April 1992
    ;
    ; Calling Function:  MATCH-ANTECEDENT-TO-ASSERTIONS
    ; Returns: () or ((()()()) (()()()) ... (()()()))
    ;
    ; This function determines whether the pattern is instantiated within the
    ; assertions list.  If it is then the preceeding atom of the phrase, i.e 'is'
    ; or 'are' establishes whether the bindings are returned in a bindings list
    ; form or NIL is returned.

    (if (antecedent-is-instantiated antecedent)
      (if (equal (last antecedent) '(instantiated))
        (if (member (first (lastn 2 antecedent)) '(is are))
          (cons bindings ())
          nil)
        (if (member (first (lastn 2 antecedent)) '(is are))
          nil
          (cons bindings ())))
      (if (equal (last antecedent) '(instantiated))
        (if (member (first (lastn 2 antecedent)) '(is are))
          nil
          (cons bindings ()))
        (if (member (first (lastn 2 antecedent)) '(is are))
          (cons bindings ())
          nil))))

(defun dir (&rest parameters &aux files d (ext "disk$mec:[mec3hartlsl."))
    (if (not parameters)
      (format t "~%(dir '*.* lisp.rbs)  where LISP.RBS must be in double
quotes~%~%")
      (progn
      (setf files (first parameters)
            d (second parameters)
            files
            (coerce
              (rest (butlast (coerce (write-to-string files)
                                     'list))) 'string)
            ext (concatenate 'string ext d "]"))
      (directory (concatenate 'string ext files)))))

(defun disjunctive-antecedents (or-antecedents bindings-list
      associated-rules rulebase-name
      &aux (state ()))
    ; 6 April 1992
    ;
    ; Calling function:  APPLY-FILTERS
```

```
    ; Returns:  ((()()()) (()()()) ... (()()()))
    ;
    ; This performs the inferencing on disjunctive antecedents.  As soon as an
    ; answer is obtained the search stops.

    (do ((or-antecedents or-antecedents (rest or-antecedents))
         (antecedents ())
         (antecedents-bindings-list bindings-list)
         (antecedent-set-count 1 (incf antecedent-set-count)))
        ((or (and (> antecedent-set-count 1) antecedents-bindings-list)
             (endp or-antecedents)) antecedents-bindings-list)
        (setf antecedents (first or-antecedents))

        (when (all-antecedents-are-not-lists antecedents)
          (setf antecedents '(,antecedents)))

        (setf state (first inference-chain)
          inference-chain '((,(first state) (,antecedent-set-count 0)
            ,(third state)) ,@(rest inference-chain))
          antecedents-bindings-list
          (apply-filters antecedents associated-rules
            rulebase-name bindings-list))))

(defun elements-p (p d)
  (and (atom p) (atom d)))

(defun enter-list (object &key (type 'list)
      &aux lst prompt AV DAV DV error-prompt units)
    ; 28 May 1993

    (if (and (not (stringp object))
             (boundp object))
        (progn
          (PrintPreface object)
          (setq prompt (object-prompt (symbol-value object))
            type (object-type (symbol-value object))
            AV (object-allowedvalues (symbol-value object))
            DV (object-defaultvalue (symbol-value object))
            DAV (object-disallowedvalues (symbol-value object))
            Units (object-units (symbol-value object))
            error-prompt (concatenate 'string "The list should be"
              (cond ((eq type 'integer-list)
                     " an integer list")
                    ((eq type 'text-list)
                     " a text list")
                    ((eq type 'real-list)
                      a real list")
                    (t " a list")))))
        (setq prompt object))

    (format t "~%Current value of ~a: ~a" prompt lst)
    (format t "~%")
```

```
    (format t prompt)
    (format t " ")
    (when units (format t "[ ~a ] " units))

    (do ((input (read-sentence) (read-sentence)))
      ((member 'end input :test #'equal)
       (terpri)
       (remove 'end (union (reverse lst) input)))

      (cond ((intersection input '(why w)) (why))
        ((intersection input '(h help))
         (help prompt)
         (enter-list object))
        ((or (and (eq type 'integer-list)
                  (all-integerp input))
             (and (eq type 'text-list)
                  (all-textp input))
             (and (eq type 'real-list)
                  (all-realp input))
             (eq type 'list))
         (setq lst (union (reverse lst) input)))
        (t
          (format t error-prompt)
          (format t "~%Please re-enter.")))
      (format t "~%Current value of ~a: ~a" prompt lst)
      (format t "~%~a " prompt)))

(defun enter-numeric (object &key (terminus '(end next))
      (type nil) &aux prompt DV value units)
    ; 14 April 1992

    (if (and (not (stringp object))
             (boundp object))
      (progn
        (PrintPreface object)
        (setq prompt (object-prompt (symbol-value object))
          type (object-type (symbol-value object))
          units (object-units (symbol-value object))
          DV (object-defaultvalue (symbol-value object))))
      (setq prompt object))

    (when (and DV (not (eq DV 'allow-none)))
      (format t "~%Default value : ~a" DV))

    (if (stringp object)
      (format t prompt)
      (progn (format t "~%") (format t prompt)))
    (format t " ")
    (when units (format t "[ ~a ] " units))
    (setq value (first (read-sentence)))
```

```
      (when (and (member type '(real integer))
                 (numberp value))
        (if (eq type 'integer)
            (setq value (truncate value))
            (setq value (float value))))

      (cond ((member value terminus) value)
        ((intersection (list value) '(w why)) (why)
         (enter-numeric object :type type))
        ((intersection (list value) '(h help))
         (help object)
         (enter-numeric object :type type))
        ((and (eq value nil)
              (not (eq DV 'allow-none)))
         (if (eq DV nil)
             (progn
               (format t DefaultValueError)
               (enter-numeric object :type type))
             DV))
        ((and (member type '(real integer))
              (not (numberp value)))
         (format t "~%You must enter a numeric value.~%")
         (enter-numeric object :type type))
        ((and (stringp object)
              (numberp value))
         value)
        (t (if (check-value object value)
               value
               (enter-numeric object :type type)))))

(defun enter-read-filename (&aux (file nil)
      (file-set nil)
      (file-success nil)
      (success-switch t))

    (setq file (enter-text 'read-filename)
      file-set (list (join (list file '.asrt))
                     (join (list file '.nod))
                     (join (list file '.var)))
      file-success (mapcar #'probe-file file-set))

    (do ((file-set file-set (rest file-set))
         (file-success file-success (rest file-success)))
      ((endp file-set) success-switch)
      (if (not (first file-success))
          (progn
            (setq success-switch nil)
            (when (not (eq 'none file))
              (format t "~%File ~a is NOT present." (first file-set))))
          (format t "~%File ~a is present." (first file-set))))

    (cond (success-switch
```

```
        file)
       ((and (not success-switch) (eq file 'none))
        'none)
       (t (terpri) (enter-read-filename))))

(defun enter-string (object &aux prompt DV value units)
    ; 14 April 1992

    (if (and (not (stringp object))
             (boundp object))
      (progn
        (PrintPreface object)
        (setq prompt (object-prompt (symbol-value object))
          units (object-units (symbol-value object))
          DV (object-defaultvalue (symbol-value object))))
      (setq prompt object))

    (when (and DV (not (eq DV 'allow-none)))
      (format t "~%Default value : ~a" DV))
    (format t "~%")
    (format t prompt)
    (format t " ")
    (when units (format t "[ ~a ] " units))
    (setq value (read-sentence))

    (cond ((intersection value '(w why))
       (why)
       (enter-string object))
      ((intersection value '(h help))
       (help object)
       (enter-string object))
      ((and (= (length value) 0)
            (eq DV 'allow-none))
       nil)
      ((= (length value) 0)
       (if (eq DV nil)
          (progn
            (format t DefaultValueError)
            (enter-string object))
          DV))
      (t
        (setf value (->string value)))))


(defun enter-text (object &aux prompt type AV DV DAV value units)
    ; 14 April 1992

    (if (and (not (stringp object))
             (boundp object))
      (progn
        (PrintPreface object)
        (setq prompt (object-prompt (symbol-value object))
```

```
            type (object-type (symbol-value object))
            AV (object-allowedvalues (symbol-value object))
            units (object-units (symbol-value object))
            DV (object-defaultvalue (symbol-value object))
            DAV (object-disallowedvalues (symbol-value object))))
      (setq prompt object))

    (when (and DV (not (eq DV 'allow-none)))
      (format t "~%Default value : ~a" DV))
    (format t "~%")
    (format t prompt)
    (format t " ")
    (when units (format t "[ ~a ] " units))
    (setq value (first (read-sentence)))

    (cond ((and (eq value nil) (not (eq DV 'allow-none)))
       (if (eq DV nil)
          (progn
             (format t DefaultValueError)
             (enter-text object))
          DV))
      ((intersection (list value) '(w why))
       (why)
       (enter-text object))
      ((intersection (list value) '(h help))
       (help object)
       (enter-text object))
      ((numberp value)
       (format t "~%You must enter a text value. ~%")
       (enter-text object))
      ((stringp object) value)
      ((and (not AV)
          DAV)
       (if (member value DAV :test #'equal)
          (progn
             (format t "~%Sorry, the answer you gave is a DISALLOWEDVALUE.")
             (format t "~%Current disallowed values are ... ~%")
             (dolist (DisAV DAV t)
                (print DisAV))
             (terpri)
             (enter-text object))
          value))
      (t (if (check-value object value)
            value
            (enter-text object)))))

(defun establish-required-bindings-from-unification-bindings
    (bindings unification-bindings antecedent
      &aux (new-bindings bindings) (required-bindings ()))
    ; 3 April 1992
    ;
    ; Calling Function:  TRY-RULE
```

```
    ; Returns:  (()()()) and (()()())
    ;
    ; This function receives three arguments (1) the original-bindings (2) the
    ; unification bindings that have been generated from unifying the antecedent
    ; with the consequents of an associated rule, and (3) the antecedent.  The
    ; function is a MULTIPLE-VALUE-BIND call and as such returns new-bindings
    ; and the required-bindings.  The new-bindings contain unification-bindings
    ; whose second argument is not a bindings-variable, ie ($variable xxxx) as
    ; opposed to ($variable_1 $variable_2), this covers the fact that the
    ; consequent directly relates to a number, text or a list.  However, if the
    ; second argument is a bindings-manipulation-template, as (AVERAGE AXIAL-
    ; VELOCITY FROM BINDINGS) from G11-RB then this has to be included in the
    ; required variables.

    (dolist (pair unification-bindings
        (values new-bindings required-bindings))
      (when (member (first pair) antecedent)
        (if (or (bindings-p (second pair))
                (listp (second pair)))
          (setf required-bindings (cons pair required-bindings))
          (setf new-bindings (cons pair new-bindings))))))

(defun evaluate (consequent &optional bindings-list)
    ; 14 April 1992
    ;
    ; Calling Function:  FIRE-CONSEQUENT

    (when (and (listp consequent)
        (equal (first consequent) '->q1)
        (listp (car (last consequent))))
      (setf consequent `(,@(butlast consequent) ,@(car (last consequent))))))

    (cond ((atom consequent) consequent)
      ((member (first consequent) *ife-functions*)
       (evaluate-function (mparse (evaluate-recursively consequent bindings-list))
          bindings-list))
      (t (evaluate-recursively consequent bindings-list))))


(defun evaluate-recursively (consequent &optional bindings-list)
    ; 14 April 1992
    ;
    ; Calling Function:  EVALUATE, EVALUATE-RECURSIVELY

    (cond ((endp consequent) nil)
      ((listp (first consequent))
       (let ((consequent
              (cons (evaluate (first consequent)
                    bindings-list)
                (evaluate (rest consequent) bindings-list))))

         (if (member (first consequent) *ife-functions*)
```

```
            (cons (evaluate-function (mparse (first consequent)) bindings-list)
               (evaluate-recursively
                  (rest consequent) bindings-list))
            consequent)))

      (t (cons (first consequent)
           (evaluate-recursively (rest consequent) bindings-list)))))


(defun evaluate-function (form bindings-list)
    ; 14 April 1992
    ;
    ; Calling functions:  EVALUATE, EVALUATE-RECURSIVELY

    (eval (make-form form bindings-list)))


(defun evaluate-antecedent (antecedent &optional bindings Associated-rules
rulebase-name)
    ; 3 April 1992
    ;
    ; Calling Function:  FILTER-BINDINGS
    ; Returns: T, (bindings) or ((()()()) (()()()) ... (()()()))
    ;
    ; The function evaluates the antecedent and returns NIL for a failed
    ; antecedent, the orignal bindings in a bindings-list form for a correctly
    ; fired antecedent, or the resulting bindings-list from using the function
    ; MATCH-ANTECEDENT-TO-ASSERTIONS-AND-ASSOCIATED-RULES.

    (let ((answer (preliminary-evaluation-of-the-antecedent antecedent)))
      (cond ((eq answer t) (cons bindings ()))
         ((eq answer nil) nil)
         (t (match-antecedent-to-assertions-and-associated-rules
              antecedent bindings Associated-rules rulebase-name)))))

(defun evaluate-pattern (pattern &optional bindings-list)
    ; 14 April 1992
    ;
    ; Calling Functions:  INSERT-OBJECT-VALUES-IN-BINDINGS-LIST
    ; Returns:  Evaluated pattern.
    ;
    ; This functioncheck to see if there exists a command within the pattern
    ; that requires subsequent evaluateion, ie JOIN.  If there is no such
    ; requirement then it is not necessary to perform the evaluateion.  This
    ; is why the pattern is returned if the success-switch is not T.

    (do ((new-pattern pattern (rest new-pattern))
       (success-switch nil))
      ((or (endp new-pattern) success-switch)
       (if success-switch
          (evaluate (instantiate-bindings pattern bindings-list)
            bindings-list)
```

```
          pattern))
        (when (and (listp (first new-pattern))
                   (eq (first (first new-pattern)) 'join))
          (setq success-switch t))))

(defun extract-bindings-value (binding)
  (second binding))

(defun extract-bindings-variable (binding)
  (first binding))

(defun fetch (object &key type &aux (assignment nil)
        (variable ())
        (bindings nil) (answer ()))
    ; 14 April 1992
    ;
    ; Calling functions:  MAKE-FETCH-FORM
    ;
    ; This function returns the value for an object or a certain phrase, for
    ; example.  AXIAL INLET VELOCITY FOR ENTRY = $VALUE

  (cond ((and (symbolp object)
              (eval `(object-p ,object)))

         (when (eq (object-status (symbol-value object)) 'volatile)
           (setf (object-value (symbol-value object)) ()))

         (when (not (object-value (symbol-value object)))
           (try-object-slots object)
           (when (not (object-value (symbol-value object)))
             (ask-object object)))

         (object-value (symbol-value object)))
        (t
          (setq bindings
                (determine-whether-the-antecedent-is-instantiated-or-not
                  object))
          (cond ((endp bindings)
                 (setf answer (ask-fact object))
                 answer)
                (t (extract-bindings-value (caar (last bindings))))))))


(defun filter-bindings (antecedent &optional bindings associated-rules
        rulebase-name
        &aux (phrase ()) (answer ()))
    ; 2 April 1992
    ;
    ; Calling Function:  FILTER-BINDINGS-LIST
    ; Returns: ((()()()) (()()()) ... (()()()))
    ;
    ; This takes an antecedent, filters the list of bindings, (()()()), through
```

```
    ; it and returns a bindings list

    (when (intersection *debug* '(filter-bindings all))
      (format t "~%>FILTER-BINDINGS.")
      (format t "~%Antecedent: ~a" antecedent)
      (format t "~%Bindings: ~a" bindings))

    (setf phrase (instantiate-bindings antecedent bindings)
      phrase (mparse (insert-object-values phrase)))

    (when (intersection *debug* '(filter-bindings all))
      (format t "~%Phrase: ~a" phrase)
      (pause))

    (setf answer (evaluate-antecedent phrase bindings associated-rules
          rulebase-name)
      answer (remove-duplicates answer :test #'equal))

    (when (intersection *debug* '(filter-bindings all))
      (format t "~%<FILTER-BINDINGS.")
      (format t "~%Answer: ~a~%" answer)
      (pause))

    answer)

(defun filter-bindings-list (antecedent &optional bindings-list
        associated-rules rulebase-name &aux (answer ()))
    ; 2 April 1992
    ;
    ; Calling Function: APPLY-FILTERS
    ; Returns: ((()()()) (()()()) ... (()()()))
    ;
    ; This takes an antecedent and a bindings-list. The bindings within the
    ; bindings-list are systematically filtered one by one. Each list of
    ; bindings are returned as a complete bindings-list. Therefore, the
    ; complete set of bindings-lists for the original bindings-list needs to
    ; be joined together to create one bindings-list. The STREAM-CONCATENATE
    ; function desperately needs removing and a simple joining mechanism
    ; introducing.

    (when (intersection *debug* '(all filter-bindings-list))
      (format t "~%> FILTER BINDINGS LIST.")
      (format t "~%Antecedent: ~a" antecedent)
      (format t "~%Bindings list: ~a" bindings-list)
      (format t "~%Associated rules: ~a~%" associated-rules)
      (pause))

    (if bindings-list
      (setf answer
        (make-bindings-list-from-bindings-lists
          (mapcar #'(lambda (bindings)
              (filter-bindings antecedent bindings
```

```
                    associated-rules rulebase-name))
                bindings-list)))
        (setf answer (filter-bindings antecedent () associated-rules
              rulebase-name)))

    (setf answer (insert-object-values-in-bindings-list answer))

    (when (intersection *debug* '(all filter-bindings-list))
      (format t "~%< FILTER BINDINGS LIST.")
      (format t "~%Antecedent: ~a" antecedent)
      (format t "~%Answer: ~a~%" answer)
      (pause))

    answer)

(defun find-binding (bindings-variable bindings)
    ; 3 April 1992
    ;
    ; Bindings-variable = $X
    ; Bindings = '(($A 2) ($X 4))
    ; Returns = ($X 4)

  (unless (eq '$_ bindings-variable)
      (assoc bindings-variable bindings)))

(defun find-network-branches (root-node connectivity)
    ; 11 April 1992
    ;
    ; Calling Function:  CREATE-INFERENCE-NETWORK, CREATE-NETWORK

    (remove ()
      (mapcar #'(lambda (node)
            (when (or (member root-node (second node))
                  (and (not root-node)
                    (not (second node))))
              (first node)))
        connectivity)))

(defun fire-consequent (consequent &optional bindings)
    ; 10 April 1992
    ;
    ; Calling Function:  FIRE-CONSEQUENTS
    ; Returns:
    ;
    ; This function fires the consequent.  If the consequent is an IFE-FUNCTION
    ; ie RUN ->SCREEN etc, the consequent is fed into the function EVALUATE.
    ; Secondly, if the consequent is OBJECT (= IS ARE INCLUDES EXCLUDES)
    ; ?????,
    ; the function INSTANTIATE-OBJECT is called.  Finally, remember-assertion
    ; is called if appropriate.

    (when (intersection *debug* '(fire-consequent all))
```

```
      (format t "~%> FIRE-CONSEQUENT")
      (format t "~%Consequent: ~a" consequent)
      (format t "~%Bindings: ~a" bindings)
      (pause))

   (cond ((member (first consequent) *ife-functions*)
          (evaluate consequent bindings) nil)
         ((eval '(object-p ,(first consequent)))
          (instantiate-object consequent))
         ((eq 'assert (first consequent))
          (remember-assertion (evaluate (rest consequent) bindings))))

   (when (intersection *debug* '(fire-consequent all))
      (format t "~%< FIRE-CONSEQUENT")
      (pause)))

(defun fire-consequents (consequents bindings-list
      &aux (first-list ()) (second-list consequents) (dummy-list ())
      (switch ()) (block-firing ()) (index 0))
   ; 8 April 1992
   ;
   ; Calling Function:  USE-IF-THEN-RULE
   ; Returns:  Bindings-list
   ;
   ; This function ...

   ; (when (and (not bindings-list)
   ;            (no-bindings-variables-in-consequents consequents))
   ;    (setf bindings-list '(())))
   ;
   ; Note:  The preceeding LISP command is required to ensure that the
   ; consequents are fired if the bindings-list is NIL and the consequents
   ; do not contain any bindings-variables.  The is exemplified in the first
   ; rule of INLET-FLOW-AREA-RB.

   (when (intersection '(all fire-consequents) *debug*)
      (format t "~%> FIRE-CONSEQUENTS")
      (when (member 'pause *debug*)
         (format t "~%---- Printer ? - Return if not required ----")
         (pause))
      (format t "~%Consequents: ~a" consequents)
      (format t "~%Bindings list: ~a" bindings-list)
      (when (member 'pause *debug*) (pause)))

   (setf block-firing (match '(fire in block relative to $variable)
         (first consequents)))

   (if (<> block-firing 'fail)
      (dolist (bindings (group-bindings-wrt-???
                  (second (assoc '$variable block-firing))
                  bindings-list)
            t)
```

```
        (fire-consequents (rest consequents) bindings))
      (progn
        (setf first-list (manipulate-bindings bindings-list))
        (when (equal (first second-list)
                     '(apply bindings to each consequent))
          (setf dummy-list first-list
                first-list (rest second-list)
                second-list dummy-list
                switch t))

        (do ((first-list first-list (rest first-list))
             (second-list second-list second-list))
            ((endp first-list))
            (when (not switch)
              (incf index))

          (do ((second-list second-list (rest second-list))
               (first-item t nil) (result ()) (item ()) (position ()))
              ((endp second-list))

              (cond ((and switch first-item)
                     (setf index 1))
                    (switch  (incf index)))
              (if switch
                  (setf result (instantiate-bindings (first first-list)
                  `(,@(first second-list) ($index ,index))))
                  (setf result (instantiate-bindings (first second-list)
                  `(,@(first first-list) ($index ,index)))))

              (setf result (insert-object-values-in-consequent
                            result))
              (fire-consequent result
                (if switch (first second-list)
                  (first first-list)))))))
    (when (intersection '(all fire-consequents) *debug*)
      (if switch
        (format t "~%< FIRE-CONSEQUENTS: ~a~%" second-list)
        (format t "~%< FIRE-CONSEQUENTS: ~a~%" first-list))
      (pause))

    (if switch second-list first-list))

(defun firstn (number lst)
  (cond ((>= number (length lst)) lst)
        ((eq number (length lst)) lst)
        (t (firstn number (butlast lst)))))

(defun group-bindings-wrt-???
    (variable bindings-list &aux (value ()) (complex-grouping ()) (answer ()))
    ; 8 April 1992
    ;
    ; Calling Function:  FIRE-CONSEQUENTS
```

```
    ; Returns: Modified bindings-list ((()()()) (()()()) ... (()()()))
    ;
    ; This function takes a variable, $name, and groups the bindings with the
    ; same variable together in one block.  The complex grouping list becomes
    ; (($variable_value_1 (bindings with the value $variable_value_1))
    ;  ($variable_value_2 (bindings with the value $variable_value_2))
    ; ... ( ... ()))

    (dolist (bindings bindings-list complex-grouping)
      (setf value (second (assoc variable bindings)))
      (if (assoc value complex-grouping)
          (rplacd (assoc value complex-grouping)
            '((,@(second (assoc value complex-grouping))
              ,bindings)))
          (setf complex-grouping
            (append complex-grouping '((,value (,bindings)))))))

    (dolist (bindings complex-grouping answer)
      (setf answer (cons (second bindings) answer))))

(defun insert-object-values (phrase)
    ; 11 April 1992
    ;
    ; Calling Functions:  INSERT-OBJECT-VALUES-IN-BINDINGS-LIST,
    ; FILTER-BINDINGS,
    ; INSERT-OBJECT-VALUES-IN-CONSEQUENT
    ; Returns: PHRASE

    (cond ((and (atom phrase)
                (object-p phrase))
           (object-value (symbol-value phrase)))
          ((atom phrase) phrase)
          ((or (member (rest phrase) '((is instantiated) (is-not instantiated)
                  (is uninstantiated) (are instantiated)
                  (are-not instantiated) (are uninstantiated))
                 :test #'equal)
               (eq (first phrase) 'define)
               (and (= (length phrase) 2) (equal (first phrase) 'instantiate)
                    (boundp (second phrase))))
           phrase)
          (t (continue-to-insert-object-values phrase))))

(defun insert-object-values-in-bindings-list (bindings-list)
    ; 13 April 1992
    ;
    ; Calling Function:  FILTER-BINDINGS-LIST
    ; Returns:  bindings-list

    (mapcar #'(lambda (bindings)
                (mapcar #'(lambda (pair)
                            (evaluate-pattern (insert-object-values pair)))
                        bindings))
```

```
    bindings-list))

(defun insert-object-values-in-consequent (consequent &aux item position)
    ; 10 April 1992
    ;
    ; Calling Function:  FIRE-CONSEQUENTS
    ; Returns:  Consequent
    ;
    ; This function takes a consequent which is about to be fired and inserts
    ; the values for all of the objects within it.  However, a consequent which
    ; is instantiating or modyfying an object must not re-insert the value but
    ; must avoid that object.  This is performed using the premise that any object
    ; before the symbols '(= IS ARE INCLUDES EXCLUDES) is not replaced.
    ; Furthermore,  object-slot-manipulation-templates, ie those templates which
    ; modify the slots of an object (status prompt etc...), and ASK consequents
    ; are also left unaltered.

    (if (or (object-slot-manipulation-template consequent)
        (equal (first consequent) 'ask))
      consequent
      (progn
        (setf item (intersection '(+= -= = is are includes excludes
                    prompt defaultvalue allowedvalues
                    status preface)
                  consequent)
          position (locate (first item) consequent))

        (mparse '(,@(firstn position consequent)
            ,@(insert-object-values
              (butfirstn position consequent)))))))

(defun insert-objects-in-values (a-values &aux (r-values nil))
    ; 14 April 1992
    ;
    ; Calling Function:  PRINT-TO-SCREEN
    ;
    ; a-values -- Accepted-VALUES, r-values -- Returned-VALUES
    ; This function takes a list of symbols and checks to see if any are objects.
    ; If this is so then the object value is inserted in its place and the new
    ; list is returned.

    (when (intersection '(all sort-values) *debug*)
      (format t "~%> SORT-VALUES.")
      (format t "~%a-values: ~a~%" a-values))

    (dolist (value a-values (reverse r-values))
      (if (and (atom value)
            (eval '(object-p ,value)))
        (setq r-values (cons (object-value (symbol-value value))
              r-values))
        (setq r-values (cons value r-values)))))
```

```
(defun insert-template-values (&rest template-values &aux template values)
    ; 15 May 1992
    ;
    ; Calling Function:  MAKE-REGIONS

    (setf template (first template-values)
      values (rest template-values)
      values (insert-objects-in-values values))

    (do ((count 1 (incf count)))
      ((> count (length values)) template)
      (setf template
        (subst (nth (- count 1) values)
          '(,count)
          template
          :test #'equal))))

(defun insidep (variable expression bindings)
  (if (equal variable expression)
    nil
    (inside-or-equal-p variable expression bindings)))

(defun inside-or-equal-p (variable expression bindings &aux lst)

  (setf lst (coerce (write-to-string expression) 'list))

  (cond ((equal variable expression) t)
    ((atom expression) nil)
    ((and (> (length lst) 2)
      (eq (intern (coerce '(,(second lst)) 'string)) '$))
     (let ((binding (find-binding expression bindings)))
      (when binding
        (inside-or-equal-p variable
          (extract-bindings-value binding)
          bindings))))
    (t (or (inside-or-equal-p variable
          (first expression)
          bindings)
        (inside-or-equal-p variable
          (rest expression)
          bindings)))))

(defun instantiate-bindings (pattern bindings)
    ; 2 April 1992
    ;
    ; Calling Functions:  INSTANTIATE-BINDINGS, TRY-RULE,
    ; FILTER-BINDINGS,
    ; FIRE-CONSEQUENTS
    ; Returns:  pattern with included bindings replaced where appropriate.
    ;
    ; This function takes a pattern, i.e. 'A Template with a $Binding', and a
    ; list of bindings (()()()).  The function bindings-p indicates whether the
```

```
    ; atom is a binding by considering the first character of the symbol.  If
    ; the first character is equal to $ the appropriate binding in the list of
    ; bindings is extracted and inserted in place of the atom.  The function
    ; is RECURSIVE.

   (cond ((bindings-p pattern)
          (let ((binding (find-binding pattern bindings)))
            (if (and binding (<> (first binding) (second binding)))
                (instantiate-bindings
                   (extract-bindings-value binding) bindings)
                pattern)))
         ((atom pattern) pattern)
         (t (cons (instantiate-bindings (first pattern) bindings)
                  (instantiate-bindings (rest pattern) bindings)))))

(defun instantiate-object
    (consequent &aux (object (first consequent))
      (operator (second consequent))
      (value (car (last consequent)))
      (type (object-type (symbol-value object))))
    ; 14 April 1992
    ;
    ; Calling Function:  FIRE-CONSEQUENT

    (when (intersection '(instantiate-object all) *debug*)
      (format t "~%> INSTANTIATE-OBJECT.")
      (format t "~%Variable: ~a" object)
      (format t "~%Operator: ~a" operator)
      (format t "~%Value: ~a" value)
      (format t "~%Type: ~a" type)
      (pause))

    (when (and (not (listp value))
               (eval '(object-p ,value)))
      (setf value (object-value (symbol-value value))))

    (cond ((eq operator 'allowedvalues)
           (setf (object-allowedvalues (symbol-value object)) value)
           (when (eq (object-type (symbol-value object)) 'list)
             (setf (object-defaultvalue (symbol-value object))
                   (car value))))

          ((eq operator 'defaultvalue)
           (setf (object-defaultvalue (symbol-value object)) value))

          ((eq operator 'preface)
           (setf (object-preface (symbol-value object)) value))

          ((eq operator 'status)
           (setf (object-status (symbol-value object)) value))

          ((eq operator 'prompt)
```

```
        (setf (object-prompt (symbol-value object))
           (make-prompt value)))

       ((and (member operator '(excludes includes))
           (eq type 'list))
        (if (eq operator 'includes)
           (unless (member value
             (object-value (symbol-value object)) :test #'equal)
              (setf (object-value (symbol-value object))
              (append (object-value (symbol-value object))
                    (list value))))
           (setf (object-value (symbol-value object))
                 (remove value
              (object-value (symbol-value object)) :test #'equal))))

       ((and (member operator '(is are))
           (member type '(string text)))
        (setf (object-value (symbol-value object)) value))

       ((and (eq operator '=)
           (member type '(integer real)))
        (setf (object-value (symbol-value object))
           (mparse (evaluate value))))

       ((and (eq operator '+=)
           (member type '(integer real)))
        (setf (object-value (symbol-value object))
           (+ (object-value (symbol-value object))
             (mparse (evaluate value)))))
       (t
         (format t "~%Inconsistant type/operator for object ~a."
           object))))

(defun join (lst)
    ; 11 April 1992
    ;
    ; Calling Functions:  UPGRADE, REMEMBER-RULE, MAKE-FORM,
    ; EVALUATE-P,
    ; TRY-OBJECT-SLOTS
    ; Returns:  From (This is an example) to THISISANEXAMPLE

  (concat-symbol (mapcar #'(lambda (item) (write-to-string item)) lst)))

(defun lastn (number lst)
  (cond ((>= number (length lst)) lst)
      ((eq number (length lst)) lst)
      (t (lastn number (rest lst)))))
```

```
(defun locate (item pattern)
     ; 11 April 1992
     ;
     ; Calling Function:  INSERT-OBJECT-VALUES-IN-CONSEQUENT
     ; Returns:  Numeric value

   (do ((count 1)
        (pattern-length (length pattern))
        (pattern pattern (rest pattern)))
       ((or (endp pattern)
            (equal item (first pattern)))
        (if (endp pattern) 0 count))
       (incf count)))

(defun make-ask-form (arguments &aux (assignment nil)
        (type (car (last arguments)))
        (variable ()))
     ; 14 April 1992
     ;
     ; Calling Function:  MAKE-FORM

     (if (eval '(object-p ,(first arguments)))
       '(ask-object ',(first arguments))
       '(ask-fact ',arguments)))

(defun make-binding (bindings-variable bindings-value)
   (list bindings-variable bindings-value))

(defun make-bindings-list-from-bindings-lists (bindings-lists &aux answer)
     ; 11 April 1992
     ;
     ; Calling Function:  FILTER-BINDINGS-LIST
     ; Returns:  ((()()()) (()()()) ... (()()())) from ((()()()))
     ; (()()))
     ; ((()()()
     ; ((()()())))
     (dolist (bindings-list bindings-lists answer)
       (dolist (bindings bindings-list t)
         (setf answer (cons bindings answer)))))

(defun make-fetch-form (arguments)
     ; 14 April 1992
     ;
     ; Calling Function:  MAKE-FORM
     ; Template "(sentence (type rulebase))" OR "(sentence rulebase)"

     (if (listp (car (last arguments)))
       '(fetch ',(butlast arguments)
          :type ',(first (car (last arguments)))
          :rulebase ,(second (car (last arguments))))
       '(fetch ',(butlast arguments) :RuleBase ,(car (last arguments)))))
```

```
(defun make-for-all-rule (name rule &aux (object (third rule)))
    ; 25 April 1992
    ;
    ; Calling Function:  RE-STRUCTURE-RULE-TO-RULE-TEMPLATE
    ; Returns:  Rule

   (if (and (eq (car (fifth rule)) 'then)
            (eq (car (fourth rule)) 'if))
     (setf rule `(,name ,rule))
      (setf rule `(,name (for all ,object
              (if nothing)
              (then ,@(butfirstn 3 rule)))))))

(defun make-form (form &optional bindings-list &aux (function (first form))
       (arguments (rest form)))
    ; 14 April 1992
    ;
    ; Calling Function: EVALUATE-FUNCTION

    (cond ((eq function 'abs)
       `(abs ,@arguments))
      ((eq function 'reset)
       `(reset ,@(mapcar #'(lambda (a) `',a) arguments)))
      ((eq function 'join)
       `(join '(,@arguments)))
      ((eq function 'remove)
       `(remove-assertion '(,@arguments)))
      ((eq function 'ask)
       (make-ask-form arguments))
      ((eq function 'fetch)
       (make-fetch-form arguments))
      ((eq function 'run)
       (make-run-form arguments bindings-list))
      ((eq function '->q1)
       (make-->q1-form arguments))
      ((eq function '->Screen)
       (make-->Screen-form arguments))
      ((eq function 'symbol-split)
       `(symbol-split ,(first arguments) ',(second arguments)))
      ((member function '(xc_1 xc_2 yc_1 yc_2 zc_1 zc_2))
       `(,function ',bindings-list))
      ((eq function '->1.0e???)
       `(->1.0e??? ',@arguments))
      ((eq function 'max)
       `(max ,@arguments))
      ((eq function 'use)
       `(use-rulebase ',@arguments))
      ((eq function 'int)
       `(floor ,@arguments))
      ((equal function 'Define)
       `(set-object ,@arguments))
      ((equal function 'instantiate)
```

```
        '(fetch ',@arguments))
       (t t)))

(defun no-bindings-variables-in-consequents (consequents)
    ; 22 May 1992
    ;
    ; Calling Function:  FIRE-CONSEQUENTS
    (do ((consequents consequents (rest consequents))
         (variable-present nil))
        ((or (endp consequents) variable-present)
         (if variable-present nil t))
        (do ((consequent (first consequents) (rest consequent)))
            ((or (endp consequent) variable-present))
            (when (bindings-p (first consequent))
              (setf variable-present t)))))

(defun make-prompt (&rest template-values &aux (template nil) (values nil)
       (phrase nil) (prompt ""))
    ; 14 April 1992
    ;
    ; Calling Functions:
    ; Returns:  A prompt in the form of a string.
    ;
    ; This functions receives a list '((template) val1 val2 ... valn).  Initially
    ; all of the values in the template given by (number) are replaced.  This
    ; list is then converted to a list of strings and concatenated together to
    ; form the prompt which is returned.

   (setq template (first template-values)
      values (rest template-values))

    (setf phrase (dolist (value values template)
          (setf template (substitute value
                '(,(locate value values))
                template :test #'equal))))

    (make-prompt-string phrase))

(defun make-prompt-string (phrase)
    ; 16 April 1992
    ;
    ; Calling Function:  MAKE-PROMPT

    (do ((phrase phrase (rest phrase))
         (word ())
         (scrn-wdth 50)
         (line-length 0)
         (count 1 (incf count))
         (newline () ())
          (string "")
         (last-word "."))
        ((endp phrase) (concatenate 'string string " "))
```

```
        (setf word (string-downcase (write-to-string (first phrase))))

        (when (eq #\| (first (coerce word 'list)))
          (setf word (coerce (rest (butlast (coerce word 'list)))
                             'string)))

        (when (and  (not (equal word "<nl>"))
                    (equal #\< (first (coerce word 'list)))
                    (equal #\> (car (last (coerce word 'list)))))
          (setf word (string-upcase
                      (coerce (rest (butlast (coerce word 'list))) 'string))))

        (when (eq #\. (car (last (coerce last-word 'list))))
              (setf word (string-capitalize word)))

        (when (>= (+ line-length  (length word) 2) scrn-wdth)
          (setf string (concatenate 'string string " ~%")
                newline t
                line-length 0))

        (cond ((eq #\. (car (last (coerce last-word 'list))))
               (setf string (concatenate 'string string
                 (if (= count 1) "" "  ") word)
                     last-word word
                     line-length (+ line-length (length word)
                    (if (= count 1) 0 2))))
              ((or (equal "~%" last-word) (equal "<nl>" last-word))
               (setf string (concatenate 'string string word)
                     last-word word
                     line-length (+ line-length (length word))))
              ((or (equal "~%" word) (equal "<nl>" word))
               (setf string (concatenate 'string string " ~%")
                     line-length 0))
              (t
               (setf string (concatenate 'string string
                     (if newline "" " ") word)
                     last-word word
                     line-length (+ line-length (length word) 1))))))

(defun make-rule-name (&optional name)
    ; 25 April 1992
    ;
    ; Calling Function:  RE-STRUCTURE-RULE-TO-RULE-TEMPLATE
    ; Returns:  symbol

    (incf *rule-count*)
  (if name
      (setf name (join '(,@(mapcar #'(lambda (item)
                    (eval '(join '(,item -))))
                 name)
            rule- ,*rule-count*)))
      (setf name (join '(rule - ,*rule-count*)))))
```

```
(defun make-run-form (arguments bindings-list)
    ; 14 April 1992
    ;
    ; Caling Function:  MAKE-FORM

    (if (eq (second arguments) 'bindings)
      `(,(first arguments) ',bindings-list ,@(rest (rest arguments)))
      `(,@arguments)))

(defun make-unique-local-required-bindings
    (local-required-bindings current-bindings &aux answer tail)
    ; 27 April 1992
    ;
    ; Calling Function:  MODIFY-BINDINGS-TO-INCLUDE-ORIGINAL-AND-
    ; REQUIRED-BINDINGS

    (dolist (pair local-required-bindings answer)

      (if (bindings-manipulation-template (second pair))
        (setf tail (second (assoc (first pair) current-bindings)))
        (setf tail (mparse
          (evaluate
            (insert-object-values
              (instantiate-bindings
                (second pair) current-bindings))
              current-bindings))))
      (when (listp tail) (setf tail (mparse tail)))

      (setf answer (cons `(,(first pair) ,tail) answer)))
    answer)

(defun modify-bindings-to-include-original-and-required-bindings
    (required-bindings current-bindings-list original-bindings
      &aux answer)
    ; 3 April 1992
    ;
    ; Calling Function:  TRY-RULE
    ; Returns: ((()()()) (()()()) (()()()))
    ;
    ; The required bindings ALWAYS consists of two bindings-p variables, or one
    ; bindings-p variable and a bindings-manipulation template.  The first is
    ; always the variable in the antecedent, and the second, where applicable,
    ; is the variable within the body of the rule.
    ;
    ; This function takes the current-bindings and removes all of the bindings
    ; that are not either in the required-bindings list or the original
    ; bindings.  Furthermore, the second argument in the required-bindings, ie
    ; the variable within the body of the rule, or the manipulations template,
    ; is replaced with the corresponding value from the current-bindings-list.

    (when (member 'mbtioarb *debug*)
      (format t "~%>
```

```
MODIFY-BINDINGS-TO-INCLUDE-ORIGINAL-AND-REQUIRED-BINDINGS")
        (format t "~%Required-bindings: ~a" required-bindings)
        (format t "~%Current-bindings-list: ~a" current-bindings-list)
        (format t "~%Original-bindings: ~a" original-bindings)
        (pause))

      (setf current-bindings-list (manipulate-current-bindings-list
          required-bindings current-bindings-list))
      (setf answer
        (remove-duplicates
          (mapcar #'(lambda (current-bindings &aux
(local-required-bindings   required-bindings))
              (setf local-required-bindings
                (make-unique-local-required-bindings
                  required-bindings current-bindings))
                `(,@original-bindings ,@local-required-bindings))
              current-bindings-list)
          :test #'equal))

      (when (member 'mbtioarb *debug*)
        (format t "~%<
MODIFY-BINDINGS-TO-INCLUDE-ORIGINAL-AND-REQUIRED-BINDINGS")
        (format t "~%Answer: ~a" answer)
        (pause))

      answer)

(defun make-->q1-form (arguments &aux (template (first arguments))
        (arguments (rest arguments))
        (form `(->q1 ',template)))

    (do ((arguments arguments (rest arguments))
        (argument ()))
       ((endp arguments) form)
       (setf argument (first arguments))
       (if (symbolp argument)
          (setf form (append form `(',argument)))
          (setf form (append form `(,argument))))))

(defun make-->Screen-form (arguments)
      ; 14 April 1992
      ;
      ; Calling Function: MAKE-FORM

      `(print-to-screen ',arguments))

(defun manipulate-bindings (bindings-list &aux variable answer)
      ; 2 June 1992
      ;
      ; Calling Functions:  FIRE-CONSEQUENTS,
      ;  MANIPULATE-CURRENT-BINDINGS-LIST
      ; Returns:  bindings-list
```

```
;
; G11-RB contains a rule that requires the IFE to obtain an initial value
; for a dependent variable. The rule fire writes to the data file the
; PHOENICS FIINIT command. For this rule to be fired the system backward
; chains on the rules in G11-RB to asscertain the $initial-value. One of the
; rules has a consequent INITIAL VALUE FOR $VALUE = (AVERAGE
; AXIAL-VELOCITY
; FROM BINDINGS). Now, the antecedents to this rule gather from the
; assertions all of the velocity (U1, V1 or W1) axial-inlet values and
; stores these in the bindings-list. In order to to obtain the average, as
; required by the consequent, the IFE must be requested to do so from within
; the rules. The

(do ((success ())
   (bindingslist bindings-list (rest bindingslist)))
  ((or success (endp bindingslist))
   (if success (manipulate-bindings bindings-list)
      (remove-duplicates bindings-list :test #'equal)))

  (do ((bindings (first bindingslist) (rest bindings))
       (value ()))
      ((or success (endp bindings)))

      (setf value (second (first bindings)))

      (cond ((<> 'fail (unify value '(average $variable from bindings)))
          (setf answer
             (unify value '(average $variable from bindings))
             variable
             (intern
                (concatenate 'string "$"
                 (write-to-string (second (first answer)))))
             bindings-list
             (average-variable-from-bindings
                variable bindings-list)
             success t))
         ((<> 'fail (unify value '(sum $variable from bindings)))
          (setf answer
             (unify value '(sum $variable from bindings))
             variable
             (intern
                 (concatenate 'string "$"
                 (write-to-string (second (first answer)))))
             bindings-list
             (sum-variable-from-bindings
                variable bindings-list)
             success t))))))
```

```
(defun manipulate-current-bindings-list
    (required-bindings current-bindings-list &aux (answer ()))
    ; 10 April 1992
    ;
    ; Calling Function:  MODIFY-BINDINGS-TO-INCLUDE-ORIGINAL-AND-
    ; REQUIRED-BINDINGS
    ; Returns:  Modified CURRENT-BINDINGS-LIST
    ;
    ; This function recieves a list of required bindings and the current bindings
    ; list from the above function.  It steps through the required-bindings and
    ; checks to see if the second argument of the pair is a
    ; manipulations-template.  If it is then the pair is CONSed to the entire
    ; set of current-bindings and then sent to the function
    ; MANIPULATE-BINDINGS.
    ; The current-bindings are then updated and the procedure repeated until all
    ; of the required-bindings have been checked.

    (dolist (pair required-bindings current-bindings-list)
      (when (bindings-manipulation-template (second pair))
        (setf answer ())

        (dolist (current-bindings current-bindings-list t)
          (setf answer (cons (cons pair current-bindings) answer)))

        (setf current-bindings-list (manipulate-bindings answer)))))

(defun manipulation-template-? (phrase)
    ; 11 April 1992
    ;
    ; Calling Functions:  BINDINGS-MANIPULATION-TEMPLATE,
    ;                     OBJECT-SLOT-MANIPULATION-TEMPLATE
    ; Returnd: T or NIL

    (do ((manipulation-templates
          *manipulation-templates* (rest manipulation-templates))
     (success-switch ()))
      ((or (endp manipulation-templates) success-switch) success-switch)
      (when (<> 'fail (match (first manipulation-templates) phrase))
        (setf success-switch t))))

(defun match (p d &optional bindings)
  (cond ((bindings-p p)
         (match-variable p d bindings))
        ((elements-p p d)
         (match-atoms p d bindings))
        ((recursive-p p d)
         (match-pieces p d bindings))
        (t 'fail)))
```

```
(defun match-antecedent-to-assertions (antecedent bindings)
    ; 2 April 1992
    ;
    ; Calling Function:
    ; MATCH-ANTECEDENT-TO-ASSERTIONS-AND-ASSOCIATED-RULES
    ; Returns: () or ((()()()) (()()) ... (()()))
    ;
    ; This matches the antecedent that has already had its objects
    ; and the current bindings inserted into it, with the assertions list.  The
    ; value returned from this procedure is the resulting bindings in a bindings
    ; list form.

    (if (member (car (last antecedent)) '(instantiated uninstantiated))
      (determine-whether-the-antecedent-is-instantiated-or-not
        antecedent bindings)
     (do ((answer ())
          (*assertions* *assertions* (rest *assertions*)))
         ((or (endp *assertions*) answer) answer)
         (setf answer (try-assertions antecedent (first *assertions*)
               bindings)))))

(defun match-antecedent-to-assertions-and-associated-rules
    (antecedent bindings associated-rules rulebase-name
      &aux (variables ()) (assertions-answer ())
      (associated-rules-answer ()) (answer ()))
    ; 3 April 1992
    ;
    ; Calling Function:  EVALUATE-ANTECEDENT
    ; Returns: () or ((()()()) (()()) ... (()()))
    ;
    ; This function takes the antecedent and initially matches it with the
    ; assertions.  If there are no assertions that have been matched then the
    ; antecedent is matched with the associated rules.  The function returns
    ; a bindings-list that originates from the received bindings.

    (when (intersection *debug* '(mataaar all))
      (format t "~%>MATCH ANTECEDENT TO ASSERTIONS AND
ASSOCIATED RULES.")
      (format t "~%Antecedent: ~a" antecedent)
      (format t "~%Bindings: ~a" bindings)
      (format t "~%Associated rules: ~a" associated-rules)
      (pause))

    (setf assertions-answer
      (match-antecedent-to-assertions antecedent bindings))

    (when (intersection *debug* '(mataaar all))
      (format t "~%Matched antecedent with assertions.")
      (format t "~%Assertions-answer: ~a" assertions-answer)
      (pause))

    (when (not assertions-answer)
```

```
      (setf associated-rules-answer
        (match-antecedent-to-associated-rules antecedent bindings
          associated-rules rulebase-name)))

    (when (intersection *debug* '(mataaar all))
      (format t "~%Matched antecedent with associated rules.")
      (format t "~%Associated-rules-answer: ~a" associated-rules-answer)
      (pause))

    (setf answer '(,@assertions-answer
      ,@associated-rules-answer))

    (when (intersection *debug* '(mataaar all))
      (format t "~%<MATCH ANTECEDENT TO ASSERTIONS AND
ASSOCIATED RULES.")
      (format t "~%Answer: ~a~%" answer)
      (pause))

    answer)

(defun match-antecedent-to-associated-rules
    (antecedent bindings associated-rules rulebase-name &aux (answer ()))
    ; 2 April 1992
    ;
    ; Calling Function:  MATCH-ANTECEDENT-TO-ASSERTIONS-AND-
    ; ASSOCIATED-RULES
    ; Returns: () or ((()()()) (()()()) ... (()()()))
    ;
    ; The function steps through the associated rules for the particular rule that
    ; the antecedent is a child of.  For each associated rule the function
    ; TRY_RULE is called and should return a bindings list.  As soon as a answer
    ; is obtained the searching stops.

   (when (intersection '(all match-antecedent-to-associated-rules) *debug*)
     (format t "~%> MATCH-ANTECEDENT-TO-ASSOCIATED-RULES.")
     (format t "~%Antecedent: ~a" antecedent)
     (format t "~%Bindings: ~a" bindings)
     (format t "~%Associated-rules: ~a~%" associated-rules)
     (pause))

    (do ((associated-rules associated-rules (rest associated-rules))
      (rule ()) (secondary-associated-rules ()))
      ((or (endp associated-rules) answer))

      (setf rule (nth (- (first (first associated-rules)) 1)
            (second (symbol-value rulebase-name)))
        secondary-associated-rules (second (first associated-rules))
        answer (try-rule antecedent rule bindings
          secondary-associated-rules rulebase-name)))

    (when (intersection '(all match-antecedent-to-associated-rules) *debug*)
      (format t "~%Answer: ~a" answer)
```

```
        (pause))

      answer)

(defun match-atoms (p d bindings)
   (if (eql p d) bindings 'fail))

(defun match-pieces (p d bindings)
   (let ((result (match (first p) (first d) bindings)))
      (if (eq 'fail result)
         'fail
         (match (rest p) (rest d) result))))

(defun match-variable (p d bindings)
   (let ((binding (find-binding p bindings)))
      (if binding
         (match (extract-bindings-value binding) d bindings)
         (add-binding p d bindings))))

(defun menu (object
      &aux (AV (object-allowedvalues (symbol-value object)))
      (units (object-units (symbol-value object)))
      (prompt
         (concatenate
            'string
            (object-prompt (symbol-value object))
            "~%(Enter 1 - "
            (write-to-string (length AV))
            ") : "))
      (reply nil)
      (DV (object-defaultvalue (symbol-value object)))
      (DAV (object-disallowedvalues (symbol-value object))))
    ; 14 April 1992
    ;
    ; Calling Function:  ASK-OBJECT

    (PrintPreface object)

   (do ((count 1 (1+ count)))
      ((> count (length AV)))
      (format t "~%~a: ~a" count (nth (1- count) AV)))

   (if DV (format t "~%~%Default value : ~a~%" DV)
      (format t "~%"))

   (format t "~%")
     (format t prompt)
     (when units (format t "[ ~a ] " units))

   (setq reply (first (read-sentence)))

   (cond ((and (not reply) DV)
```

```
        DV)
       ((intersection (list reply) '(why w)) (why)
        (menu object))
       ((intersection (list reply) '(h help))
        (help object)
        (menu object))
       ((and (not (integerp reply))
          (member reply AV :test #'equal))
        reply)
       ((and (not AV)
          (not (integerp reply))
          DAV)
        (if (member reply DAV :test #'equal)
          (progn
            (format t "~%Sorry, the answer you gave is a DISALLOWEDVALUE.
Current disallowed values are ...")
            (dolist (value DAV t)
              (print value))
            (menu object))
          reply))
       ((and (not (integerp reply))
          (not (member reply AV)))
        (format t "~%You must enter an integer between 1 and ~a, or type an allowed
value.~%"
          (length av))
        (menu object))
       ((or (< reply 1)
          (> reply (length av)))
        (format t "~%You must enter an integer between 1 and ~a, or type an allowed
value.~%"
          (length av))
        (menu object))
       (t
         (nth (1- reply) AV))))

(defun modify-bindings-list-to-include-$values
    (values &optional (bindings-list ()) &aux (answer ()))
    ; 7 April 1992
    ;
    ; Calling Function:  USE-FOR-ALL-RULE
    ; Returns:  (((()())) ((()())) ... ((()())))
    ;
    ; This function adds to each binding within the bindings-list each of the
    ; $values associated with the list object used in USE-FOR-ALL-RULE.  If the
    ; was a bindings-list of 4 elements and a values list of 3 elements then the
    ; resulting bindings-list would consist of 12 elements.  That is the size of
    ; the returning list is the multiple of the number of elements in the values
    ; and bindings-list.

    (if bindings-list
      (dolist (bindings bindings-list answer)
      (dolist (value values t)
```

```
            (setf answer '(,@answer (,@bindings ($value ,value))))))
      (dolist (value values answer)
            (setf answer '(,@answer (($value ,value) ,@bindings-list))))))

(defun mparse (expression &aux (operators '([] * / * ^ expt - + - ()))
      (op1 ()) (operator1 ())
      (op2 ()) (operator2 ()) (result 0))

   (if (or (listp expression) (numberp expression))
      (progn
         (when (numberp expression) (setf expression (list expression)))

         (setf op1 (first expression)
            operator1 (second expression)
            op2 (third expression)
            operator2 (fourth expression))

         (cond ((eq operator1 '^)
             (setf operator1 'expt))
            ((eq operator2 '^)
             (setf operator2 'expt))
            (t t))

         (cond ((endp expression) nil)
            ((and op1 (listp op1))
             (setf result (mparse op1))
             (if (listp result)
                (cons result (mparse (rest expression)))
                (mparse (cons result (rest expression)))))
            ((or (not (numberp op1))
                (not (member operator1 operators)))
             (setf result (mparse (rest expression)))
             (if (listp result)
                (cons op1 result)
                (cons op1 '(,result))))
            ((and (numberp op1)
                (not operator1)) op1)
            ((and op2 (listp op2))
             (setf result (mparse op2))
             (mparse '(,op1 ,operator1 ,result ,@(cdddr expression))))
            ((or (member operator2 (member operator1 operators))
                (not (member operator2 (member operator1 operators))))
             (setf result (eval '(,operator1 ,op1 ,op2)))
             (mparse (cons result (cdddr expression))))
            (t
               (eval '(,operator1 ,op1 ,(mparse (cddr expression)))))))
      expression))
```

```
(defun multiple-argument-command (arguments &aux (command ()) (string ""))
    ; 14 April 1992
    ;
    ; Calling Function:  ->Q1

  (setf command (first arguments)
      arguments (rest arguments)
      string (concatenate 'string
         (string-capitalize (write-to-string command))
         "("))

  (when (eval '(object-p ,(first arguments)))
    (setf arguments '(,(object-value
          (symbol-value (first arguments))))))

  (do ((arguments arguments (rest arguments))
      (argument ()))
     ((endp arguments) string)
     (setf argument (first arguments))
     (if (= (length arguments) 1)
        (setf string (concatenate 'string
              string
              (if (stringp argument)
                 argument
                 (write-to-string argument))
              ")"))
        (setf string (concatenate 'string
              string
              (if (stringp argument)
                 argument
                 (write-to-string argument))
              ","))))))

(defun object-slot-manipulation-template (phrase)
    ; 11 April 1992
    ;
    ; Calling Function:  INSERT-OBJECT-VALUES-IN-CONSEQUENT
    ; Returns:  T or NIL

    (manipulation-template-? phrase))

(defun pause (&aux answer)
    ; 11 April 1992
    ;
    ; Calling Functions:  All functions with diagnostic requirements.

    (when (member 'pause *debug*)
      (format t "~%--- Pause ---")
      (format t "~%Enter a command or RETURN to continue ... ")
      (setf answer (first (read-sentence)))
      (if (not answer)
         (format t "OK - Continuing ...~%")
```

```
        (progn
          (when (eval '(object-p ,answer))
            (setf answer '(print ,answer)))
          (eval answer)
          (terpri)
          (pause)))))

(defun preliminary-evaluation-of-the-antecedent (antecedent)
  ; 3 April 1992
  ;
  ; Calling function:  EVALUATE-ANTECEDENT
  ; Returns:  T, NIL, or antecedent
  ;
  ; The preliminary-evaluation-of-the-antecedent checks for a list of three
  ; elements, ie (Arg1 operator Arg2), and then evaluates the condition.  The
  ; function returns T or NIL for an evaluated antecedent.  Alternatively, the
  ; orginal list is returned if the length is greater than three elements.
  ; This allows for the antecedent to be a template under which the function
  ; MATCH-ANTECEDENT-TO-ASSERTIONS-AND-ASSOCIATED-RULES
  ; needs to be called.

  (cond ((> (length antecedent) 3)
         antecedent)
        ((and (= (length antecedent) 3)
              (bindings-p-in-antecedent antecedent))
         antecedent)
        ((equal (second antecedent) 'includes)
         (if (eval '(member ',(third antecedent) ',(first antecedent)
                    :test #'equal))
             t nil))
        ((equal (second antecedent) 'excludes)
         (if (eval '(not (member ',(third antecedent)
                                 ',(first antecedent)
                                 :test #'equal)))
             t nil))
        ((equal (second antecedent) 'overlaps)
         (if (eval '(intersection ',(third antecedent)
                                  ',(first antecedent)
                                  :test #'equal))
             t nil))

        ((member (second antecedent) '(is are))
         (cond ((equal (third antecedent) 'instantiated)
                (if (object-value (symbol-value (first antecedent)))
                    t nil))
               ((equal (third antecedent) 'uninstantiated)
                (if (object-value (symbol-value (first antecedent)))
                    nil t))
               (t (eval '(equal ',(first antecedent)
                                ',(third antecedent))))))

        ((equal (second antecedent) '=)
```

```
        (cond ((equal (third antecedent) 'instantiated)
               (if (object-value (symbol-value (first antecedent)))
                   t nil))
              ((equal (third antecedent) 'uninstantiated)
               (if (object-value (symbol-value (first antecedent)))
                   nil t))
              (t (eval `(= ,(first antecedent)
                           ,(third antecedent))))))

       ((member (second antecedent) '(is-not are-not))
        (cond ((equal (third antecedent) 'instantiated)
               (if (object-value (symbol-value (first antecedent)))
                   nil t))
              ((equal (third antecedent) 'uninstantiated)
               (if (object-value (symbol-value (first antecedent)))
                   t nil))
              (t (eval `(not (equal ,(first antecedent)
                                    ,(third antecedent)))))))

       ((equal (second antecedent) '<>)
        (cond ((equal (third antecedent) 'instantiated)
               (if (object-value (symbol-value (first antecedent)))
                   nil t))
              ((equal (third antecedent) 'uninstantiated)
               (if (object-value (symbol-value (first antecedent)))
                   t nil))
              (t (eval `(not (= ,(first antecedent)
                                ,(third antecedent)))))))

       ((equal (second antecedent) '>=)
        (eval `(>= ,(first antecedent) ,(third antecedent))))
       ((equal (second antecedent) '<=)
        (eval `(<= ,(first antecedent) ,(third antecedent))))
       ((equal (second antecedent) '>)
        (eval `(> ,(first antecedent) ,(third antecedent))))
       ((equal (second antecedent) '<)
        (eval `(< ,(first antecedent) ,(third antecedent))))
       (t antecedent)))

(defun PrintPreface (object)
    (when (object-preface (symbol-value object))
      (print-to-screen (object-preface (symbol-value object)))))

(defun prepare-message (&rest template-values
       &aux (template (first template-values))
       (values (rest template-values))
       (phrase nil)
       (scrn-wdth 60)
       (line "")
       (string "  ")
       (line-length 0)
       (scrn-text nil))
```

```
; 14 July 1993
;
; Calling Function:
; Returns a string value formated to be printed

(when (intersection '(print-to-screen all) *debug*)
  (format t "~%> PRINT-MESSAGE.")
  (format t "~%Template-Values: ~a" template-values)
  (format t "~%Template: ~a" template)
  (format t "~%Values: ~a" values)
  (pause))

(when (listp (first template))
  (when (symbolp (first (first template)))
    (setq values (rest template)
      template (first template))))

(setq values (insert-objects-in-values values))

(do ((template template (rest template)))
  ((endp template) (setf phrase (reverse phrase)))
  (if (and (listp (first template))
       (= (length (first template)) 1)
       (numberp (car (first template))))
    (setq phrase (cons (nth (1- (car (first template))) values)
        phrase))
    (setq phrase (cons (first template) phrase))))

(do* ((phrase phrase (rest phrase))
   (word "")
   (line-length 0)
   (last-word ".")  ;This is to force the first word to be capitalized
   (coerced-word ())
   (line "")
   (string "  "))
  ((endp phrase) string)

  (if (stringp (first phrase))
    (setf word (string-downcase (first phrase)))
    (setf word (string-downcase (write-to-string (first phrase)))))

  (setf coerced-word (coerce word 'list))

  (when (eq #\. (car (last (butlast coerced-word))))
    (setq word
      (coerce (rest (butlast coerced-word)) 'string)))
  (when (eq #\. (car (last (coerce last-word 'list))))
    (setq word (string-capitalize word)))

  (cond ((or (equal (string-upcase word) "<NL>")
       (equal (string-upcase word) "<NL>."))
     (setf string (concatenate 'string string "~%  ")
```

```
            line-length 0))

          ((eq #\^  (second coerced-word))
           (setf string (concatenate 'string string " "
              (string-upcase
               (coerce (rest (rest (butlast coerced-word))) 'string)))))

          ((>= (+ line-length (length word)) (- scrn-wdth 2))
           (setf string (concatenate 'string string "~%   " word)
              line-length 0))
          ((<= (+ line-length (length word)) (- scrn-wdth 2))
           (setq string (concatenate 'string string " " word))
           (if (eq #\. (car (last coerced-word)))
              (setq line-length (+ line-length (length word) 2))
              (setq line-length (+ line-length (length word) 1))))
          (t
            (setq line-length (length word))))
        (setq last-word word)))

(defun print-to-screen (&rest template-values
        &aux (template (first template-values))
        (values (rest template-values))
        (phrase nil)
        (scrn-wdth 75)
        (line nil)
        (line-length 0)
        (scrn-text nil))
      ; 14 April 1992
      ;
      ; Calling Function:  ASK-OBJECT

      (when (intersection '(print-to-screen all) *debug*)
        (format t "~%> PRINT-TO-SCREEN.")
        (format t "~%Template-Values: ~a" template-values)
        (format t "~%Template: ~a" template)
        (format t "~%Values: ~a" values)
        (pause))

      (when (listp (first template))
        (when (symbolp (first (first template)))
          (setq values (rest template)
            template (first template))))

      (setq values (insert-objects-in-values values))

      (do ((template template (rest template)))
        ((endp template) (setf phrase (reverse phrase)))
        (if (and (listp (first template))
              (= (length (first template)) 1)
              (numberp (car (first template))))
          (setq phrase (cons (nth (1- (car (first template))) values)
                phrase))
```

```
        (setq phrase (cons (first template) phrase))))

    (do* ((phrase phrase (rest phrase))
       (word (string-downcase (write-to-string (first phrase)))
          (string-downcase (write-to-string (first phrase))))
       (line-length 0)
       (last-word ".")  ;This is to force the first word to be capitalized
       (coerced-word (coerce word 'list) (coerce word 'list))
       (line nil))
      ((endp phrase) (if (eq (last (last scrn-text)) (first line))
          scrn-text
          (write-line-to-screen line))
       (format t "~%"))

      (when (eq #\. (car (last (butlast coerced-word))))
         (setq word
            (coerce (rest (butlast coerced-word)) 'string)))
      (when (eq #\. (car (last (coerce last-word 'list))))
         (setq word (string-capitalize word)))

      (cond ((or (equal (string-upcase word) "<NL>")
              (equal (string-upcase word) "<NL>."))
          (write-line-to-screen line)
          (setq line () line-length 0))
         ((<= (+ line-length (length word)) (- scrn-wdth 2))
          (setq line (append line (list word)))
          (if (eq #\. (car (last coerced-word)))
             (setq line-length (+ line-length (length word) 2))
             (setq line-length (+ line-length (length word) 1))))
         (t
           (write-line-to-screen line)
           (setq line (list word))
           (setq line-length (length word))))
      (setq last-word word)))

(defun re-structure-rule-to-rule-template (rule &aux name)
    ; 25 April 1992
    ;
    ; Calling MACRO:  REMEMBER-RULE
    ; Returns:  Rule

    (when (atom (first rule))
      (setf rule (cons rule ())))

    (if (equal (first (first rule)) 'Rule_)
      (setf name (make-rule-name (rest (first rule)))
         rule (rest rule))
      (setf name (make-rule-name)))

    (cond ((or (member (first (first rule))
             '(use ask run ->ql define ->screen instantiate))
          (eval `(object-p ,(first (first rule)))))
```

```
        '(,name (if nothing) (then ,@rule)))
       ((equal (first (first rule)) 'for)
        (make-for-all-rule name (first rule)))
       (t '(,name ,@rule))))

(defun read-sentence (&optional (prompt ""))
    ; 16 April 1992
    ;
    ; Calling Functions:  Multiple

  (when (listen) (read-line))
  (format t prompt)
  (with-input-from-string
     (input (read-line))
     (do ((word (read input nil)
                (read input nil))
          (sentence nil))
         ((not word) (return (reverse sentence)))

         (if (symbolp word)
            (progn
               (setf word (string-upcase (write-to-string word))
                  word (coerce word 'list))

               (when (intersection '(#\. #\? #\!) word)
                  (setf word (rest (butlast word))))

               (setf word (remove #\. word)
                  word (remove #\? word)
                  word (remove #\! word)
                  word (coerce word 'string)
                  word (intern word))
               (push word sentence))
            (push word sentence)))))

(defun recursive-p (p d)
  (and (listp p) (listp d)))

(defun remember-assertion (assertion)
    ; 13 April 1992
    ;
    ; Calling Function:  FIRE-CONSEQUENT

    (do ((head ())
       (assertions '(this is to initially force a list))
       (template ())
       (tail *assertions* (rest tail))
       (inserted ()))
      ((or (endp assertions) inserted)
       (when inserted (setf *assertions* '(,@head ,@tail))))
      (setf assertions (first tail)
         template (first assertions))
```

```
        (when (and (<> (unify template assertion) 'fail)
                   (not (member assertion (rest assertions) :test #'equal)))
          (setf assertions '(,@assertions ,assertion)
            inserted t))

        (setf head '(,@head ,assertions))))

(defun remove-assertion (assertion &optional (assertions *assertions*))
    ; 11 April 1992
    ;
    ; Calling Function:  MAKE-FORM

    (mapcar #'(lambda (assertions)
                (delete assertion assertions :test #'equal))
      *assertions*))

(defun remove-variable-pair-from-bindings-list
    (variable bindings-list &aux answer)
    ; 2 June 1992
    ;
    ; Calling Function:  MANIPULATE-BINDINGS

    (dolist (bindings bindings-list answer)
      (setf answer (cons (delete (assoc variable bindings) bindings)
                         answer))))

(defun reset (&rest objects)
  (if objects
      (dolist (object objects t)
        (eval '(set-object ,object ,@(second (assoc object *objects*)))))
      (progn
        (setf q1 ())
        (setf *boundaries* ())
        (setf *regions* '((x ()) (y ()) (z ())))
        (setf *nodes* nil)
        (setf *assertions* (reset-assertions))
        (load "object.lsp"))))

(defun restore ()
  (setf *nodes* () *boundaries* () *regions* () *assertions* ())
  (load "object.lsp")
  (load "store.lsp"))


(defun rule-antecedents (rule)
    (cond ((equal (first (second rule)) 'for)
           (if (equal (first (fourth (second rule))) 'if)
               (rest (fourth (second rule)))
               ()))
          (t (rest (second rule)))))
```

```
(defun rule-consequents (rule)
     (cond ((equal (first (second rule)) 'for)
        (if (equal (first (fourth (second rule))) 'if)
           (rest (fifth (second rule)))
           (butfirstn 3 (second rule))))
       (t (rest (third rule)))))

(defun rule-name (rule) (first rule))

(defun save-assertions ()
   (with-open-file (file-stream "assert.lsp" :direction :output
         :if-exists :over-write)
      (format file-stream "(setf *assertions* '~s)" *assertions*)))

(defun single-argument-command (arguments &aux (command (first arguments))
        (variable (second arguments))
        (value (third arguments))
        (string ""))
     ; 14 April 1992
     ;
     ; Calling Function:  ->Q1

   (when (eval '(object-p ,value))
      (setf value (object-value (symbol-value value))))

   (concatenate 'string (string-capitalize (write-to-string command))
       "(" (write-to-string variable) ")="
       (if (stringp value) value
          (write-to-string value))))

(defun store (&aux value key lst)
   (with-open-file (stream "store.lsp" :direction :output
         :if-exists :rename-and-delete)
      (format stream "~%(setf *nodes* '~a)" *nodes*)
      (format stream "~%(setf *boundaries* '~a)" *boundaries*)
      (format stream "~%(setf *regions* '~a)" *regions*)
      (format stream "~%(setf *assertions* '~a)" *assertions*)
      (dolist (object *objects* t)
         (format stream "~%(set-object ~s" (first object))
         (do ((lst (second object))
            (key ())
            (value ()))
           ((endp lst))
           (setf key (first lst)
              value (second lst)
              lst (rest (rest lst)))
           (format stream "~% ~s ~s" key value))
         (format stream ")"))))
```

```
(defun sum-variable-from-bindings (variable bindings-list &aux (sum 0) answer)
    ; 2 June 1992
    ;
    ; Calling Function:  MANIPULATE-BINDINGS

    (when (member 'sum-variable-from-bindings *debug*)
      (format t "~%>SUM-VARIABLE-FROM-BINDINGS.")
      (format t "~%Variable: ~a" variable)
      (format t "~%Bindings: ~a" Bindings-list)
      (pause))

    (do ((bindings bindings-list (rest bindings)))
      ((endp bindings))

      (setf sum (+ sum (second (assoc variable (first bindings))))))

    (setf bindings-list
      (remove-variable-pair-from-bindings-list
        variable bindings-list)
      variable
      (intern (coerce (cddr (butlast (coerce (write-to-string variable)
                    'list)))
          'string)))

    (setf answer (subst sum '(sum ,variable from bindings)
        bindings-list :test #'equal))

    (when (member 'sum-variable-from-bindings *debug*)
      (format t "~%<SUM-VARIABLE-FROM-BINDINGS.")
      (format t "~%Answer: ~a" answer)
      (pause))

    answer)

(defun symbol-split (number symbol &aux (string (write-to-string symbol))
      (lst (coerce string 'list))
      (counter 0) (total (length string)))
    ; 14 April 1992
    ;
    ; Calling Function:  MAKE-FORM

  (do ((result ())
      (lst lst (rest lst))
      (counter 0 (incf counter)))
    ((= counter number) (intern (coerce result 'string)))
    (setf result (append result (list (first lst))))))

(defun try-assertions (antecedent assertions bindings &aux template answer)
    ; 11 April 1992
    ;
    ; Calling Function:  MATCH-ANTECEDENT-TO-ASSERTIONS
    ; Returns:  Bindings-list
```

```
    (when (intersection '(all try-assertions) *debug*)
       (format t "~%> TRY-ASSERTIONS.")
       (format t "~%Antecedent: ~a" antecedent)
       (format t "~%Assertions: ~a" assertions)
       (format t "~%Bindings: ~a~%" bindings)
        (pause))

      (setf template (first assertions)
        assertions (rest assertions))

      (when (<> (unify antecedent template) 'fail)
        (setf answer
           (remove 'fail (mapcar #'(lambda (assertion)
                    (match antecedent assertion bindings))
                 assertions))))

    (when (intersection '(all try-assertions) *debug*)
       (format t "~%< TRY-ASSERTIONS.")
       (format t "~%Answer: ~a" answer)
       (pause))

      answer)

(defun try-object-slots (object)
     ; 11 April 1992
     ;
     ; Calling Functions:  CONTINUE-TO-INSERT-OBJECT-VALUES, FETCH

    (cond ((object-fixedvalue (symbol-value object))
         (setf (object-value (symbol-value object))
            (object-fixedvalue (symbol-value object))))
        ((object-computevalue (symbol-value object))
         (setf (object-value (symbol-value object))
            (eval (object-computevalue (symbol-value object)))))
        ((object-rulebase (symbol-value object))
         (setf (object-rulebase (symbol-value object)) nil)
         (eval '(use-rulebase ',(join (list object '-RB))))
         (setf (object-rulebase (symbol-value object)) t))
        ((equal (object-prompt (symbol-value object)) 'never) t)
        (t nil)))

(defun try-rule (antecedent rule bindings associated-rules rulebase-name
        &aux (required-bindings ()) (answer ())
        (unification-bindings ()))
     ; 3 April 1992
     ;
     ; Calling Function:  MATCH-ANTECEDENT-TO-ASSOCIATED-RULES
     ; Returns:  () or ((()()()) (()()()) ... (()()()))
     ;
     ; This function effectively introduces the backward chaining mechanism into
     ; the inference engine.  The antecedent is taken from a rule that is
     ; currently under consideration, it has been instantiated with the bindings
```

```
  ; and has had all of the necessary objects instantiated with the appropriate
  ; values.  TRY-RULE initially makes the rule unique by instantiating the
  ; binding-variables within the rule with the bindings passed to this
  ; function.  The UNIFICATION-BINDINGS are then created which are used to
  ; establish which bindings are required after the rule has been fired.  If
  ; there are no unification bindings but the rule will still allow the
  ; antecedent to be validated then there is no need to find the required
  ; bindings or to make the rule unique again.

(when (intersection '(all try-rule) *debug*)
   (format t "~%> TRY-RULE.")
    (pause))

  (setf rule (instantiate-bindings rule bindings)
    unification-bindings (unify-antecedent-with-rule-consequents
      antecedent (rule-consequents rule)))

  (when (<> unification-bindings 'fail)    ;NIL or some value, (()())
    (when unification-bindings

      (multiple-value-bind
        (new-bindings new-required-bindings)
        (establish-required-bindings-from-unification-bindings
          bindings unification-bindings antecedent)
        (setf required-bindings new-required-bindings
          bindings new-bindings))

      (setf rule (instantiate-bindings rule unification-bindings)))

    (setf answer (use-rule '(,rule ,associated-rules)
        rulebase-name '(,bindings) :fire-actions ())
      inference-chain (rest inference-chain)))

(when (intersection '(all try-rule) *debug*)
   (format t "~%--- TRY-RULE ---")
    (format t "~%Required-bindings: ~a" required-bindings)
    (format t "~%Answer: ~a" answer)
    (format t "~%Bindings: ~a~%" bindings)
    (pause))

  (when answer
    (if required-bindings
      (setf answer
        (modify-bindings-to-include-original-and-required-bindings
          required-bindings answer bindings))
      (setf answer '(,bindings))))

(when (intersection '(all try-rule) *debug*)
   (format t "~%< TRY-RULE.")
   (format t "~%Antecedent: ~a" antecedent)
   (format t "~%Rule: ~a" rule)
   (format t "~%Bindings: ~a" bindings)
```

```
        (format t "~%Associated rules: ~a" associated-rules)
        (format t "~%Unification bindings: ~a" unification-bindings)
        (format t "~%Required bindings: ~a~%" required-bindings)
        (format t "~%Answer: ~a~%" answer)
        (pause))

    answer)

(defun unify (p1 p2 &optional bindings)
  (cond ((bindings-p p1)
         (unify-variable p1 p2 bindings))
        ((bindings-p p2)
         (unify-variable p2 p1 bindings))
        ((elements-p p1 p2)
         (unify-atoms p1 p2 bindings))
        ((recursive-p p1 p2)
         (unify-pieces p1 p2 bindings))
        (t 'fail)))

(defun unify-atoms (p1 p2 bindings)
  (if (eql p1 p2) bindings 'fail))

(defun unify-pieces (p1 p2 bindings)
  (let ((result (unify (first p1) (first p2) bindings)))
    (if (eq 'fail result)
        'fail
        (unify (rest p1) (rest p2) result))))

(defun unify-variable (p1 p2 bindings)
  (let ((binding (find-binding p1 bindings)))
    (if binding
        (unify (extract-bindings-value binding) p2 bindings)
        (if (insidep p1 p2 bindings)
            'fail
            (add-binding p1 p2 bindings)))))

(defun unify-antecedent-with-rule-consequents (antecedent consequents)
    ; 2 April 1992
    ;
    ; Calling Function:  TRY-RULE
    ; Returns:  (()()())
    ;
    ; This takes an antecedent that has had all the bindings and objects
    ; instantiated from the associated list of bindings and the object values
    ; respectively.  It then tries to UNIFY the antecedent with each consequent.
    ; Success results in a list of bindings being returned from the calling
    ; function.  If there exists bindings in both the antecedent and the
    ; consequent then the antecedent binding is the first in the pair.  For
    ; example ... (UNIFY '(THIS IS A $B1 $B2) '($B3 IS A $B4 EXAMPLE)) gives
    ; (($B3 THIS) ($B1 $B4) ($B2 EXAMPLE)).

  (do ((consequents consequents (rest consequents))
```

```
        (result nil)
        (success-switch 'fail))
       ((or (<> success-switch 'fail) (endp consequents)) success-switch)
       (let ((result (unify antecedent (first consequents))))
          (when (<> result 'fail) (setq success-switch result)))))

(defun unify-consequent-with-antecedents (consequent antecedents
        &aux (success-switch ()))
    ; 11 April 1992
    ;
    ; Calling Function:  AUGMENT-RULEBASE

    (if (eq (first (first antecedents)) 'or)

      (do ((antecedents (rest (first antecedents)) (rest antecedents)))
         ((or (endp antecedents) success-switch) success-switch)
         (setf success-switch (unify-consequent-with-antecedents
               consequent (first antecedents))))

      (do ((antecedents antecedents (rest antecedents)))
         ((or (endp antecedents) success-switch) success-switch)
         (when (<> (unify consequent (first antecedents)) 'fail)
           (setf success-switch t)))))

(defun use-for-all-rule (rule rulebase-name associated-rules
        bindings-list fire-actions
        &aux (list-object ()) (values ()) answer)
    ; 7 April 1992
    ;
    ; Calling Function:  USE-RULE
    ; Returns: ((()()()) (()()()) ... (()()()))
    ;
    ; This function checks to see if the list quantification rule,
    ; ie FOR ALL ???, where ??? is a list-object, has a traditional production
    ; rule bound to it or simply requires the firing of given consequents.  The
    ; bindings-list that is passed to this function is initially modified to
    ; accomodate the values contained within the list-object.  If the consequents
    ; only are to be fired then '(IF NOTHING) is concatenated to the rule in
    ; order to utilise the function USE-IF-THEN-RULE.  This is checked for in
    ; in the function.

    (when (intersection *debug* '(use-for-all-rule all))
      (format t "~%>USE-FOR-ALL-RULE")
      (format t "~%Rule:  ~a" rule)
      (pause))

   (setf list-object (third (second rule))
      values (object-value (symbol-value list-object))
      rule `(,(first rule) ,@(butfirstn 3 (second rule))))

    (when (intersection *debug* '(use-for-all-rule all))
      (format t "~%List object:  ~a" List-object)
```

```
      (format t "~%Values: ~a" values)
      (format t "~%Bindings list: ~a" bindings-list)
      (pause))

    (setf bindings-list
      (modify-bindings-list-to-include-$values values bindings-list))

    (when (intersection *debug* '(use-for-all-rule all))
      (format t "~%Bindings list: ~a~%" bindings-list)
      (pause))

    (when bindings-list
    (setf answer (use-if-then-rule rule rulebase-name associated-rules
          bindings-list fire-actions)))

    (when (intersection *debug* '(use-for-all-rule all))
      (format t "~%<USE-FOR-ALL-RULE")
      (format t "~%Answer:  ~a" answer)
      (pause))

    answer)

(defun use-if-then-rule (rule rulebase-name associated-rules
      bindings-list fire-actions
     &aux (antecedents (rule-antecedents rule)))
  ; 7 April 1992
  ;
  ; Calling Function:  USE-RULE
  ; Returns: (((()())) (()()) ... (()()))
  ;
  ; This infers on a traditional production rule if....then.  The 'dummy ifs'
  ; are generated by the USE-FOR-ALL-RULE function.  If there are no dummy
  ; ifs
  ; then try and confirm the antecedents within the rule else only fire the
  ; consequents.  The consequents will only be fired if fire-actions is true.
  ; This is the default and is only NIL when USE-RULE is called from
  ; TRY-RULE.

  (when (intersection '(all use-if-then-rule) *debug*)
    (format t "~%> USE-IF-THEN-RULE.")
    (format t "~%Rule Name: ~a" (rule-name rule))
    (format t "~%Bindings-list: ~a~%" bindings-list)
    (pause))

  (when (and (equal (second rule) '(if nothing))
       (not bindings-list))
    (setf bindings-list '(())))

  (when (<> (second rule) '(if nothing))
    (setf bindings-list (apply-filters (rule-antecedents rule)
          associated-rules rulebase-name bindings-list)))
```

```
    (when (intersection '(all use-if-then-rule) *debug*)
       (format t "~%--- USE-IF-THEN-RULE. ---")
       (format t "~%Antecedents: ~a" antecedents)
       (format t "~%Consequents: ~a" (rule-consequents rule))
       (format t "~%Bindings-list: ~a" bindings-list)
        (format t "~%Fire-actions:  ~a~%" fire-actions)
        (pause))

    (when (and bindings-list fire-actions)
        (fire-consequents (rule-consequents rule) bindings-list))

    (when (intersection '(all use-if-then-rule) *debug*)
       (format t "~%< USE-IF-THEN-RULE.")
        (format t "~%Bindings-list:  ~a" bindings-list)
        (pause))

     (setf inference-chain (rest inference-chain))
     bindings-list)

(defun use-rule (rule-pair rulebase-name &optional (bindings-list ())
       &key (fire-actions t)
       &aux (rule ()) (associated-rules ())
       (antecedents ()))
    ; 6 April 1992
    ;
    ; Calling Functions: (1) USE-RULEBASE, (2) TRY-RULEBASE
    ; Returns:  ((()()()) (()()()) ... (()()()))
    ;
    ; This function decides when to use the FOR-ALL rule or the IF-THEN rule.
    ; Furthermore, the key word FIRE-ACTIONS only becomes NIL when
    ; USE-RULE
    ; is called from TRY-RULE.  This is a consequence of the backward chaining
    ; mechanism (there is no need to fire the consequents as only one is being
    ; proved correct by firing the rule, namely the consequent related to the
    ; previous antecedent under consideration).  When called from TRY-RULE the
    ; rule-pair consists of an actual rule, as opposed to a rule number, and the
    ; associated rules.

     (if (listp (first rule-pair))
       (setf rule (first rule-pair))
       (setf rule (nth (- (first rule-pair) 1)
            (second (symbol-value rulebase-name)))))

     (setf associated-rules (second rule-pair)
       antecedents (second rule))

     (when (intersection *debug* '(all use-rule))
       (format t "~%> USE-RULE.")
       (format t "~%Rule: ~a" rule)
       (format t "~%Associated rules: ~a" associated-rules)
       (format t "~%Antecedents: ~a" antecedents)
       (format t "~%Bindings list: ~a~%" bindings-list)
```

```
        (pause))

      (setf inference-chain '((,(first rule-pair) (1) ,rulebase-name)
        ,@inference-chain))

    (cond ((equal (first antecedents) 'if)
           (use-if-then-rule rule
             rulebase-name
             associated-rules
             bindings-list
             fire-actions))
          ((equal (firstn 2 antecedents) '(for all))
           (use-for-all-rule rule
             rulebase-name
             associated-rules
             bindings-list
             fire-actions))
          (t
            (print 'rule-format-error))))

(defun use-rulebase (rulebase-name &aux (rulebase ())
        (network ())
        (rules (second (symbol-value rulebase-name)))
        (rule ()))
      ; 3 April 1992
      ;
      ; Calling function:  NON (called from control network or == prompt)
      ; Returns:  'RULEBASE-COMPLETE
      ;
      ; This takes a rulebase (network rules) and initiates a forward chaining
      ; inferencing process on the rules.  Backward chaining is performed as and
      ; when required.

      (do ((network (first (symbol-value rulebase-name)) (rest network)))
        ((endp network) 'rulebase-complete)

        (setf rule (first network))

        (when (intersection '(all use-rulebase) *debug*)
          (format t "~%USE-RULEBASE.")
          (format t "~%Network: ~a" network)
          (format t "~%Rule: ~a" rule)
          (pause))

        (setf inference-chain '(,@inference-chain))
        (use-rule rule rulebase-name)))

(defun variable-command (arguments &aux (command (first arguments))
        (value (second arguments)))
      ; 14 April 1992
      ;
      ; Calling Function: ->Q1
```

```
    (when (eval '(object-p ,value))
       (setf value (object-value (symbol-value value))))

    (concatenate 'string (string-capitalize (write-to-string command))
        "=" (if (stringp value) value
           (write-to-string value))))

(defun write-line-to-screen (line &aux lst)
     ; 14 April 1992
     ;
     ; Calling Function:  PRINT-TO-SCREEN

     (format t "~%")
     (dolist (word line t)
       (setf lst (coerce word 'list))

       (cond ((eq #\^ (second lst))
           (format t "~a " (string-upcase
                 (coerce (rest (rest (butlast lst))) 'string))))
          (t
            (format t "~a " word)))

       (when (eq #\. (car (last lst)))
          (format t " "))))

; ---------------------------------------------------------------------
;
; Inference Chain: (Antecedent-rule
;                      antecedent-number
;                           rulebase-name)

;(setf inference-chain '((4 (2 1) g11-rb) (7 (1) g11-rb) (8 (1) g11-rb) (1 (2) g11-rb)))

(defun why (&optional (chain inference-chain) (print-p t)
       &aux (state ()) (rulebase-name ())
       (rule ())  (antecedent ()) (action ())
       (prompt "KBFE - Why "))

     (if chain
       (progn
         (setf state (first chain)
         rulebase-name (third state)
         rule  (nth (- (first state) 1) (second (symbol-value rulebase-name))))
         (if (eq (length (second state)) 1)
        (setf antecedent (nth (- (first (second state)) 1) (rule-antecedents rule)))
        (setf antecedent (nth (- (second (second state)) 1)
          (nth (first (second state)) (first (rule-antecedents rule))))))

       (when print-p
       (format t "~%I am inferring on the rulebase ~a," rulebase-name)
       (format t "~%and trying to fire the following rule ... ~%")
       (print rule)
```

```
   (format t "~%~%... by proving the antecedent ... ~%")
   (print antecedent)
   (terpri)
   (terpri)))
    (format t "~%Sorry, end of inference chain.~%~%"))

(setf action (read-sentence prompt))

(cond ((intersection action '(exit quit q)) ())
   ((intersection action '(help h))
    (why-help)
    (why chain nil))
   ((intersection action '(display d))
    (print (symbol-value (second action))) (terpri) (terpri)
    (why chain nil))
   ((intersection action '(||))
    (if (eval `(object-p ,(second action)))
       (format t "~%~a :  ~a~%~%"
       (second action) (object-value (symbol-value (second action))))
       (format t "~%Sorry, there is no such object.~%~%"))
    (why chain nil))
   ((intersection action '(show s))
    (format t "~%Inference chain:  ~a" inference-chain)
    (format t "~%Remaining chain:  ~a~%~%" chain)
    (why chain nil))
   ((intersection action '(rulebase rb))
    (print (symbol-value rulebase-name))
    (terpri)
    (terpri)
    (why chain nil))
   ((intersection action '(top t))
    (why inference-chain))
   ((intersection action '(bottom b))
    (why `(,@(last inference-chain))))
   ((intersection action '(position p))
    (format t "~%~a level(s) remaining to base rule." (length chain))
    (format t "~%~a level(s) in current inference chain.~%~%"
       (length inference-chain))
    (why chain nil))
   ((intersection action '(r repeat))
    (why chain))
   ((intersection action '(bu backup))
    (if chain
       (setf chain (butfirstn
       (max (- (locate state inference-chain) 2)
                0)
            inference-chain))
       (setf chain `(,@(last inference-chain))))
    (why chain))
   ((intersection action '(why w))
    (why (rest chain)))
   (t
```

```
        (format t "~%Sorry, I don't understand your response.~%~%")
        (why chain nil))))

(defun update-inference-chain
    (antecedent-number &aux (state ()) (antecedent-no ()))

    (setf state (first inference-chain)
      antecedent-no (second state))

    (if (eq (length antecedent-no) 1)
      (setf antecedent-no `(,antecedent-number))
      (setf antecedent-no `(,(first antecedent-no) ,antecedent-number)))

    (setf inference-chain `((,(first state) ,antecedent-no ,(third state))
      ,@(rest inference-chain))))

(defun ->string (phrase &aux (scrn-text ""))
    ; 18 November 1992
    ;
    ; Calling Function:

    (do* ((phrase phrase (rest phrase))
      (word-count 1 (incf word-count))
      (word (string-downcase (write-to-string (first phrase)))
        (string-downcase (write-to-string (first phrase))))
      (coerced-word (coerce word 'list) (coerce word 'list))
      (last-word "."))  ;This is to force the first word to be capitalized
      ((endp phrase) scrn-text)

      (cond ((eq #\. (car (last (butlast coerced-word))))
        (setf word (coerce (rest (butlast coerced-word)) 'string)
          scrn-text (concatenate 'string scrn-text " " word)))
        ((eq #\^ (second coerced-word))
        (setf word (string-upcase (coerce (rest (rest (butlast
          coerced-word))) 'string))
          scrn-text (concatenate 'string scrn-text " " word)))
        ((eq #\. (car (last (coerce last-word 'list))))
        (setf word (string-capitalize word))
        (if (> word-count 1)
          (setf scrn-text (concatenate 'string scrn-text "  " word))
          (setf scrn-text (concatenate 'string scrn-text word))))
        (t
          (setf scrn-text (concatenate 'string scrn-text " " word))))

      (setq last-word word)))

(defun ObjectHelp (object &aux default-help novice-help experienced-help
      advanced-help)
    (setf default-help (second (assoc 'default (object-help (symbol-value object))))
      novice-help (second (assoc 'novice (object-help (symbol-value object))))
      experienced-help (second (assoc 'experienced
      (object-help (symbol-value object))))
```

```
        advanced-help (second (assoc 'advanced (object-help (symbol-value object)))))

    (when (not novice-help) (setf novice-help default-help))
    (when (not experienced-help) (setf experienced-help default-help))
    (when (not advanced-help) (setf advanced-help default-help))

    (cond ((and novice-help (eq (object-value TargetUserModel) 'novice))
       (print-to-screen novice-help))
      ((and experienced-help
          (eq (object-value TargetUserModel) 'Experienced))
       (print-to-screen experienced-help))
      ((and advanced-help (eq (object-value TargetUserModel) 'Advanced))
       (print-to-screen advanced-help))
      (t (format t HelpErrorMessage))))

(defun help (&optional (object ()) (pass ())
      &aux (action ()) (Prompt "KBFE - Help "))

    (cond ((and  object
          (not (stringp object))
          (boundp object)
          (not pass))
       (ObjectHelp object)
       (terpri)
       (Help object t))
      ((and object (stringp object))
       (format t HelpErrorMessage))
      (t

        (setf action (read-sentence prompt))

        (cond ((intersection action '(q exit quit)) ())
          ((intersection action '(h help))
           (help-help object)
           (help object t))
          ((intersection action '(| |))
           (if (eval '(object-p ,(second action)))
              (format t "~%~a :  ~a~%~%"
       (second action) (object-value (symbol-value (second action))))
              (format t "~%Sorry, there is no such object.~%~%"))
           (help object t))
          ((and object (intersection action '(s show)))
           (format t "~%~a~%~%" (symbol-value object))
           (help object t))
          ((and object (intersection action '(v value)))
           (format t "~%~a :  ~a~%~%" object (object-value
(symbol-value object)))
           (help object t))
          ((intersection action '(Help h))
           (help-help object)
           (Help object t))
          ((intersection action '(TUM))
```

```
            (format t "~%Target User Model :  ~a~%~%"
              (object-value TargetUserModel))
            (Help object t))
           ((intersection action '(UM))
            (format t "~%User Model :  ~a~%~%"
              (object-value UserModel))
            (help object t))
           (t
            (format t "~%Sorry, I don't understand your response.~%~%")
             (Help object t))))))

(defun link (filename &aux (file-success nil)
      (rb-name ()))

    (setf filename (concatenate 'string filename ".lsp")
      file-success (probe-file filename))

    (if (not file-success)
      (format t "~%File ~a is NOT present.~%~%" (string-upcase filename))
      (progn
        (with-open-file (stream filename :direction :input)
          (setf rb-name (second (read stream))))
        (load filename)
        (format t ";;; LINKING  ~a~%" (string-downcase filename))
        (eval `(augment-rulebase ,rb-name)))))

;;;;  --- FUNCTIONS WITHOUT ALPHA-NUMERIC LEADING CHARACTERS ---

(defun ->q1 (&rest arguments)
    (unless (member arguments q1 :test #'equal)
      (setf q1 `(,@q1 ,arguments))))

(defun ->1.0e??? (number &aux operator (exponent 0) string)
    ;  2 June 1992
    ;
    ;  Calling Function:  MAKE-FORM

    (if (>= number 10.0)
      (setf operator '/
        string "1.0e")
      (setf operator '*
        string "1.0e-"))

    (do ((success ()))
      (success)

      (setf number (eval `(,operator ,number 10.0))
        exponent (1+ exponent))

      (when (and (>= number 1) (< number 10.0))
        (setf success t)))
```

```
    (concatenate 'string string (write-to-string exponent)))

(defun write->q1 (arguments &aux (template ()) (file-stream ()))
    ; 14 April 1992
    ;
    ; Calling Function:  PHOENICS

  (setf template (first arguments)
      arguments (rest arguments))

  (with-open-file (file-stream (object-value target-file)
        :direction :output
        :if-does-not-exist :create
        :if-exists :append)
    (cond ((equal template '?[])
        (format file-stream
          (multiple-argument-command arguments)))
        ((equal template '?[]=)
        (format file-stream
          (single-argument-command arguments)))
        ((equal template '?=)
        (format file-stream
          (variable-command arguments)))
        ((equal template 'message)
        (format file-stream
          (Prepare-Message arguments))))))

(defun why-help ()
    (format t "~%---------- WHY Help ----------")
    (format t
"~%==========================================
==========================")
    (format t "~%Why       (W)  - Explain the need for the inference shown.")
    (format t "~%Repeat    (R)  - Repeat explanation of current position.")
    (format t "~%BackUp    (BU) - Backup through inference chain.")
    (format t "~%Bottom    (B)  - Move to base rule (root rule) in inference chain.")
    (format t "~%Top       (T)  - Move to the leaf rule in the inference chain.")
    (format t "~%Position  (P)  - Show position of interrogation in inference chain.")
    (format t "~%RuleBase  (RB) - Lists rules in rulebase.")
    (format t "~%? <object>     - What is the current value of <object> ?")
    (format t "~%Display ? (D)  - Display the object frame given by ?")
    (format t "~%Show      (S)  - Show inference chain and remaining chain to base
rule.~%")
    (format t "~%Help      (H)  - Shows this list.")
    (format t "~%Quit      (Q)  - Leave WHY interrogation.")
    (format t
"~%==========================================
==========================~%")
    (terpri))
```

```
(defun help-help (&optional (object ()))
    (format t "~%---------- HELP Help ----------")
    (format t
"~%==========
==========")
    (when object
    (format t "~%Show          (S)  - Show ~a structure." object)
    (format t "~%Value         (V)  - Show ~a value." object))
    (format t "~%Target User Model (TUM) - Show the Target User Model.")
    (format t "~%User Model      (UM)  - Show the User Model.")
    (format t "~%? <object>          - Display the object given by ?~%")
    (format t "~%Help          (H)  - Shows this list.")
    (format t "~%Quit          (Q)  - Leave Help")
    (format t
"~%==========
==========~%")
    (terpri))
```

End-Of-File

Filename: **GEOM.LSP**

```
(defun array-to-mesh-list (array limit &aux (lst ()))
    ; 14 May 1992
    ;
    ; Calling Function:  MESH-REGIONS

  (do ((i 0 (1+ i))
      (last-i -1))
     ((and (>= last-i 0) (= (aref array last-i) limit)) lst)
     (setf lst (append lst (list (aref array i))))
     (incf last-i)))

(defun assert-boundary-cardinal-information ()
    ; 8 May 1992
    ;
    ; Calling Function:  ASSERT-GEOMETRICAL-INFORMATION

  (dolist (boundary *boundaries* t)
     (remember-assertion
        `(cardinal for surface ,(first boundary) is
           ,(second (assoc 'cardinal (second boundary)))))))

(defun assert-geometrical-information ()
    ; 8 May 1992
    ;
    ; Calling Function:  GEOMETRY

  (assert-boundary-cardinal-information)
  (assert-obstruction-surface-information))
```

```
(defun assert-grid-information ()
    ; 8 May 1992
    ;
    ; Calling Function:  GRID

  (assert-regional-information)
  (assert-regional-boundary-containment))

(defun assert-obstruction-surface-information ()
    ; 8 May 1992
    ;
    ; Calling Function:  ASSERT-GEOMETRICAL-INFORMATION

  (dolist (boundary *boundaries* t)
    (when (eq 'obstruction (second (assoc 'type (second boundary))))
      (remember-assertion
        '(Surface ,(first boundary) is part of
          ,(second (assoc 'name (second boundary))))))))

(defun assert-regional-boundary-containment
  (&aux (start 0) (end 0) (regions ()) (axis ()) (surface ())
    (FC 0.0) (LC 0.0) (nodes ()) (number-of-regions 0)
    x1 x2 y1 y2 z1 z2 dummy1 dummy2)
    ; 8 May 1992
    ;
    ; Calling Function:  ASSERT-GRID-INFORMATION

  (dolist (boundary *boundaries* t)
    (setf nodes (first boundary)
      surface (first boundary)
      x1 (first (get-coordinates (first nodes)))
      x2 (first (get-coordinates (second nodes)))
      y1 (second (get-coordinates (first nodes)))
      y2 (second (get-coordinates (second nodes)))
      z1 (third (get-coordinates (first nodes)))
      z2 (third (get-coordinates (second nodes))))

    (dolist (axis *regions* t)
      (setf regions (second axis)
        number-of-regions (length regions)
        start 0
        end 0
        axis (first axis)
        dummy1 (symbol-value (eval '(join '(,axis 1))))
        dummy2 (symbol-value (eval '(join '(,axis 2))))
        FC (min dummy1 dummy2)
        LC (max dummy1 dummy2))

      (if regions
        (do ((regions regions (rest regions))
             (answer ()))
            ((or answer (endp regions)))
```

```
                (setf answer
                   (interface-boundary axis regions FC LC surface))
                (when (not answer)
                   (setf answer
                      (domain-boundary axis regions FC LC surface))))

             (remember-assertion
                '(surface ,surface is in ,axis regions 1 to 1))))))


(defun assert-regional-information
     (&aux (cell-count 1) (regions ()) (region-count 1) (number 0)
       (cell ()) (lst ()) (datum 0.0) (symbol ()) c1 c2)
     ; 8 May 1992
     ;
     ; Calling Function:  ASSERT-GRID-INFORMATION

   (dolist (axis *regions* t)
      (setf regions (cadr axis) axis (first axis) cell-count 1
          region-count 1 number (eval '(max 1 ,(length regions))))
      (remember-assertion '(,axis has ,number regions))

      (when (not regions)
         (remember-assertion '(,axis region 1 cells 1 to 1)))

      (dolist (region regions t)
         (setf lst (cadr (assoc 'mesh (cadr region)))
            number (length lst)
            datum (cadr (assoc 'c1 (cadr region)))
            c1 (cadr (assoc 'c1 (cadr region)))
            c2 (cadr (assoc 'c2 (cadr region))))

         (remember-assertion
            '(,axis region ,region-count cells ,cell-count to
             ,(+ (1- cell-count) number)))

         (dolist (cell lst t)
            (setf symbol (join '(,axis -grid))
               (object-value (symbol-value symbol))
               '(,@(object-value (symbol-value symbol))
                ,(+ datum cell)))
            (incf cell-count))
         (incf region-count))))

(defun assign-regional-alpha-values ()
     ; 14 May 1992
     ;
     ; Calling Function:  GEOMETRY

   (dolist (axis '(x y z) t)
      (dolist (region (second (assoc axis *regions*)) t)
         (when region
```

```
            (rplacd (assoc 'alpha (second region))
          (list
            (evaluate-alpha
              axis
              (second (assoc 'c1 (second region)))
              (second (assoc 'c2 (second region))))))))))

(defun assign-surface-type (pair surfaces
      &aux (surface1 (first pair)) (surface2 (second pair))
      (direction (first (second (assoc surface1 surfaces))))
      (ordinate1 nil) (ordinate2 nil))
    ; 8 May 1992
    ;
    ; Calling Function:  DEFINE-OBSTRUCTION-CARDINALS

  (cond ((eq 'west direction)
       (setq ordinate1 (first (get-coordinates (first surface1)))
         ordinate2 (first (get-coordinates (first surface2)))))
      ((eq 'south direction)
       (setq ordinate1 (second (get-coordinates (first surface1)))
         ordinate2 (second (get-coordinates (first surface2)))))
      ((eq 'low direction)
       (setf ordinate1 (third (get-coordinates (first surface1)))
         ordinate2 (third (get-coordinates (first surface2))))))

   (if (> ordinate1 ordinate2)
     (values (second (second (assoc surface1 surfaces)))
       (first (second (assoc surface1 surfaces))))
     (values (first (second (assoc surface1 surfaces)))
       (second (second (assoc surface1 surfaces))))))

(defun average (lst &aux (sum 0))
    ; 14 May 1992
    ;
    ; Calling Functions:  GET-BOUNDARY-LAYER-THICKNESS

  (if (all-numberp lst)
    (progn
      (dolist (num lst t)
        (setq sum (+ sum num)))
      (* 1.0 (/ sum (length lst))))    ;<--- Answer being returned.
    (format t "~%Error: ~a should be a purely numeric list.")))

(defun boundary-surface-plane-number (axis surface-nodes)
    ; 8 May 1992
    ;
    ; Calling Function:  INTERFACE-BOUNDARY

  (let ((answer (same-plane axis surface-nodes))
      (plane 1))
    (if answer
      (do ((regions (second (assoc axis *regions*)) (rest regions))
```

```
            (success-switch ()))
           ((or success-switch (endp regions)) plane)
           (if (= answer (second (assoc 'c1 (second (first regions)))))
              (setf success-switch t)
              (incf plane)))
        nil)))

(defun cardinals (&aux (surface nil) (a-list nil) (cardinal nil)
      (x ()) (y ()) (z ()))
   ; 8 May 1992
   ;
   ; Calling Function:  GEOMETRY

  (dolist (boundary *boundaries* t)
     (setq surface (first boundary)
        a-list (second boundary))
     (when (and (not (assoc 'cardinal a-list))
           (member (second (assoc 'type a-list))
              '(inlet outlet wall)))
        (setf x (same-plane 'x surface)
           y (same-plane 'y surface)
           z (same-plane 'z surface))

        (cond ((and (not (eq (object-value axis-1) 'unused)) x)
            (if (equal x (object-value x-min))
               (setf cardinal 'west)
               (setf cardinal 'east)))
           ((and (not (eq (object-value axis-2) 'unused)) y)
            (if (equal y (object-value y-min))
               (setf cardinal 'south)
               (setf cardinal 'north)))
           ((and (not (eq (object-value axis-3) 'unused)) z)
            (if (equal z (object-value z-min))
               (setf cardinal 'low)
               (setf cardinal 'high))))
        (rplacd
           (assoc surface *boundaries* :test 'equal)
           (list (acons 'cardinal
                 (list cardinal)
                 (second (assoc surface
                       *boundaries*
                       :test #'equal)))))))
  (define-obstruction-cardinals))

(defun check-connectivity (node connector)
   ; 8 May 1992
   ;
   ; Calling Function:  MAKE-SURFACES

  (if (member connector (get-connectivity node)) t nil))
```

```
(defun connect-node-to-connections (node connections &aux old)
    ; 1 May 1992
    ;
    ; Calling Function:  ENTER-CONNECTIVITIES

    (setf old (assoc node *nodes*))
    (nsubst `(,(first old)
       ,(second old)
       ,(remove-duplicates `(,@connections ,@(third old))))
      old *nodes*)

    (dolist (connection connections t)
      (setf old (assoc connection *nodes*))
      (nsubst `(,(first old)
          ,(second old)
          ,(remove-duplicates (cons node (third old))))
        old *nodes*)))

(defun convert (value)
    ; 15 May 1992
    ;
    ; Calling Function:  ENTER-NODAL-COORDINATES

    (* (object-value conversion-factor) value))

(defun define-obstruction-cardinals
    (&aux (names (get-obstruction-names)) (surfaces nil)
      (x ()) (y ()) (z ()) (pairs nil)
      (surface-type1 nil) (surface-type2 nil))
    ; 8 May 1992
    ;
    ; Calling Function:  CARDINALS

  (dolist (name names t)
    (setq surfaces (get-obstruction-surfaces name))
    (dolist (surface surfaces t)
      (setf x (same-plane 'x (first surface))
        y (same-plane 'y (first surface))
        z (same-plane 'z (first surface)))
      (when (not (assoc 'cardinal
                (second (assoc (first surface)
                      *boundaries*
                      :test #'equal))))
        (cond ((and (not (eq (object-value axis-1) 'unused)) x)
           (rplacd surface '((west east))))
          (y
            (rplacd surface '((south north))))
          ((and (not (eq (object-value axis-3) 'unused)) z)
           (rplacd surface '((low high))))
          (t t))))
    (setq pairs (pair-surfaces surfaces))
    (when (not (eq nil pairs))
```

```
        (dolist (pair pairs t)
          (multiple-value-bind
            (surface-type1 surface-type2)
            (assign-surface-type pair surfaces)
          (rplacd
            (assoc (first pair) *boundaries* :test #'equal)
            (list (acons 'cardinal
                         (list surface-type1)
                         (second (assoc (first pair) *boundaries* :test #'equal)))))
          (rplacd
            (assoc (second pair) *boundaries* :test #'equal)
              (list (acons 'cardinal
                         (list surface-type2)
                         (second (assoc (second pair) *boundaries* :test #'equal)))))))))

(defun domain-boundary (axis regions FC LC surface &aux (start 1) (end 1))
    ; 8 May 1992
    ;
    ; Calling Function:  ASSERT-REGIONAL-BOUNDARY-CONTAINMENT

  (do ((regions regions (rest regions)))
      ((endp regions))
      (when (= start 1)
        (when (= FC (second (assoc 'c1 (cadr (first regions)))))
          (setf start (first (first regions)))))
      (when (= LC (second (assoc 'c2 (cadr (first regions)))))
        (setf end (first (first regions))))))

  (if (= start end)
      (progn
        (eval '(remember-assertion
         '(surface ,surface is in ,axis regions ,start to ,start))) t)
      (progn
        (eval '(remember-assertion
        '(surface ,surface is in ,axis regions ,start to ,end)))
        t)))

(defun enter-connectivities ()
    ; 8 May 1992
    ;
    ; Calling Function:  ENTER-NODES

    (terpri)
    (do ((prompt "Enter connectivity command, ? for help ")
      (response ()))
      ((member (first response) '(complete end)))

      (setf response (read-sentence prompt))

      (cond ((member (first response) '(|| h help))
          (print-connectivity-help) (terpri))
        ((equal response '(list)) (print *nodes*) (terpri) (terpri))
```

```
        ((and (member (first response) '(c connect))
              (> (length response) 2)
              (all-integerp (rest response)))
         (connect-node-to-connections (second response)
              (butfirstn 2 response)))
        ((and (member (first response) '(r remove))
              (all-integerp (rest response)))
         (remove-connectivities (second response)
              (butfirstn 2 response)))
        ((not (member (first response) '(complete end)))
         (format t "~%Connectivity command error.~%")))))

(defun enter-inlets ()
    ; 8 May 1992
    ;
    ; Calling Function:  GEOMETRY
    ;
    ; Notes:  This function uses some semi-inferencing through the use of the
    ;         function ASK-FACT

  (fetch 'number-of-inlets)
  (do ((i 1 (1+ i))
       (name nil)
       (nodes nil))
      ((= i (1+ (object-value number-of-inlets))))
      (setf name (ask-fact '(boundary name for inlet ,i
                   ((type text) (disallowedvalues boundary-names)
                    (consequent boundary-names includes $value))))
            nodes
            (enter-list (make-prompt '(surface nodes for ,name))))

      (remember-assertion '(boundary name for inlet ,i ,nodes is ,name))

      (update-boundaries (make-surfaces nodes) :name name :type 'inlet)))

(defun enter-nodes (&aux coordinates)
    ; 30 April 1992
    ;
    ; Calling Function: ????????

  (do ((node 1 (incf node)))
      ((equal coordinates 'end) (setf *nodes* (reverse *nodes*)))

      (when (and (= node 1) *nodes*)
            (setf node (1+ (length *nodes*))))

      (setf coordinates (enter-nodal-coordinates node))

      (when (<> coordinates 'end)
            (setf *nodes* (cons '(,node ,coordinates ()) *nodes*)))

    (enter-connectivities))
```

```
(defun enter-nodal-coordinates (node)
    ; 30 April 1992
    ;
    ; Calling Function:  ENTER-NODES

    (terpri)
  (do ((axes '(axis-1 axis-2 axis-3))
       (axis ()) (prompt ())  (response '(0.0) '(0.0))
       (x 0.0) (y 0.0) (z 0.0) (terminate-entry ()))
      ((or (endp axes)
           terminate-entry)
       (if terminate-entry 'end `(,(convert x) ,(convert y) ,(convert z))))

      (setf axis (first axes))

      (when (<> (object-value (symbol-value axis)) 'unused)
        (setf prompt (make-prompt
                `(enter the ,(object-value (symbol-value axis))
                   ordinate for node ,node))
              response (read-sentence prompt)))

      (cond ((and (> (length response) 1)
                  (member (first response) '(m modify)))
             (setf response (modify-xyz node `(,x ,y ,z) response)
                   x (first response)
                   y (second response)
                   z (third response))
             (terpri))
            ((member (first response) '(|| h help))
             (print-enter-nodal-coordinates-help) (terpri))
            ((equal (first response) 'list)
             (print (reverse *nodes*))
             (terpri) (terpri))
            ((equal response '(end))
             (if (or (equal axis 'axis-1)
                     (and (equal axis 'axis-2)
                          (equal (object-value coordinates)
                                 'cylindrical)))
                 (setf terminate-entry t)
                 (format t "~%Enter a numeric value.")))
            ((numberp (first response))
             (cond ((equal axis 'axis-1)
                    (setf x (float (first response))))
                   ((equal axis 'axis-2)
                    (setf y (float (first response))))
                   (t (setf z (float (first response))))))
             (setf axes (rest axes)))
            (t (print 'error)
               (terpri)))))
```

```
(defun enter-obstructions ()
      ; 8 May 1992
      ;
      ; Calling Function:  GEOMETRY

    (fetch 'number-of-obstructions)
    (do ((i 1 (1+ i))
         (name nil)
         (nodes nil))
        ((= i (1+ (object-value number-of-obstructions))))
        (when (= i 1) (fetch 'porosity-definition))
        (setf name (ask-fact '(boundary name for obstruction ,i
                    ((type text) (disallowedvalues boundary-names)
                     (consequent boundary-names includes $value))))
              nodes
              (enter-list (make-prompt '(surface nodes for ,name))))

        (remember-assertion
           '(boundary name for obstruction ,i ,nodes is ,name))

        (update-boundaries
           (make-surfaces nodes) :name name :type 'obstruction)))

(defun enter-outlets ()
      ; 8 May 1992
      ;
      ; Calling Function:  GEOMETRY

    (fetch 'number-of-outlets)
    (do ((i 1 (1+ i))
         (name nil)
         (nodes nil))
        ((= i (1+ (object-value number-of-outlets))))
        (setf name (ask-fact '(boundary name for outlet ,i
                    ((type text) (disallowedvalues boundary-names)
                     (consequent boundary-names includes $value))))
              nodes
              (enter-list (make-prompt '(surface nodes for ,name))))

        (remember-assertion '(boundary name for outlet ,i ,nodes is ,name))

        (update-boundaries (make-surfaces nodes) :name name :type 'outlet)))

(defun evaluate-alpha (axis c1 c2 &aux (nodes '(() ()))
         (coordinates ())
         (ordinate 0))
      ; 14 May 1992
      ;
      ; Calling Function: ASSIGN-REGIONAL-ALPHA-VALUES
    (dolist (node *nodes* t)
       (setq node (first node)
             coordinates (get-coordinates node))
```

```
        (cond ((eq axis 'x)
               (setq ordinate (first coordinates)))
              ((eq axis 'y)
               (setq ordinate (second coordinates)))
              ((eq axis 'z)
               (setq ordinate (third coordinates))))
        (when (= c1 ordinate)
          (rplaca nodes (append (first nodes) (list node))))
        (when (= c2 ordinate)
          (rplacd nodes (list (append (second nodes) (list node))))))

    (let ((First-Nodes (first nodes))
          (Last-Nodes (second nodes))
          (First-Pairs nil)
          (Last-Pairs nil))
        (do* ((first-nodes first-nodes (rest first-nodes))
              (last-nodes last-nodes (rest last-nodes))
              (first-node (first first-nodes) (first first-nodes))
              (last-node (first last-nodes) (first last-nodes)))
             ((and (endp first-nodes) (endp first-nodes)))
             (when (rest first-nodes)
               (dolist (node (rest first-nodes) t)
                 (setf First-Pairs (append First-Pairs
                 `((,first-node ,node))))))
             (when (rest last-nodes)
               (dolist (node (rest last-nodes) t)
                 (setf Last-Pairs (append Last-Pairs
                 `((,last-node ,node)))))))
        (setf nodes (cons First-Pairs (cons Last-Pairs nil))))

    (let ((alpha 0.0)
          (side-1 ())
          (side-2 ()))
        (setf side-1 (remove nil (get-boundary-type-list (first nodes)))
              side-2 (remove nil (get-boundary-type-list (second nodes))))
        (cond ((and (intersection '(wall obstruction) side-1)
                    (intersection '(wall obstruction) side-2))
               0.5)
              ((and (intersection '(wall obstruction) side-1)
                    (not (intersection '(wall obstruction) side-2)))
               0.0)
              (t
                1.0))))

(defun geometry nil
    ; 8 May 1992
    ;
    ; Calling Function:  RULEBASE - GEOMETRY-RB

    (set-domain)
    (make-boundaries)
    (enter-inlets)
```

```
    (enter-outlets)
    (enter-obstructions)
    (cardinals)
    (remove-redundant-boundaries)
    (make-regions)
    (assign-regional-alpha-values)
    (assert-geometrical-information))

(defun get-boundary-type-list (nodes)
     ; 14 May 1992
     ;
     ; Calling Function:  EVALUATE-ALPHA

    (mapcar #'(lambda (item)
          (do* ((boundaries *boundaries* (rest boundaries))
              (boundary (first boundaries) (first boundaries))
              (a-list (second boundary) (second boundary))
              (surface (first boundary) (first boundary))
              (answer nil))
             ((or (endp boundaries)
                  answer) answer)
             (if (or (equal item surface)
                     (equal (reverse item) surface))
                 (setq answer (second (assoc 'type a-list)))
                 (setq answer nil))))
        nodes))

(defun get-boundary-layer-thickness (&aux (X-min 0) (U-Bar 0) (ldelta 0))
     ; 14 May 1992
     ;
     ; Calling Function:  OBJECT SLOT

     ;   X-min:  Smallest length of the wall/obstruction surfaces
     ;   U-Bar:  Average inlet mainstream velocity
     ;   lDelta: Boundary layer thickness based on the following :-
     ;
     ;                lDelta = 5 (Nu*X-min/U-Bar)^0.5
     ;
     ;          This is the equation for the LAMINAR boundary layer thickness.
     ;          (Schlichting, H., page 598 -->).
     ;

    (setq X-min (eval '(min ,@(get-wall-lengths)))
        U-Bar (average (get-inlet-velocities))
        lDelta (* 5.0
          (sqrt (/ (* (object-value viscosity)
                      X-min)
                   U-Bar))))
    ldelta)
```

```
(defun get-coordinates (node)
     ; 8 May 1992
     ;
     ; Calling Function:  Any function that uses the nodal coordinates

   (second (assoc node *nodes*)))

(defun get-connectivity (node)
     ; 8 May 1992
     ;
     ; Calling Function:  CHECK-CONNECTIVITY

   (third (assoc node *nodes*)))

(defun get-inlet-velocities (&aux (stream ()))
     ; 14 May 1992
     ;
     ; Calling Function:  GET-BOUNDARY-LAYER-THICKNESS

   (setq stream
      (apply-filters
         '(($phi at inlet boundary $name is constant at $velocity)
          ((u1 v1 w1) includes $phi))))

   (do ((stream stream (rest stream))
      (velocities ()))
     ((endp stream) velocities)   ;<--- Returning list VELOCITIES
     (setq velocities (append (cdr (assoc '$velocity (first stream)))
           velocities))))

(defun get-obstruction-surfaces (name &aux (surfaces nil))
     ; 8 May 1992
     ;
     ; Calling Function:  DEFINE-OBSTRUCTION-CARDINALS

   (mapcar #'(lambda (boundary)
         (when (eq (second (assoc 'name (second boundary)))
               name)
            (setq surfaces (acons (first boundary)
                  (list '())
                  surfaces))))
      *boundaries*)
    surfaces)

(defun get-obstruction-names ()
     ; 8 May 1992
     ;
     ; Calling Function:  DEFINE-OBSTRUCTION-CARDINALS
   (let ((names nil))
      (mapcar #'(lambda (boundary)
       (when (eq (second (assoc 'type (second boundary)))
          'obstruction)
```

```
          (setq names (union (list (second
             (assoc 'name (second boundary))))
                names))))
          *boundaries*) names))

(defun get-wall-lengths (&aux (lengths ()) (coords1 ()) (coords2 ()))
     ; 14 May 1992
     ;
     ; Calling Function:  GET-BOUNDARY-LAYER-THICKNESS

   (dolist (boundary *boundaries* t)
      (dolist (surface (make-surfaces (first boundary)) t)
         (setf coords1 (get-coordinates (first surface))
            coords2 (get-coordinates (second surface))
            lengths '(,@lengths ,(abs (- (first coords1)
                  (first coords2)))
             ,(abs (- (second coords1)
                  (second coords2)))
             ,(abs (- (third coords1)
                  (third coords2)))))))
   (remove-duplicates (remove 0.0 (mapcar #'float lengths))))

(defun interface-boundary (axis regions FC LC surface)
     ; 8 May 1992
     ;
     ; Calling Function:  ASSERT-REGIONAL-BOUNDARY-CONTAINMENT

   (let ((plane (boundary-surface-plane-number axis surface))
       (number-of-planes (1+ (length regions))))
      (cond ((not (numberp plane)) nil)
         ((and (> plane 1)
             (< plane number-of-planes))
          (eval '(remember-assertion
          '(surface ,surface interfaces ,axis regions ,(- plane 1) and
            ,plane))) t)
         ((= plane 1)
          (eval '(remember-assertion
           '(surface ,surface is in ,axis regions 1 to 1))) t)
         ((= plane number-of-planes)
          (eval '(remember-assertion
           '(surface ,surface is in ,axis regions ,(- plane 1) to
                ,(- plane 1)))) t)
         (t nil))))

(defun list-nodes ()
     ; 8 May 1992
     ;
     ; Calling Function:  MAKE-BOUNDARIES

   (let ((lst nil))
      (dotimes (number (length *nodes*) (reverse lst))
         (setq lst (cons (1+ number) lst)))))
```

```
(defun make-boundaries ()
    ; 8 May 1992
    ;
    ; Calling Function:  GEOMETRY

  (setq *boundaries* nil)
  (let ((surfaces (make-surfaces (list-nodes))))
    (dolist (surface surfaces t)
      (setq *boundaries*
        (acons surface
          '(((name unknown) (type wall)))
          *boundaries*)))))

(defun make-regions (&aux (xs ()) (ys ()) (zs ()))
    ; 14 May 1992
    ;
    ; Calling Function:  GEOMETRY

  (setq *regions* '((x ()) (y ()) (z ()))
    x1 () x2 () y1 () y2 () z1 () z2 ())

  (dolist (node *nodes* t)
    (setf xs (cons (first (get-coordinates (first node))) xs)
      ys (cons (second (get-coordinates (first node))) ys)
      zs (cons (third (get-coordinates (first node))) zs)))

  (setf xs (sort (remove-duplicates xs) #'<)
    ys (sort (remove-duplicates ys) #'<)
    zs (sort (remove-duplicates zs) #'<))

  (do* ((x1 xs (rest x1))
    (y1 ys (rest y1))
    (z1 zs (rest z1))
    (x2 (rest x1) (rest x1))
    (y2 (rest y1) (rest y1))
    (z2 (rest z1) (rest z1))
    (form nil)
    (count 1 (1+ count)))
   ((and (endp x2) (endp y2) (endp z2)) t)
   (dolist (axis '((x x1 x2) (y y1 y2) (z z1 z2)) t)
     (when (first (symbol-value (third axis)))
       (rplacd (assoc (first axis) *regions*)
         (list
           (append
             (second (assoc (first axis) *regions*))
             `((,count ((alpha 0)
           (c1 ,(first
           (symbol-value   (second axis))))
           (c2 ,(first
           (symbol-value   (third axis))))
           (l ,(- (first (symbol-value (third axis)))
          (first (symbol-value (second axis)))))))
```

```
            (mesh ()))))))))
          (remember-assertion
            '(,(first axis) region ,count co-ordinates
             ,(first (symbol-value (second axis))) to
             ,(first (symbol-value (third axis)))))))))

(defun make-surfaces (nodes &optional (lst nil))
    ; 8 May 1992
    ;
    ; Calling Function:
  (cond ((endp nodes) nil)
      (t (dolist (node (rest nodes) t)
           (if (check-connectivity (first nodes) node)
             (setq lst (cons
               (sort '(,(first nodes) ,node) #'<)
                 lst))
             t))
        (concatenate 'list lst (make-surfaces (rest nodes))))))

(defun mesh-regions (&aux (alpha 0.0) (L 0.0) (regions ()) (lst ()) (y ()))
    ; 14 May 1992
    ;
    ; Calling Function:  GRID

  (setf y (make-array 1000 :element-type 'single-float))
  (dolist (axis *regions* t)
    (setf regions (cadr axis))
    (dolist (region regions t)
      (when region
        (setf alpha (float (cadr (assoc 'alpha (cadr region))))
          L (float (cadr (assoc 'L (cadr region)))))
        (external-call generate-mesh (float (object-value
            aspect-ratio))
          (float (object-value delta))
          L alpha y)
        (rplacd (assoc 'mesh (cadr region))
          (list (array-to-mesh-list y L)))))))

(defun modify-xyz (current-node xyz response &optional (prompt ""))
      &aux axis prompt modifiable-node answer)
    ; 30 April 1992
    ;
    ; Calling Function:  ENTER-NODAL-COORDINATES

    (terpri)
  (setf axis (second response))

  (unless prompt
    (setf prompt (make-prompt
        '(Original coordinates (,current-node ,xyz) <nl>
          Modify ,axis ordinate for node ,current-node))))
```

```
(if (and (= (length response) 3)
         (<> (third response) current-node))
    (let* ((modifiable-node (third response))
           (x (/ (first (second (assoc modifiable-node *nodes*)))
                 (object-value conversion-factor)))
           (y (/ (second (second (assoc modifiable-node *nodes*)))
                 (object-value conversion-factor)))
           (z (/ (third (second (assoc modifiable-node *nodes*)))
                 (object-value conversion-factor))))
      (if (> modifiable-node current-node)
          (format t "~%This has not yet been defined.~%")
          (progn
       (nsubst '(,modifiable-node
           ,(modify-xyz modifiable-node
              '(,(convert x) ,(convert y) ,(convert z))
           (butlast response) prompt)
              ,(third (assoc modifiable-node *nodes*)))
              (assoc modifiable-node *nodes*)
                    *nodes*)
                '(,x ,y ,z))))
      (cond ((equal axis 'x)
           (if (first xyz)
              '(,(convert (enter-numeric prompt :type 'real))
               ,(second xyz) ,(third xyz))
              (print 'modify-x-error)))
          ((equal axis 'y)
           (if (second xyz)
              '(,(first xyz)
               ,(convert (enter-numeric prompt :type 'real))
               ,(third xyz))
              (print 'modify-y-error)))
          (t (if (third xyz)
                 '(,(first xyz) ,(second xyz)
                  ,(convert (enter-numeric prompt :type 'real)))
                 (print 'modify-z-error))))))

(defun name-walls (&aux (surface nil) (a-list nil)
       (sentence '(Boundary name for wall surface))
       (prompt "") (name nil))
    ; 8 May 1992
    ;
    ; Calling Function:  GEOMETRY

   (dolist (boundary *boundaries* t)
      (setq surface (first boundary))
      (setq a-list (second                    ))
      (when (eq (second (assoc 'type a-list)) 'wall)
         (setq name (ente                    make-prompt '(,@sentence,surface))))
         (remember-assertion '(,@sentence ,surface is ,name))
         (update-boundaries (list surface)
            :name name))))
```

```
(defun orientate-nodes (n1 n2) ;order nodes so that angle 0 -> 90 degrees
    ; 8 May 1992
    ;
    ; Calling Function:
  (let ((angle (surface-angle n1 n2)))
    (if (and (>= angle 0.0) (<= angle 1.5708)) ;Radians
      (cons n1 (cons n2 nil))
      (cons n2 (cons n1 nil)))))

(defun pair-surfaces (surfaces &aux (west-east nil) (south-north nil)
      (low-high nil) (answer ()))
    ; 8 May 1992
    ;
    ; Calling Function:  DEFINE-OBSTRUCTION-CARDINALS

  (dolist (surface surfaces t)
    (cond ((eq 'west (first (second surface)))
       (setq west-east (cons (first surface) west-east)))
      ((eq 'south (first (second surface)))
       (setq south-north (cons (first surface) south-north)))
      ((eq 'low (first (second surface)))
       (setq low-high (cons (first surface) low-high)))
      (t t)))
  (when west-east
    (setf answer (cons west-east answer)))
  (when low-high
    (setf answer (cons low-high answer)))
  (when south-north
    (setf answer (cons south-north answer)))
  answer)

(defun print-connectivity-help ()
    ; 1 May 1992
    ;
    ; Calling Function:  ENTER-CONNECTIVITIES

    (format t "~%--- Connectivity HELP ---")
    (format t "~%LIST                    - list nodal information")
    (format t "~%CONNECT node_i j .... z - Connects node_i to j .... z")
    (format t "~%C       node_i j .... z - Connects node_i to j .... z")
    (format t "~%REMOVE  node_i j .... z - Removes node_i from j ... z")
    (format t "~%R       node_i j .... z - Removes node_i from j ... z")
    (terpri))

(defun print-enter-nodal-coordinates-help ()
    ; 8 May 1992
    ;
    ; Calling Function:  ENTER-NODAL-COORDINATES

    (print-to-screen '(The nodal-coordinates should be entered in
        <,(object-value dimensional-units)> depending on the prompt.  <NL>
        LIST        -  Lists the nodes <NL>
```

```
        M AXIS NODE  -  Modify the coordinate of NODE on axis AXIS <NL>
        M AXIS       -  Modify the current nodal coordinate on AXIS)))

(defun remove-connectivities (node connections &aux old)
    ; 1 May 1992
    ;
    ; Calling Function:  ENTER-CONNECTIVITIES

    (dolist (connection connections t)

      (setf old (assoc node *nodes*))
      (nsubst `(,(first old)
          ,(second old)
          ,(remove connection (third old)))
        old *nodes*)

      (setf old (assoc connection *nodes*))
      (nsubst `(,(first old)
          ,(second old)
          ,(remove node (third old)))
        old *nodes*)))

(defun replace-name-of-boundary (boundary name)
    ; 8 May 1992
    ;
    ; Calling Function:  UPDATE-BOUNDARIES

  (let ((surface (car boundary))
        (data (cadr boundary)))
    (cons surface
      (list (acons 'name (list name)
              (remove-key-from-alist 'name data))))))

(defun replace-nth (position value lst &aux (lst-len (length lst)))
    (do ((count 1 (incf count))
      (success-switch ())
      (head ()))
      ((or (> position lst-len)
          success-switch)
      (append (reverse head) lst))

      (if (= position count)
        (setf head (cons value head)
          success-switch t)
        (setf head (cons (first lst) head)))

      (setf lst (rest lst))))

(defun replace-type-of-boundary (boundary type)
    ; 8 May 1992
    ;
    ; Calling Function:  UPDATE-BOUNDARIES
```

```
   (let ((surface (car boundary))
         (data (cadr boundary)))
      (cons surface
         (list (acons 'type (list type)
                  (remove-key-from-alist 'type data))))))

(defun remove-key-from-alist (key lst)
     ; 8 May 1992
     ;
     ; Calling Function:  REPLACE-TYPE-OF-BOUNDARY,
REPLACE-NAME-OF-BOUNDARY

   (if (eq (first (first lst)) key)
      (rest lst)
      (cons (first lst) (remove-key-from-alist key (rest lst)))))

(defun remove-redundant-boundaries
     (&optional (boundaries *boundaries*)
       &aux (boundary (first boundaries))
       (x (first (get-coordinates (caar boundary))))
       (y (second (get-coordinates (caar boundary))))
       (z (third (get-coordinates (caar boundary)))))
     ; 8 May 1992
     ;
     ; Calling Function:  GEOMETRY

   (cond ((endp boundaries) nil)
       ((and (eq (cadr (assoc 'type (second boundary))) 'obstruction)
          (or (and (not (eq (object-value axis-1) 'unused))
                (same-plane 'x (first boundary))
                (or (= x (object-value x-min))
                   (= x (object-value x-max))))
             (and (same-plane 'y (first boundary))
                (or (= y (object-value y-min))
                   (= y (object-value y-max))))
             (and (not (eq (object-value axis-3) 'unused))
                (same-plane 'z (first boundary))
                (or (= z (object-value z-min))
                   (= z (object-value z-max))))))
        (remove-redundant-boundaries (rest boundaries)))
       (t
         (setq *boundaries* (cons (first boundaries)
               (remove-redundant-boundaries
                  (rest boundaries)))))))

(defun set-minimum-region-size (&aux (minimum 1.0e10))
     ; 8 May 1992
     ;
     ; Calling Function:

   (dolist (axis *regions* minimum)
      (dolist (region (second axis) t)
```

```
        (setf minimum
          (eval '(min ,minimum
                  ,(second (assoc 'l (second region)))))))))

(defun same-plane (axis surface-nodes
      &aux (index
        (- (length (member axis '(z y x))) 1))
      (node1 (first surface-nodes))
      (ord1 (nth index (get-coordinates node1)))
      (surface-nodes (rest surface-nodes))
      (ord2 ()) (node2 ()))
    ; 8 May 1992
    ;
    ; Calling Function:  BOUNDARY-SURFACE-PLANE-NUMBER,
    ;          CARDINALS,
    ;       DEFINE-OBSTRUCTION-CARDINALS,
    ;       REMOVE-REDUNDANT-BOUNDARIES

  (cond ((endp surface-nodes) ord1)
      ((= ord1 (nth index (get-coordinates (first surface-nodes))))
       (same-plane axis surface-nodes))
      (t nil)))

(defun set-domain (&aux (coordinates ()))
    ; 8 May 1992
    ;
    ; Calling Function:  GEOMETRY

  (setf (object-value x-min) 1.0e35 (object-value x-max) -1.0e35
      (object-value y-min) 1.0e35 (object-value y-max) -1.0e35
      (object-value z-min) 1.0e35 (object-value z-max) -1.0e35)
  (dolist (node *nodes* t)
      (setf coordinates (get-coordinates (first node)))
      (when (< (first coordinates) (object-value x-min))
        (setf (object-value x-min) (first coordinates)))
      (when (> (first coordinates) (object-value x-max))
        (setf (object-value x-max) (first coordinates)))
      (when (< (second coordinates) (object-value y-min))
        (setf (object-value y-min) (second coordinates)))
      (when (> (second coordinates) (object-value y-max))
        (setf (object-value y-max) (second coordinates)))
      (when (< (third coordinates) (object-value z-min))
        (setf (object-value z-min) (third coordinates)))
      (when (> (third coordinates) (object-value z-max))
        (setf (object-value z-max) (third coordinates)))))

(defun surface-angle (n1 n2)
    ; 8 May 1992
    ;
    ; Calling Function:  ORIENTATE-NODES

  (let* ((xf (first (get-coordinates n1)))
```

```
        (yf (second (get-coordinates n1)))
        (xl (first (get-coordinates n2)))
        (yl (second (get-coordinates n2)))
        (h (sqrt (+ (sqr (- xl xf)) (sqr (- yl yf))))))
       (if (< yl yf)
          (* -1.0 (acos (/ (- xl xf) h)))
          (acos (/ (- xl xf) h)))))

(defun sqr (x) (* x x))

(defun update-boundaries (surfaces &key name type)
     ; 8 May 1992
     ;
     ; Calling Functions:  ENTER-INLETS, ENTER-OUTLETS,
     ; ENTER-OBSTRUCTIONS,
     ; NAME-WALLS
   (dolist (surface surfaces t)
      (let* ((boundary (assoc surface *boundaries* :test #'equal)))
         (setq *boundaries* (remove boundary *boundaries*))
         (when name
       (setq boundary (replace-name-of-boundary boundary name)))
         (when type
       (setq boundary (replace-type-of-boundary boundary type)))
         (setq *boundaries* (cons boundary *boundaries*)))))

(defun xc_1 (bindings)
     (first (get-coordinates (first (second (assoc '$nodes bindings))))))

(defun xc_2 (bindings)
     (first (get-coordinates (second (second (assoc '$nodes bindings))))))

(defun yc_1 (bindings)
     (second (get-coordinates (first (second (assoc '$nodes bindings))))))

(defun yc_2 (bindings)
     (second (get-coordinates (second (second (assoc '$nodes bindings))))))

(defun zc_1 (bindings)
     (third (get-coordinates (first (second (assoc '$nodes bindings))))))

(defun zc_2 (bindings)
     (third (get-coordinates (second (second (assoc '$nodes bindings))))))

; NON ALPHA-NUMERIC FUNCTION NAMES

(defun ^ (n1 n2) (exp (* n2 (log n1))))
```

APPENDIX H

LISP mathematical parser

```
(defun mparse (expression &aux (operators '([] * / * ^ expt - + - ()))
                               (op1 ()) (operator1 ())
                               (op2 ()) (operator2 ()) (result 0))

  (when (numberp expression) (setf expression (list expression)))

  (setf op1 (first expression)
        operator1 (second expression)
        op2 (third expression)
        operator2 (fourth expression))

  (cond ((eq operator1 '^)
         (setf operator1 'expt))
        ((eq operator2 '^)
         (setf operator2 'expt))
        (t t))

  (cond ((endp expression) nil)
        ((and op1 (listp op1))
         (setf result (mparse op1))
         (if (listp result)
             (cons result (mparse (rest expression)))
             (mparse (cons result (rest expression)))))
        ((or (not (numberp op1))
             (not (member operator1 operators)))
         (setf result (mparse (rest expression)))
         (if (listp result)
             (cons op1 result)
             (cons op1 '(,result))))
        ((and (numberp op1)
              (not operator1)) op1)
        ((and op2 (listp op2))
         (setf result (mparse op2))
         (mparse '(,op1 ,operator1 ,result ,@(cdddr expression))))
        ((or (member operator2 (member operator1 operators))
             (not (member operator2 (member operator1 operators))))
         (setf result (eval '(,operator1 ,op1 ,op2)))
         (mparse (cons result (cdddr expression))))
        (t
         (eval '(,operator1 ,op1 ,(mparse (cddr expression)))))))
```

APPENDIX I

Pseudo real time control FORTRAN code

```
      Subroutine SERTIC
      Include 'PHOINC:SATEAR'
      Include 'PHOINC:GRDLOC'
      Include 'PHOINC:GRDEAR'

      common/sertic1/Residuals(50,100),IshINum,IshPhi(0:50),
     1 shC(50,3),IRange(50),shMonitor(50,100),TRatio,shRes(50,100),
     1 IshRange
      common/lgrnd/lg(20)/igrnd/ig(20)/rgrnd/rg(20)

      if (isweep.eq.1) return

      if (isweep.eq.2) then
        call OpenFiles
        call Initialise
        call GetVariables
        return
      endif

      if (IshINum.lt.IG(19)) return    ! IG(19) = lower limit for range
                  ! IG(20) = Upper limit for range

      call GetResiduals
      call GetMonitorValues
      IshINum=IshINum+1

      if (IG(20)-IG(19)-IshINum) 10,10,20

10    IshINum=1

      call CurveFit

      call CloseFiles

20    return
      end

      Subroutine OpenFiles

      open(unit=51,file='pjet.bef',status='unknown')
      open(unit=52,file='pjet.aft',status='unknown')
      open(unit=53,file='sertic.log',status='unknown')
      open(unit=54,file='residuals.dat',status='unknown')

      return
      end
```

```
      Subroutine CloseFiles

      close(51)
      close(52)
      close(53)
      close(54)

      return
      end

      Subroutine Initialise

      Include 'PHOINC:SATEAR'
      Include 'PHOINC:GRDLOC'
      Include 'PHOINC:GRDEAR'

      common/sertic1/Residuals(50,100),IshINum,IshPhi(0:50),
     1 shC(50,3),IRange(50),shMonitor(50,100),TRatio,shRes(50,100),
     1 IshRange
      common/lgrnd/lg(20)/igrnd/ig(20)/rgrnd/rg(20)
      IshINum=1
      if (IG(19).eq.0) IG(19)=1
      TRatio=2.0
      return
      end

      Subroutine GetVariables

      Include 'PHOINC:SATEAR'
      Include 'PHOINC:GRDLOC'
      Include 'PHOINC:GRDEAR'

      common/sertic1/Residuals(50,100),IshINum,IshPhi(0:50),
     1 shC(50,3),IRange(50),shMonitor(50,100),TRatio,shRes(50,100),
     1 IshRange

C The variables to be solved are highlighted in the array ISLN(IshPhi).
C       Solved variables are given a value greater than 2.

      IshCount=0
      Do 10 Ish=1,50
10      if (ISLN(Ish).gt.2) IshCount=IshCount+1
      IshPhi(0)=IshCount
      IshCount=1
      Do 20 Ish=1,50
        if (ISLN(Ish).gt.2) then
                IshPhi(IshCount)=Ish
                IshCount=IshCount+1
        endif
20    continue
      return
      end
```

```
      Subroutine GetMonitorValues
      Include 'PHOINC:SATEAR'
      Include 'PHOINC:GRDLOC'
      Include 'PHOINC:GRDEAR'

      COMMON/GR1/STOR(50)/GR2/SLBRES(50)/GR3/TOTRES(50)
      common/sertic1/Residuals(50,100),IshINum,IshPhi(0:50),
     1 shC(50,3),IRange(50),shMonitor(50,100),TRatio,shRes(50,100),
     1 IshRange

      if (IZ.eq.IZMON) then
        Do 10 Ish=1,IshPhi(0)
10              shMonitor(IshPhi(Ish),IshINum)=
     1        f(l0f(lbiffv(IshPhi(Ish))+IYMON+NY*(IXMON-1)))
      endif
      return
      end

      Subroutine GetResiduals
      Include 'PHOINC:SATEAR'
      Include 'PHOINC:GRDLOC'
      Include 'PHOINC:GRDEAR'

      COMMON/GR1/STOR(50)/GR2/SLBRES(50)/GR3/TOTRES(50)
      common/sertic1/Residuals(50,100),IshINum,IshPhi(0:50),
     1 shC(50,3),IRange(50),shMonitor(50,100),TRatio,shRes(50,100),
     1 IshRange

      DO 10 Ish=1,IshPhi(0)
10      Residuals(IshPhi(Ish),IshINum)=TOTRES(IshPhi(Ish))
      return
      end

      Subroutine CurveFit
      Real shX,shY,shSX,shSY,shSXX,shSXY,shSYY,shCO

      Dimension shX(85),shY(85)

      common/sertic1/Residuals(50,100),IshINum,IshPhi(0:50),
     1 shC(50,3),IRange(50),shMonitor(50,100),TRatio,shRes(50,100),
     1 IshRange
      common/lgrnd/lg(20)/igrnd/ig(20)/rgrnd/rg(20)

      Do 10 Ish=1,IshPhi(0)
10      IRange(IshPhi(Ish))=IG(20)-IG(19)-2

      Do 20 Ish=1,IshPhi(0)
        IModify=0
25      shSX=0.0
        shSY=0.0
        shSXY=0.0
        shSXX=0.0
```

```
      shSYY=0.0

      Do 30 Ish1=1,IRange(IshPhi(Ish))

            shSX=shSX+float(Ish1)
            shSY=shSY+log(Residuals(IshPhi(Ish),Ish1))
            shSXX=shSXX+(float(Ish1)**2.0)
            shSYY=shSYY+
     1           (log(Residuals(IshPhi(Ish),Ish1)))**2.0
            shSXY=shSXY+float(Ish1)*
     1           log(Residuals(IshPhi(Ish),Ish1))
30    continue

      shSXX=shSXX-(shSX**2.0)/float(IRange(IshPhi(Ish)))
      shSYY=shSYY-(shSY**2.0)/float(IRange(IshPhi(Ish)))
      shSXY=shSXY-(shSX*shSY)/float(IRange(IshPhi(Ish)))
      shC(IshPhi(Ish),2)=shSXY/shsXX
      shC(IshPhi(Ish),1)=exp(
     1              (shSY/float(IRange(IshPhi(Ish))))-
     1         shC(IshPhi(Ish),2)*shSX/
     1         IRange(IshPhi(Ish)))
      shC(IshPhi(Ish),3)=shSXY/(sqrt(shSXX*shSYY))

      if (IModify.eq.0) then
c           write(54,*) 'Before','Phi = ',IshPhi(Ish)
c     else
c           write(54,*) 'After','Phi = ',IshPhi(Ish)
c     endif

c     Do 40 Ish1=1,IRange(IshPhi(Ish))
      if (IshPhi(Ish).eq.7) then
      write(IUnit,1) 'A'
      write(IUnit,3) '0 0'
      write(IUnit,3) 'W1*'
      write(IUnit,1) '*'
      write(IUnit,1) '*'
      write(IUnit,1) '*'
      write(IUnit,1) '*'
      write(IUnit,1) '*'
      write(IUnit,1) '1'
      write(IUnit,5) 'X,1,0'
      write(IUnit,11) IRange(7)

1     format(a1)
3     format(A3)
5     format(a5)
11    format(i3)
      Do 40 Ish1=1,IRange(7)
            shRes(IshPhi(Ish),Ish1)=
     1      shC(IshPhi(Ish),1)*
     1       exp(shC(IshPhi(Ish),2)*Ish1)-
     1       Residuals(IshPhi(Ish),Ish1)
```

```
                write(54,*) Ish1,shRes(IshPhi(Ish),Ish1)
40      continue
        shSY=0.0
        shSYY=0.0

        Do 50 Ish1=1,IRange(IshPhi(Ish))
                shSYY=shSYY+shRes(IshPhi(Ish),Ish1)**2.0
                shSY=shSY+shRes(IshPhi(Ish),Ish1)
50      continue

        Stdev=sqrt((shSYY-
     1                  (shSY*shSY/float(IRange(IshPhi(Ish))))/
     1          IRange(IshPhi(Ish)))

        write(54,*) 'Stdev = ',Stdev
        endif
        endif
        Ish1=1

        if (IModify.eq.1) goto 20

        if (IshPhi(Ish).eq.7) call PA4(51)

60      if (Ish1.gt.IRange(IshPhi(Ish))) goto 90
        if (abs(shRes(IshPhi(Ish),Ish1))-Tratio*Stdev) 80,80,70
70      Do 75 Ish2=Ish1,IRange(IshPhi(Ish))-1
                Residuals(IshPhi(Ish),Ish2)=
     1          Residuals(IshPhi(Ish),Ish2+1)
                shRes(IshPhi(Ish),Ish2)=
     1          shRes(IshPhi(Ish),Ish2+1)
75      continue
        IRange(IshPhi(Ish))=IRange(IshPhi(Ish))-1
        IModify=1
        goto 60
80      Ish1=Ish1+1
        goto 60
90      if (IModify.eq.1) then
                goto 25
        endif
20    continue

      call PA4(52)
      return
      end

      Subroutine MINITAB(IUnit)

      common/sertic1/Residuals(50,100),IshINum,IshPhi(0:50),
     1 shC(50,3),IRange(50),shMonitor(50,100),TRatio,shRes(50,100),
     1 IshRange
      COMMON/GR1/STOR(50)/GR2/SLBRES(50)/GR3/TOTRES(50)
      common/lgrnd/lg(20)/igrnd/ig(20)/rgrnd/rg(20)
```

```
      Do 10 Ish=1,(ig(20)-ig(19)-2)
        write(IUnit,*) Ish,(Residuals(IshPhi(Ish1),Ish),
     1                  Ish1=1,IshPhi(0))
10    continue
      return
      end

      Subroutine PA4(IUnit)

      Character*8 Text

      common/sertic1/Residuals(50,100),IshINum,IshPhi(0:50),
     1 shC(50,3),IRange(50),shMonitor(50,100),TRatio,shRes(50,100),
     1 IshRange
      COMMON/GR1/STOR(50)/GR2/SLBRES(50)/GR3/TOTRES(50)

c     Do 10 Ish=1,IshPhi(0)
c       write(IUnit,*) '*** New data ***'
        Text='After.*'
        write(IUnit,1) 'A'
        write(IUnit,3) '0 0'
        write(IUnit,3) 'W1*'
        if (IUnit.eq.51) Text='Before*'
        write(IUnit,7) Text
        write(IUnit,*) shC(7,3)
        write(IUnit,1) '*'
        write(IUnit,1) '*'
        write(IUnit,1) '*'
        write(IUnit,1) '2'
        write(IUnit,5) 'X,1,0'
        write(IUnit,11) IRange(7)
        Do 20 Ish1=1,IRange(7)
c       Do 20 Ish1=1,IRange(IshPhi(Ish))
c20             write(IUnit,*) Ish1,Residuals(IshPhi(Ish),Ish1)
20              write(IUnit,*) Ish1,Residuals(7,Ish1)

        write(IUnit,5) 'X,1,0'
        write(IUnit,11) IRange(7)
        Do 30 Ish1=1,IRange(7)
                shAA=shC(7,1)*exp(shC(7,2)*
     1              float(Ish1))
                write(IUnit,*) Ish1,shAA
30      continue

1       format(a1)
3       format(a3)
5       format(a5)
7       format(a7)
11      format(i3)
c10   continue
      return
      end
```

APPENDIX J

Published work

Development of an Intelligent Front-End for a Computational Fluid Dynamics Package

K. Jambunathan, E. Lai, S. L. Hartle and B. L. Button

Department of Mechanical Engineering, Nottingham Polytechnic, Burton Street, Nottingham, NG1 4BU, UK

Computer modelling based on numerical simulation packages is becoming increasingly popular to aid design. A preliminary development of an Intelligent Front-End (IFE) for integration into a commercial Computational Fluid Dynamics (CFD) package is described. An expert system environment, LEONARDO, is used to implement the IFE thus facilitating rapid development and easy access to externally located data. The methodology used for knowledge acquisition and representation in conjunction with external FORTRAN coding has been shown to be a powerful approach in improving response times.

Key Words: Intelligent Front-Ends, IFE, expert systems, PHOENICS, Computational Fluid Dynamics, CFD, numerical simulation packages, Intelligent Knowledge Based System, IKBS.

NOMENCLATURE

B_r	Radial body force contribution
B_z	Axial body force contribution
c_p	Specific heat at constant pressure
d	Diameter of jet
k	Thermal conductivity
p	Pressure
r	Radial distance
T	Temperature
T_{wall}	Impingement wall temperature
T_{jet}	Jet inlet temperature
v	Radial velocity
w	Axial velocity
w_{jet}	Mean jet velocity at nozzle exit
z	Axial distance
z_{max}	Maximum value of z

Greek

μ	Laminar dynamic viscosity
ρ	Density

1. INTRODUCTION

1.1. Numerical simulation packages

Within the engineering industry the use of numerical simulation packages, such as finite element, boundary element or finite difference schemes play an extremely important role in computer aided design. Recent emergence of cheap yet powerful microcomputers has enabled relatively small companies to access comprehensive Computational Fluid Dynamics (CFD) packages. CFD modelling of physical situations can be an extremely complex procedure and it usually requires specialist expertise and familiarity with the package to establish a working model. The generation of an input data file to a CFD package can be cumbersome and simple modifications usually require extensive alterations to the format. These modifications can be very susceptible to catastrophic failure due to the enormous potential for human errors in typing or a momentary lack of concentration. This risk increases directly with the size of a data file which is usually large in a realistic problem. The data files contain information relating to the geometry, boundary conditions, properties and solution parameters associated with the analysis. In common with other numerical schemes most CFD packages tend to be of a generic nature thus allowing numerous permutations of analyses to be performed. For example a CFD package might be able to consider laminar/turbulent flows, heat/mass transfer and chemical reaction processes. The availability of a number of options for the user to choose increases the number of commands he may have to enter, each of which informs the main source code to either include or omit a particular option from the analysis, thus limiting the number of variables the program needs to solve. Clearly the marketability of the software package relates to its versatility to model a variety of different class of problems. Even though the availability of CFD packages is increasing, their popularity and potential market is yet to be fully realised, especially by small companies. This is mainly because of the costs involved in releasing engineers to attend the necessary training courses to become proficient with the package, and the need for these engineers to have at least a basic understanding of the processes involved in order to get the full benefit from the courses. The time required to become familiar with a numerical stress analysis package[1] is anything up to one year depending on the ability of the user[2]. This timescale is typical for most software packages and experience has shown this to be so for PHOENICS[3], which is being used as an example for this work and to which an Intelligent Front-End (IFE) is being developed.

Accepted October 1990. Discussion closes June 1991

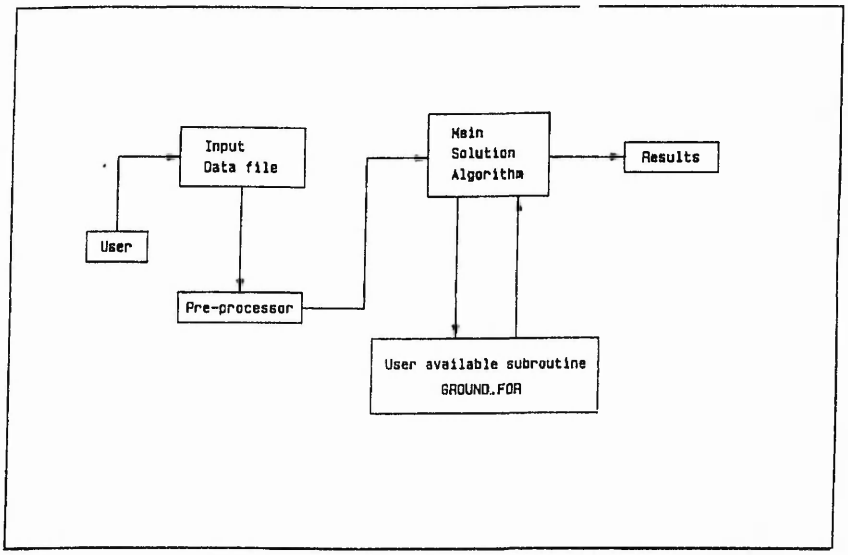


Fig. 1. Simplified overall program structure of PHOENICS

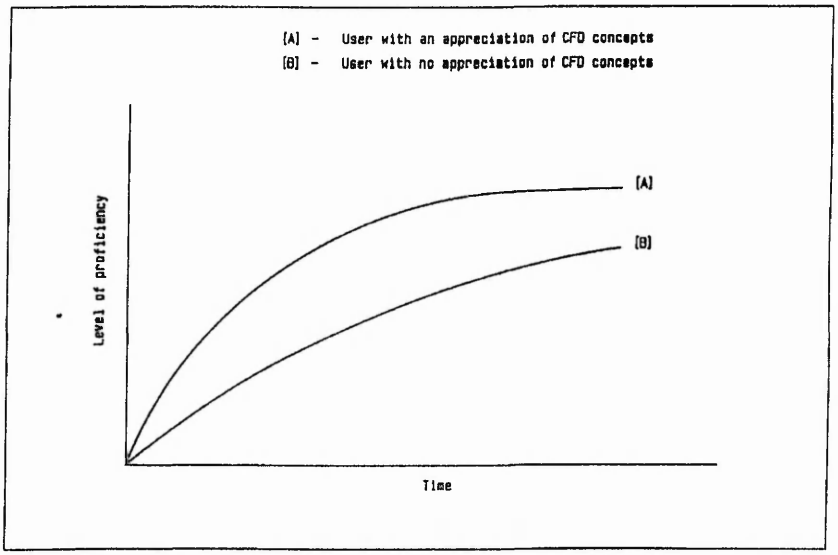


Fig. 2. Suggested learning curves for PHOENICS users

PHOENICS is a general-purpose finite difference package designed for the simulation of fluid flow, heat/mass transfer and chemical reaction processes. The program structure of PHOENICS is basically divided into two sections: the preprocessor and the solution algorithm (Fig. 1). A user defines the problem using the PHOENICS Input Language to generate an input data file which will be interpreted and compiled by the preprocessor. The compiled version of the data file is then submitted to the solution algorithm for analysis and the results are written to an output file. Figure 2 suggests how the learning curves for two individual users of PHOENICS may differ when one has prior knowledge of CFD concepts and finite difference techniques, while the other has not.

1.2. IFE based on IKBS techniques

An IFE, as shown in Fig. 3, is designed to remove the complexities associated with entering a problem specification to a numerical simulation package. IFE's differ significantly from conventional data entry techniques in that they are able to explicitly define a user's problem in the terminology required by the package. This is performed by asking the user questions, structured in

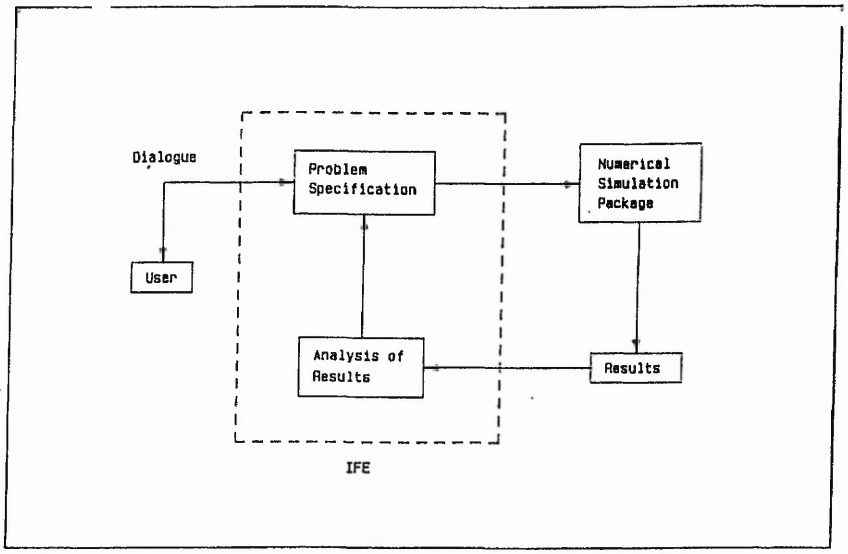


Fig. 3. The typical structure of an Intelligent Front-End (IFE)

English, that allow the IFE to create the necessary commands to correctly specify the complete problem to be analysed. The questions asked by the IFE are entered into a Knowledge Base (KB) using either a classical artificial intelligence language such as LISP or PROLOG, or by using a commercially developed Expert System (ES) shell or environment. These questions are then inferred upon by the knowledge manager or inference engine, and depending on the order in which they are presented to the user the necessary commands are generated. The number of questions to be entered reflects the number of possible commands the numerical package has. Usually the questions are entered into the KB as a set of rules that have been generated by a process of knowledge elicitation, and these rules have to be represented within the KB. The IFE should provide post processing facilities[4], to assess the results of the analysis, and to advise the user on possible modifications to the input data file. A true IFE should not be limited to one application of an ES. Indeed it should contain various combinations of the ten areas highlighted as potential ES applications[5]. Examples of such combinations are interpretation, monitoring and advice. ES's have been coupled with aerodynamic packages to aid the design process of axial cooling fans[6]. Other applications of IFE's are given in Refs 2, 7–11.

The feasibility of introducing an IKBS to PHOENICS was investigated[12]. The authors concluded that there was a need for developing such a system because the PHOENICS compiler only superficially examines the input data file for specific syntax errors and it does not indicate any omissions from the data file that could affect a successful analysis. The study also highlighted a need for an on-line adviser that would aid the correct modelling sequence and which can be accomplished by using a set of structured questions to be inferred upon by the knowledge manager.

1.3. Selection of Artificial Intelligence tool

Artificial Intelligence (AI) and more specifically ES usually contain logical symbolic processing as well as conventional computational techniques[13,14]. ES's can be written by using standard languages like FORTRAN but these have one distinct disadvantage over symbolic processing techniques; the developer must introduce a pseudo-inferencing procedure into the code. This requirement makes subsequent modifications difficult without rewriting the code. On the other hand the symbolic reasoning approach which uses an inference engine with either backward or forward chaining will automatically consider the new rules. Recent development of ES shells or environments has adopted the latter approach and they are much easier to use because the KB can be easily modified. Several commercially available ES environments were considered[15–17] but the ultimate restrictions of cost, potential versatility, and availability made LEONARDO[18] the most favourable in this case.

LEONARDO utilises numerous knowledge represenation techniques including frames, production rules, quantification rules and a procedural language. It also allows interfacing to externally compiled conventional programs and various database files. The hierarchial class structures often used for representing the object/parent-object relationships are performed by quantification rules. LEONARDO also facilitates the use of *lists* which have been used extensively in this project.

1.4 Computational Fluid Dynamics

CFD utilises the capacity of digital computers to perform vast amounts of repetitive calculations to solve the governing equations of motion: Navier-Stokes, continuity, energy and pressure equations[19]. These equations can be solved iteratively using either finite difference or finite element methods.

2. CASE STUDY: PREDICTION OF JET IMPINGEMENT HEAT TRANSFER

The initial development of the front end was performed on an IBM-PC AT compatible using the LEONARDO ES

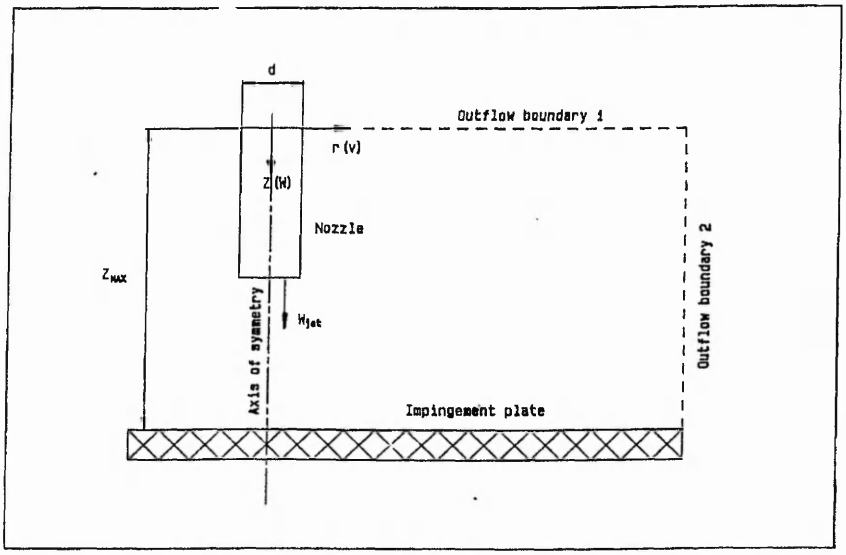


Fig. 4. Unconfined jet impingement geometry

environment. The system was designed with the eventual transportation to a VAX-785 machine in mind. A case study was conducted to analyse a laminar jet impinging upon a surface within an unconfined region, as shown in Fig. 4. The governing equations of motion are as follows.

The Navier-Stokes equations

$$\rho\left(v\frac{\partial v}{\partial r}+w\frac{\partial v}{\partial z}\right)=-\frac{\partial p}{\partial r}+\mu\left(\frac{\partial^2 v}{\partial r^2}+\frac{1}{r}\frac{\partial v}{\partial r}+\frac{\partial^2 v}{\partial z^2}-\frac{v}{r^2}\right)+B_r \tag{1}$$

$$\rho\left(v\frac{\partial w}{\partial r}+w\frac{\partial w}{\partial z}\right)=-\frac{\partial p}{\partial z}+\mu\left(\frac{\partial^2 w}{\partial r^2}+\frac{1}{r}\frac{\partial w}{\partial r}+\frac{\partial^2 w}{\partial z^2}\right)+B_z \tag{2}$$

where B_r and B_z represent the radial and axial body force contributions respectively. The additional viscous dissipation terms are neglected.

The continuity equation

$$\frac{1}{r}\frac{\partial}{\partial r}(rv)+\frac{\partial w}{\partial z}=0 \tag{3}$$

The energy equation

$$\frac{1}{r}\frac{\partial}{\partial r}(r\rho Tv)+\frac{\partial}{\partial z}(\rho Tw)=\frac{1}{r}\frac{\partial}{\partial r}\left(\frac{rk}{c_p}\frac{\partial T}{\partial r}\right)+\frac{\partial}{\partial z}\left(\frac{k}{c_p}\frac{\partial T}{\partial z}\right) \tag{4}$$

The boundary conditions are:-

Impingement plate

$$w=0,\quad v=0 \qquad \text{at} \quad z=Z_{max}$$
$$T=T_{wall}$$

Velocity profile at nozzle exit

Flat $w=W_{jet}$

Temperature at nozzle exit

$$T=T_{jet}$$

Outflow boundaries 1 and 2

$$p=0.0 \qquad \text{at} \quad z=0$$
$$\text{and} \quad r=37.5d$$

Considering the problem specification given above, it would take a proficient user approximately fifteen minutes to formulate the correct model for PHOENICS. This assumes that the flow conditions involved are thoroughly understood and the overall boundary conditions are known. The PHOENICS input data file for the case study is shown in Fig. 5. On overage it would take in excess of thirty minutes, for an experienced user, to simply type in the comands, to be followed by manually checking for typing errors or omissions

3. AN INTELLIGENT FRONT-END FOR PHOENICS

The primary reason for developing an IFE was to enable novice users of CFD to become familiar with the techniques employed to model fluid flow problems using a commercial software package. To this end it was important to integrate into the system, knowledge relating the metamorphosis of the user's problem definition to appropriate PHOENICS commands. This can be seen to be the fundamental requirements placed upon an IFE, and as such would consist of generating a usable data file from an interactive session with a user. This approach was suggested within the feasibility

```
TALK='.RUN( 1, 1);VDU=TTY
TEXT(2D UNCONFINED IMPINGING ROUND JET - THERMAL)
REAL(WIN)
WIN=1500*ENUL/0.01
CARTES=F
NY=61
YVLAST=0.05
YFRAC(1)=-11.0,     YFRAC(2)=9.09091E-3
YFRAC(3)=1.0,       YFRAC(4)=4.45455E-3
YFRAC(5)=49.0,      YFRAC(6)=1.82746E-2
NZ=37
ZWLAST=1.0
ZFRAC(1)=-18.0,     ZFRAC(2)=1.66667E-3
ZFRAC(3)=19.0,      ZFRAC(4)=1.11111E-3
SOLUTN(P1,Y,Y,Y,N,N,N)
SOLUTN(V1,Y,Y,N,N,N,N)
SOLUTN(W1,Y,Y,N,N,N,N)
SOLUTN(H1,Y,Y,Y,N,N,N)
ENUL=1.461E-5
RHO1=1.2250
CONPOR(0.0,CELL,1,1,12,12,1,18)
FIINIT(P1)=RHO1*WIN
FIINIT(V1)=WIN
FIINIT(W1)=WIN
FIINIT(H1)=21.0
PATCH(WALL,CELL,1,1,1,NY,NZ,NZ,1,1)
COVAL(WALL,V1,FIXVAL,0.0)
COVAL(WALL,W1,FIXVAL,0.0)
COVAL(WALL,H1,FIXVAL,100.0)
PATCH(REGION,CELL,1,1,1,11,1,18,1,1)
COVAL(REGION,V1,FIXVAL,0.0)
PATCH(INLET,LOW,1,1,1,11,1,1,1,1)
COVAL(INLET,W1,ONLYMS,WIN)
COVAL(INLET,P1,FIXFLU,RHO1*WIN)
COVAL(INLET,H1,ONLYMS,21.0)
PATCH(OUTLET1,NORTH,1,1,NY,NY,1,NZ-1,1,1)
COVAL(OUTLET1,P1,FIXVAL,0.0)
PATCH(OUTLET2,LOW,1,1,13,NY,1,1,1,1)
COVAL(OUTLET2,P1,FIXVAL,0.0)
PATCH(PLATE,HWALL,1,1,1,NY,NZ-1,NZ-1,1,1)
COVAL(PLATE,W1,FIXVAL,0.0)
COVAL(PLATE,V1,1.0,0.0)
PATCH(PIPEOUT,SWALL,1,1,13,13,1,18,1,1)
COVAL(PIPEOUT,W1,1.0,0.0)
COVAL(PIPEOUT,V1,FIXVAL,0.0)
PATCH(PIPEIN,NWALL,1,1,11,11,1,18,1,1)
COVAL(PIPEIN,W1,1.0,0.0)
COVAL(PIPEIN,V1,FIXVAL,0.0)
LSWEEP=300
RESREF(W1)=1.0E-8
RESREF(V1)=1.0E-8
RESREF(P1)=1.0E-8
RESREF(H1)=1.0E-8
RELAX(V1,FALSDT,0.5)
RELAX(W1,FALSDT,0.5)
RELAX(P1,LINRLX,0.8)
RELAX(H1,FALSDT,1.0)
ECHO=F
OUTPUT(H1,Y,Y,Y,Y,Y,Y)
OUTPUT(P1,Y,Y,Y,Y,Y,Y)
OUTPUT(V1,Y,Y,Y,Y,Y,Y)
OUTPUT(W1,Y,Y,Y,Y,Y,Y)
IYMON=14
IZMON=33
NPLT=1
STOP
```

Fig. 5. Laminar jet impingement PHOENICS data file

study[12]. However, it was thought prudent to also allow partially experienced users the ability to have their manually created data files checked prior to submitting them for analysis. This facility would mimic the process of asking the advice of an expert who would indicate any errors with the data and recommend possible improvements. Certain mistakes, for example the inadvertent transposition of arguments within commands, have been shown to be accepted by PHOENICS, thus indicating an acceptable data file, but have lead to erroneous results. Errors such as these can take hours to find if a large data file has been submitted. In order to eliminate the tedious task of checking the independently generated data file manually, thus reducing the time involved, a prototype system was developed that would examine the contents of the file and would assess the validity of the commands. This would upgrade the existing facility within the PHOENICS preprocessor, which simply states that an error occurs on one or more lines, to a higher level whereby detailed information regarding the invalid statements would be displayed.

Figure 6 shows the status of the development to date and how the data file generator and the data file checker are utilised within the overall system.

3.1. Knowledge elicitation

The knowledge for the IFE was obtained from three different sources. Firstly, practical experience with PHOENICS as a user. The commands that have to be used to correctly model a CFD problem are explicitly defined within the PHOENICS reference manuals. This was thought to be possibly the most important method of understanding the operation of PHOENICS since there is no substitute for experience. The second method was through directly conversing with experienced users, and extracting their knowledge on problem specifications. Finally, by acting as a pseudo-expert when supervising and advising inexperienced users. Knowledge acquired in this manner was transformed into various rules which formed the infrastructure of the KB.

3.2. Knowledge representation and structuring

Production rules were the primary method of knowledge representation together with a combination of other standard techniques such as procedural language, class structures, frames, singular objects and lists. The knowledge bases were structured in a manner which would allow inferencing processes to be performed on the rulesets sequentially. The initial arrangement was such that for a specific question there would be a corresponding object (or defined variable) instantiated. This approach soon exceeded the size limitation of LEONARDO since its PC version can only accommodate a maximum of one thousand objects. Subsequent modifications of the structure of the KB led to the categorisation of statements which defined a CFD problem into seven areas or lists, as illustrated below. Information organised in this way significantly reduce the number of objects and hence the possibility of exceeding the size limitation.

Category	Method of acquisition
– Grid specification	Defined or Inferred
– Solution variables	Inferred
– Fluid properties	Defined
– Initial conditions	Inferred
– Boundary conditions	Defined
– Solution parameters	Inferred
– Output requirements	Defined

The prototype IFE was developed for specific analyses in jet impingement and as such would not permit a generic problem definition. Figure 6 shows an IFE without post processing facilities.

3.3. IFE data file checker/advisor

The infrastructure of the data file checker/adviser is shown in Fig. 7. In its present form PHOENICS only superficially examines the input data file and gives ambiguous error messages relating to the syntax which

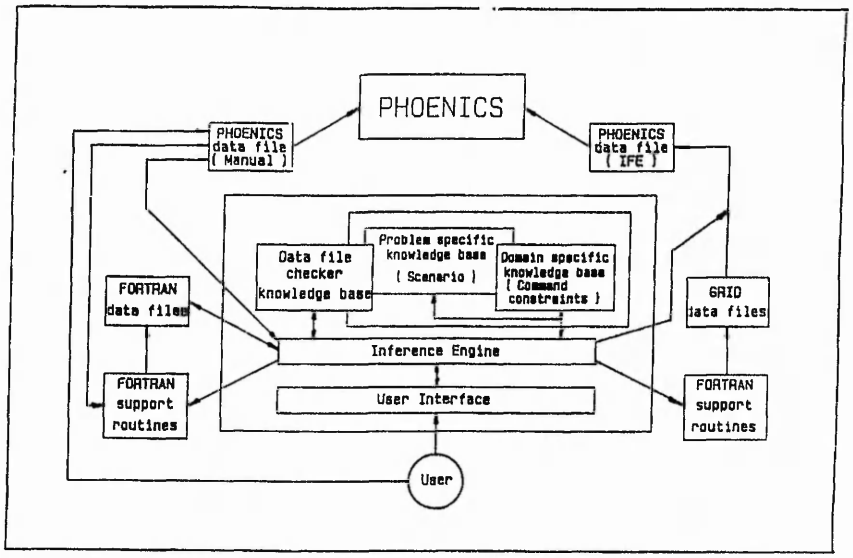


Fig. 6. Preliminary infrastructure of PHOENICS IFE

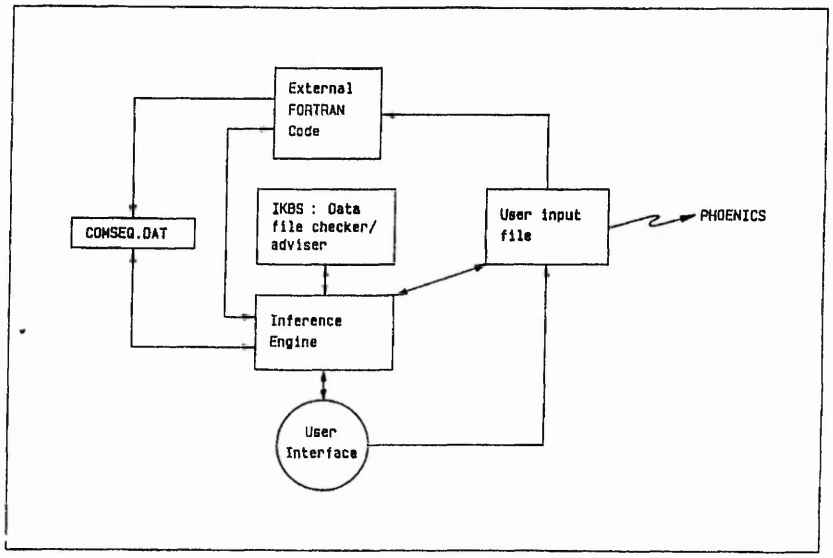


Fig. 7. Infrastructure for data file checker/adviser

can be frustrating, even for experienced users. With this in mind, a file checker/adviser was developed within the framework of an IFE to check a data file and to provide on-line advice prior to PHOENICS submission.

Commands in the data file can be entered in any order, thus allowing named variables to be used within a statement before they are declared. For example, the name "TEMP" (Fig. 8) appears as an argument within the command statement COVAL but is assigned as the name of H1 on the following line. This would present a problem if sequential checking by the IFE is to be implemented, because PHOENICS requires a solution variable or an assigned name as the second argument within the COVAL statement. Therefore, the order in which the commands are checked must be predefined before activating the IFE. This is accomplished by submitting the data file to an external FORTRAN program that generates a file, COMSEQ.DAT, which contains the commands in a pseudo-sequential checking order. The entries in COMSEQ.DAT (Fig. 9) are arranged in modules each of which contains three elements. The first two elements identifies the PHOENICS commands under consideration and the number of occurrences. The third indicates the line numbers where the command could be found. The response time of the system appeared to be considerably increased if data is continually accessed

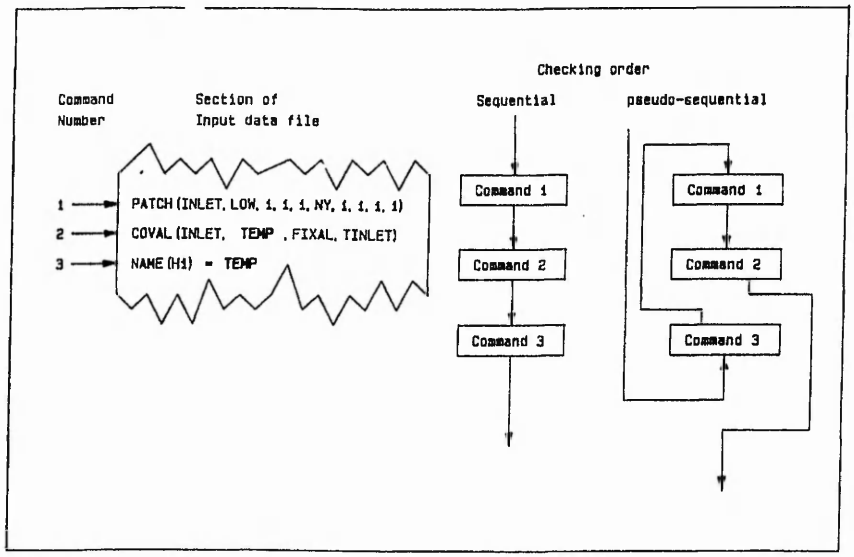


Fig. 8. Simplified overall program structure of PHOENICS

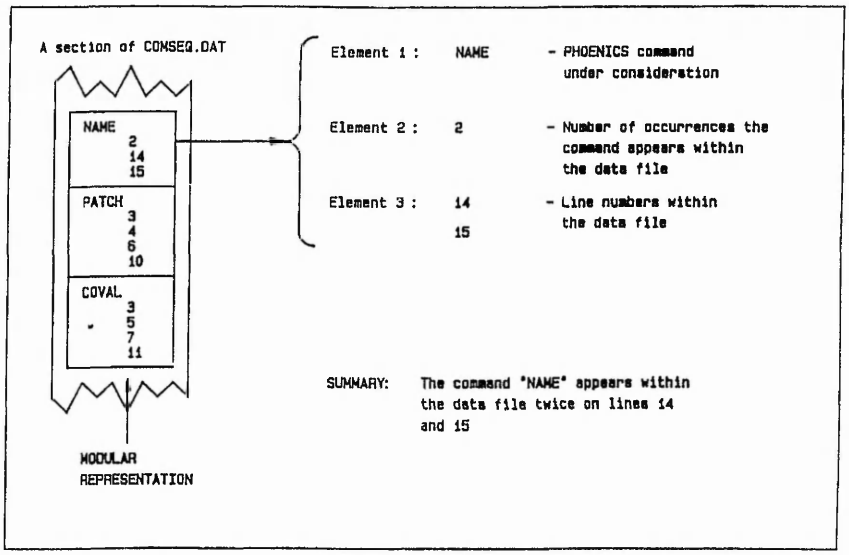


Fig. 9. An example of COMSEQ.DAT relating to Fig. 8

from external files. Consequently the data within COMSEQ.DAT is initially read and then stored internally, as the checking order, within a list object.

The data entries could be in the form of purely numeric values, mathematical expressions or a combination of both, for example:

GRDPWR(X, 10, 0.5, 1.0) ---------- Declares NX = 10
GRDPWR(Y, 10, 0.3, 1.0) ---------- Declares NY = 10

PATCH(INLET, LOW, NX/2 + 1, NX, 1, NY, 1, 1, 1, 1)
PATCH(INLET, LOW, 6, 10, 1, 15, 1, 1, 1, 1)

Substitution of the mathematical expressions for their numeric values is automatically performed within PHOENICS. However, the file checker/adviser within the IFE requires the development of a mathematical interpreter, which implements a combination of locally generated lists and recursive procedures, to evaluate any mathematical expression.

The information relating to the commands, and their validity, is represent.. sing the classical production rule technique, of the form ...

if [condition] then [conclusion]

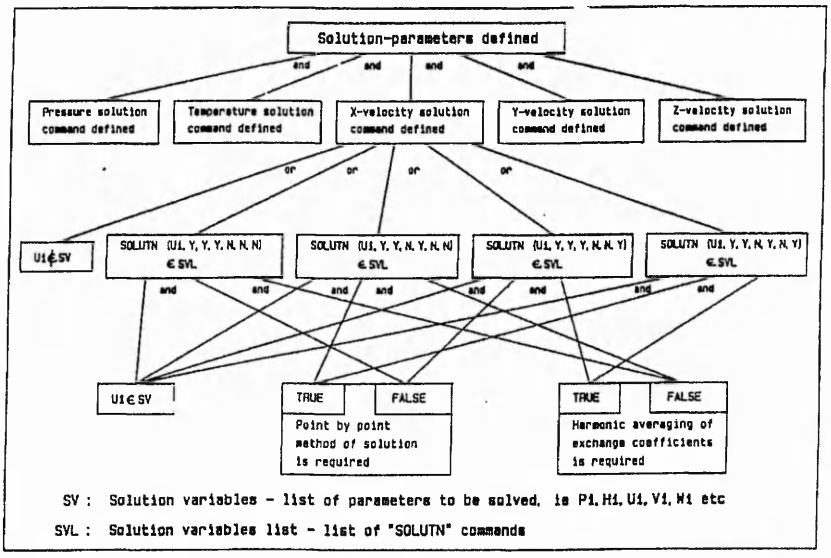


Fig. 10. Knowledge structure for inferring "SOLUTN" commands

... where the conclusion, in the majority of cases, initiated a message within an error list.

At the end of a checking process, the list identifying all the errors would be presented to the user in a report format. Using this the user could either alter the commands interactively or modify the data file independently.

3.4. Laminar jet impingement IKBS data file generator

Figure 6 shows the data file generator to comprise of two separate KB; the problem specific, or scenario, and domain specific KB. The problem specific KB has the ability to contain predefined values which allow generic parameters to be subsumed within the domain specific KB. For example the number of dimensions which defaults to 2, in the scenario KB, indicates that the axial and radial velocities, w and v, are to be assigned the appropriate SOLUTN command. Furthermore, the data file generator utilises external FORTRAN code, which is supplied with informtion from the system, to automatically mesh the geometry supplied by the user. The files produced by the FORTRAN code contain the mesh topology and the boundary condition definitions. Further information required by PHOENICS which specify the solution parameters are established through inferencing on the domain specific KB.

The jet impingement application requires specific input data relating to the type of confinement which allows the number of boundary conditions to be inferred, which is contained within the scenario KB. The knowledge/rules which are entered into an object frame within the ruleset slot, are represented in a modular format for easy modifications. The process of assigning values to an object might require the knowledge manager to search different rulesets in order to establish further object values. The inferred objects thus formed are constantly used for subsequent inference. Figure 10 is an example of a knowledge structure for inferring the solution variables required and the selection of the appropriate commands.

4. CONCLUDING REMARKS

An expert system environment, LEONARDO, was used for the development of an IFE to a commercially available CFD package, PHOENICS. The IFE presently allows manually generated data files to be checked for possible errors and gives advice on appropriate corrections. It also permits users to enter into a question and answer session which automatically generates the data file for a flow analysis problem. The techniques employed in knowledge representation consist of lists, production rules, classes and procedures all of which are implemented intrinsically within rulesets. These combinations have led to improved responses through optimised external file accessing, and have also reduced the number of defined variables for specific PHOENICS commands. Statements for defining a CFD problem have been generically categorised into lists. Depending upon the users responses the inference process within the rulesets allow the specific definition of commands to be generated. On completion of user consultation these commands are written using procedural language to an external file ready for submission to PHOENICS.

5. FUTURE DEVELOPMENT

The IFE in its present form allows repetitive numerical modelling of laminar jet impingement applications using PHOENICS. Future development will allow turbulent flow simulations to be included. Attempts are being made to embed further KB into GROUND.FOR, a subroutine within PHOENICS available for user developed FORTRAN code. Embedding an IKBS into the GROUND.FOR routine, as shown in Fig. 11, would allow the system to monitor the solution algorithm of PHOENICS on a pseudo-real-time basis. This has not, as yet, been implemented and the problems cannot be foreseen. However, a preliminary assessment has been carried out and the overall concept is considered feasible.

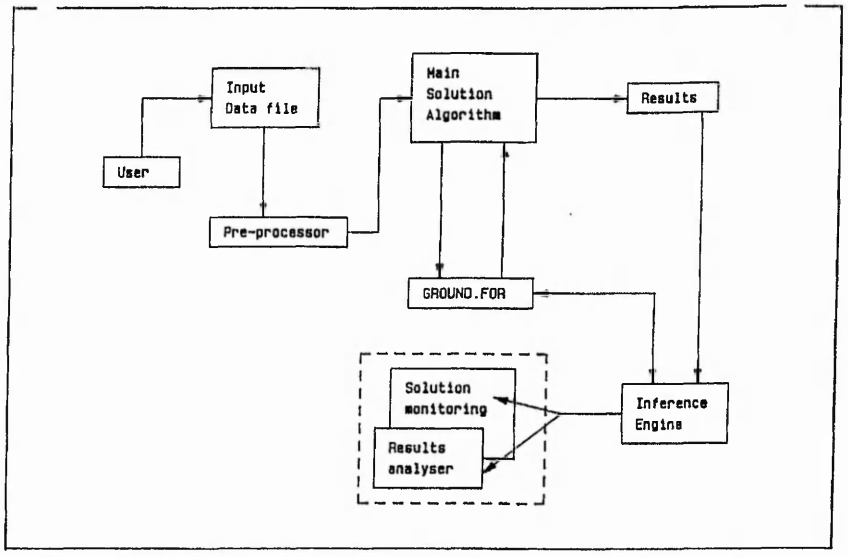


Fig. 11. Proposed future developments

A KB dedicated to analysing the results for continuity and other such requirements would assess the validity of the solution. If a problem needs to be repeated, for example to assess grid independency, the system would alter the data file for resubmission.

Intelligent grid optimisation using an iterative approach is also being considered with the view to incorporating the technique into a KB. This would allow the system to advise the user on a suitable mesh specification ensuring a grid independent solution.

ACKNOWLEDGEMENTS

The work was carried out in the Department of Mechanical Engineering using computer facilities provided by the Polytechnic's Computer services. The research assistantship held by Mr. S. L. Hartle is funded by the Polytechnics and Colleges Funding Council PCFC), under their special research initiative, and is gratefully acknowledged. Thanks are also due to Creative Logic Ltd., for the technical support provided.

REFERENCES

1 MARC User information manual, available from MARC Analysis Research Corporation, 260 Sheridan Avenue, Court House Plaza, Suite 200, Palo Alto, California 94306

2 Bennett, J. S. and Englemore, R. Sacon. A Knowledge-Based Consultant for Structural Analysis. In: *Proceedings of the sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan, 1979, 47–49

3 PHOENICS, Parabolic Hyperbolic Or Elliptic Numerical Integration Code Series, Concentration Heat And Momentum Limited, CHAM, Bakery House, 40 High Street, Wimbledon, LONDON, SW19 5AU, England

4 Bundy, A. *Intelligent Front Ends in Expert Systems*. Pergamon Infotech State of the Art Report "Expert Systems", Pergamon Infotech Ltd., 1984

5 Hayes-Roth, F., Waterman, D. S. and Lenat, D. B. (eds.) *Building Expert Systems*. Addison-Wesley Publishing Company, Inc., 1983

6 Tong, S. S. Design of aerodynamic bodies using Artificial Intelligence/Expert System technique. *AIAA paper 85-0112, American Institute of Aeronautics and Astronautics*, Aerospace Sciences Meeting, 23rd, Reno, NV, January 14–17, 1985

7 Wiess, S., Kulikowski, C., Apte, C., Uschold, M., Patchett, J., Brigham, R. and Spitzer, B. Building expert systems or controlling complex programs. In: *Proceedings of the American Association for Artificial Intelligence*, 1982, 322–326

8 Barstow, D., Duffey, R., Smoliar, S. and Vestal, S. An overview of PHINIX. In: *National Conference on Artificial Intelligence, American Association for Artificial Intelligence*, Pittsburgh, Pennsylvania, August, 1982, 367–369

9 Uschold, M., Harding, N., Muetzelfeldt, R. and Bundy, A. *An Intelligent Front End for Ecological Modelling*. Research paper 223, Dept. of Artificial Intelligence, Edinburgh University, 1984

10 Tangen, K. and Wretling, U. Intelligent Front Ends to Numerical Simulation Programs. In: Bramer, M. A. ed. *Research and Development in Expert Systems III*, 1986, 224–265

11 Fink, R. K., Callow, R. A., Larson, T. K. and Ransom, V. H. *ATHENA AIDE: An Expert System for ATHENA code input model preparation*. Idaho National Engineering Laboratory EG and G Idaho Inc., Idaho Falls (USA), 1987, 7p

12 Uzel, A. R., Edwards, R. J. and Button, B. L. A study into the feasibility of an intelligent knowledge based system (IKBS) in computational fluid mechanics (CFM). *Engineering Applications of Artificial Intelligence*, 1988, Vol. 1, September, 187–193

13 Mehta, U. B. and Kutler, P. *Computational Aerodynamics and Artificial Intelligence*. National Aeronautics and Space Administration, 1984, NASA Technical Memorandum 85994

14 Mehta, U. B. Knowledge based systems for computational aerodynamics and fluid dynamics. In: Kowalik, J. S. ed. *Knowledge Based Problem Solving*, 1986, 183–212

15 Xi PLUS. Expertech Ltd., 172 Bath Road, Slough, SL1 3XE

16 EGERIA. Intelligent Systems International Ltd., 11 Oakdene Road, Redhill, Surrey, RH1 6BT

17 CRYSTAL. Intelligent Environments Limited, Northumberland House, 15–19 Pertersham Road, Richmond-Upon-Thames, Surrey, TW10 6TP

18 LEONARDO. Creative Logic Ltd., Brunel Science Park, Kingston Lane, Uxbridge, Middlesex, UB8 3PQ

19 Patankar, S. V. *Numerical Heat Transfer and Fluid Flow*. McGraw-Hill series in Computational Methods in mechanics and thermal sciences, 1980

Engng Applic. Artif. Intell. Vol. 4, No. 5, pp. 385–392, 1991
Printed in Great Britain.

0952-1976/91 $3.00+0.00

Contributed Paper

Development of an Intelligent Front End: An Experience

K. JAMBUNATHAN
Nottingham Polytechnic

E. LAI
Nottingham Polytechnic

S. L. HARTLE
Nottingham Polytechnic

B. L. BUTTON
Nottingham Polytechnic

This paper describes the techniques used in the development of a prototype Intelligent Front End (IFE) for a Computational Fluid Dynamics (CFD) package. The prototype was developed using a commercially available Expert System (ES) shell, LEONARDO, on an IBM PC-AT compatible. The experience has highlighted the inadequacies of attempting to use LEONARDO for the creation of a practical IFE, and as such led to the development using a traditional Artificial Intelligence (AI) language, LISP.

Keywords: Intelligent front ends, IFE, computational fluid dynamics, CFD, PHOENICS, expert systems, ES.

INTRODUCTION

Numerical simulation packages

Within the engineering industry the use of numerical simulation packages, such as finite element, boundary element or finite difference schemes play an extremely important role in computer aided design. The recent emergence of cheap, yet powerful, microcomputers has enabled relatively small companies to access comprehensive Computational Fluid Dynamics (CFD) packages. CFD modelling of physical phenomena can be an extremely complex procedure and it usually requires specialist expertise and familiarity with the package to establish a working model. The generation of an input data file to a CFD package can be cumbersome, and simple modifications usually require extensive alterations to the format. These modifications can be very susceptible to catastrophic failure due to the enormous potential for human errors in typing or a momentary lack of concentration. This risk increases directly with the size of a data file which is usually large in a realistic problem. The data files contain information relating to the geometry, boundary conditions, properties and solution parameters associated with the analysis. In common with other numerical schemes most CFD packages tend to be of a generic nature, thus allowing numerous permutations of analyses to be performed. For example a CFD package might be able to consider laminar/turbulent flows, heat/mass transfer and chemical reaction processes. The availability of a number of options for the user to choose from increases the number of commands he may have to enter, each of which informs the main source code to either include or omit a particular option from the analysis, thus limiting the number of variables the program needs to solve. Clearly the marketability of the software package relates to its versatility to model a variety of different classes of problems. Even though the availability of CFD packages is increasing, their popularity and potential market is yet to be fully realised, especially by small companies. This is mainly because of the costs involved in releasing engineers to attend the necessary training courses to become proficient with the package, and the need for these engineers to have at least a basic understanding of the processes involved in order to get

Correspondence should be sent to: K. Jambunathan, Department of Mechanical Engineering, Nottingham Polytechnic, Burton Street, Nottingham NG1 4BU, U.K.

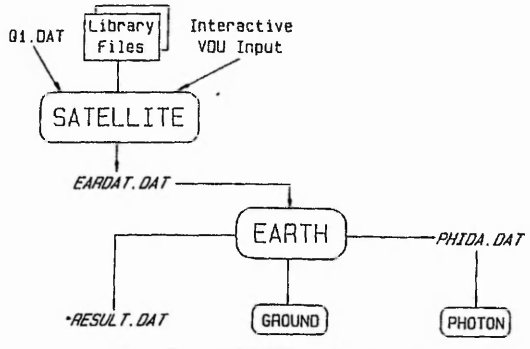


Fig. 1. The PHOENICS environment.

the full benefit from the courses. The time required to become familiar with a numerical stress analysis package[1] is anything up to 1 year, depending on the ability of the user.[2] This timescale is typical for most software packages and experience has shown this to be so for PHOENICS,[3] which is being used as an example for this work and to which an Intelligent Front End (IFE) is being developed.

Computational fluid dynamics

The processes of heat/mass transfer, chemical reactions and fluid flow pervade all aspects of human life. These processes can be observed in engineering (combustion engines, aircraft, rockets, heat exchangers, air conditioning plants, etc.), the natural environment (pollution, storms, floods, fires, etc.), and in the human body (blood flow, temperature control via heat and mass transfer). As a consequence of the enormous influence the processes have on human life, it is essential to be able to predict their behaviour in order to deal with them effectively. Extensive research throughout the world, over many years, has yielded many powerful numerical simulation packages. The basis of such numerical packages lies with the solution of the governing differential equations of fluid flow and heat/mass transfer. The most popular numerical techniques are finite element, finite difference and finite volume. PHOENICS is a general-purpose finite volume package designed for the simulation of fluid flow, heat/mass transfer and chemical reaction processes. The program structure of PHOENICS is basically divided into two sections: the preprocessor, SATELLITE, and the solution algorithm, EARTH (Fig. 1). A user defines the problem using the PHOENICS Input Language (PIL) to generate an input data file which will be interpreted and compiled by the preprocessor to form a data file EARDAT.DAT, which is read by EARTH. After the analysis has been completed the results are written to two files, RESULT.DAT and PHIDA.DAT. The former is used for a tabular presentation of the resulting flow field, whereas the latter is used for restarts and post processing packages for graphical output.

Intelligent front ends

Under the Alvey programme which commenced in 1983 five key technology areas were highlighted, one of which dealt with Intelligent Knowledge Based Systems (IKBS). Within this key technology existed nine research themes: intelligent front ends, intelligent computer-aided instruction, expert systems, natural language understanding, image interpretation, declarative languages, inference and knowledge representation, parallel architectures, and intelligent data base systems. A succinctly modified version of the original SERC/DoI definition of an IFE is "An intelligent front end (IFE) is a kind of expert system. It is a user-friendly interface to a complex software package which would otherwise be technically incomprehensible and/or too complex to be accessible to many potential users".[4] As part of the Alvey project there have been two workshops on IFEs at which discussions on the overall research areas took place.[5,6]

An IFE, as shown in Fig. 2, is designed to remove the complexities associated with entering a problem specification to a numerical simulation package. IFEs differ significantly from conventional data-entry techniques in that they are able to explicitly define a user's problem in the terminology required by the package. This is performed by asking the user questions, structured in English, that allow the IFE to create the necessary commands to correctly specify the complete problem to be analysed. There exist several developed IFEs for various application packages, examples of which are given in Refs 7–14.

Expert system shells

Artificial Intelligence (AI), and more specifically ESs, usually contain logical symbolic processing as well as conventional computational techniques.[15,16] Recent development of ES shells has allowed the integration of symbolic and numeric processing for ES applications. Several commercially available ES shells were considered for the development of the IFE,[17–19] but the ultimate restrictions of cost, *potential versatility*, and availability made LEONARDO[20] appear the most

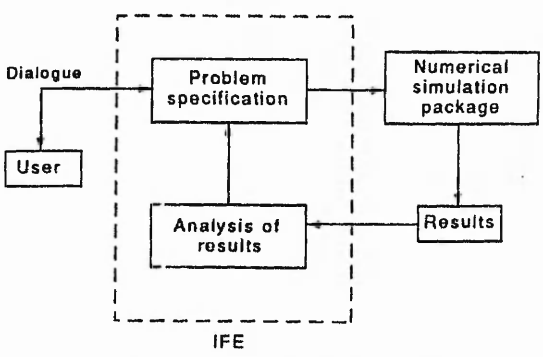


Fig. 2. The typical structure of an IFE.

favourable in this case. LEONARDO is written in FORTRAN and utilises numerous knowledge-representation techniques including frames, production rules, quantification rules, lists and a procedural language. It also allows interfacing to externally executable files and various database files. Forsyth[21] reviewed LEONARDO version 3.00, level 3. The initial impression that was conveyed gave a rather glowing report on the facilities contained therein. Indeed, any new user to LEONARDO would easily come to the same conclusion. However, after eighteen months exposure to the system, during which the shell was subjected to a series of rigorous tests, the initial views became very dim and cloudy due to its inherent slowness and unreliability, the latter included unexpected spontaneous corruption of knowledge bases which were only recoverable from previously saved files.

PROTOTYPE PHOENICS INTELLIGENT FRONT END

PHOENICS has had no interactive front end until recently. The PC version has been given a limitedly flexible menu-driven interface which allows semi-complex problems to be defined. Further, the mainframe version has a much-reduced menu interface which proves to be painfully slow and does not afford user-friendliness. Both menu systems assume that users have some prior knowledge on the use of the package and that they are conversant with the terminologies used in CFD and PHOENICS.

Finite element packages such as FLOTRAN[22] have been developed with the integration into existing Computer Aided Design (CAD) software in mind, the limiting factor being the data transfer between one package and another. However, Concentration Heat And Momentum (CHAM) Ltd,[3] who develop PHOENICS, appear to have ignored compatibility with existing CAD software. File conversion programs could be written only if the structure of the various PHOENICS output files, EARDAT.DAT and PHIDA.DAT shown in Fig. 1, are known. Indeed, FEMVIEW[23] have recently been working with CHAM to develop independent pre- and post-processing facilities for PHOENICS, as their collaboration allows them access to this information. The software developed by FEMVIEW is not marketed with PHOENICS.

The PHOENICS IFE attempts to assist a user in the generation of the mesh, by applying heuristics where appropriate, and in the process of defining the problem to be analyzed, having once established the user's proficiency with PHOENICS. The IFE reduces the need to become familiar with the command syntax required in order specify a problem; however, it does not negate the need for an understanding of fundamental fluid mechanics.

The prototype IFE that has been developed thus far has been written using a commercially available expert system shell, LEONARDO (Versions 3.17, 3.18 and 3.20). The initial stages of development saw rapid progress towards a working system. However, this progress could not be maintained as the demands of an IFE caused the limitations of the software to emerge. Inadequate validation of LEONARDO gave rise to a large number of "bugs" in the software that caused havoc at certain stages of the development. Implementation of pseudo-lists, which are character *1200 strings within the FORTRAN code, created the situation where it was not possible to store numeric values within the structure. This resulted in a numeric-to-string converter code having to be created, further lengthening the response times. Excessive disk accessing proved to be the greatest problem, whereby simple tasks could take considerable time to perform. This is exemplified in the evaluation of a simple string expression, "2+(26.47/49)^3" using a specifically developed mathematical parser (see below) which required 15.25 s to complete when executed through LEONARDO. This is in contrast with an execution time of 1.51 s when run directly in DOS, based on the same expression. Figure 3 shows the initial infrastructure on which the development has been based.[24]

The infrastructure of the IFE centres around the inference engine and its interaction with the knowledge bases and supplementary external FORTRAN routines. The latter are used to improve the performance of the system by reducing response times through not constantly accessing overlay files when using internal LEONARDO procedures to perform complex calculations. The knowledge bases have two distinct roles: a data file checker and a data file generator. The data file checker[25] is aimed at partially experienced users of PHOENICS who are capable of creating a data file which is then checked prior to submission to PHOENICS.

The data file generator, which assists the user in the generation of a data file through an interactive session, has two sub-knowledge bases: first, the problem-specific knowledge base which contains rules relating to the limitations of the system given a specific application. Second, the domain-specific knowledge base contains rules relating to the syntax of the PHOENICS commands.

The knowledge for the IFE was obtained from three different sources: practical experience with PHOENICS as a user; directly conversing with experienced users; and by acting as a pseudo-expert when supervising and advising inexperienced users.

Throughout the development of the IFE, a number of limitations were identified when representing knowledge within LEONARDO, and as such various techniques were created to aid such representation.

Further, it was necessary to integrate into the data file checker a mathematical parser in the form of an external FORTRAN code. Two applied techniques are described in the following sections.

APPLIED TECHNIQUES

Mathematical parser

Parsing with respect to AI usually refers to analyzing natural language. Clocksin and Mellish[26] introduced the concept of parsing using PROLOG grammar rules to study the structure of an English sentence. However, the main thrust of the problem concerned here does not include parsing English sentences, but mathematical expressions. Parsing of mathematical expressions can be considered as being an extremely important facet of the IFE. The application of the parser is two-fold: firstly, by directly evaluting an expression within a command where a numeric value should reside; and secondly, by applying the necessary equations that could be stored in a list form to aid the storage of information within list structures.

The need for a mathematical parser within an IFE manifested itself from the development of an Intelligent Data File Checker (IDFC).[25] Within the PHOENICS data file it is possible to insert mathematical expressions, using previously declared variables, where numeric values should reside. The PHOENICS preprocessor handles the necessary transposition of variables and deals with the subsequent calculations. However, sequential reading of the data from within the IFE would instantiate a text string where a numerical value is to be expected. This would cause the system to fail unless an equivalent numeric value could be calculated, hence the need to develop the mathematical parser.

In order to successfully implement mathematical parsers it is necessary to continually store variables and their associated values within a list, whereby transposition of variables for their values in an expression would facilitate direct evaluation. Essentially, a mathematical parser reduces an expression into the fundamental components of operators and operands, and then proceeds to determine their values. This dissection of an expression involves delimiting operators and operands within the expression. Assuming an expression is given by:

$$NY+(WIN/NZ)\hat{}3,$$

with the variable-value list containing the following information:

$$NY,20,NZ,49,WIN,26.47,\ldots\ldots \text{ plus others.}$$

Dissecting the expression and then delimiting it with commas produces the following:

$$NY,+,(,WIN,/,NZ,),\hat{},3.$$

Substitution for the variables is performed by removing the appropriate value from the variable-value list and inserting it into the expression. In the example given this would result in:

$$20,+,(26.47,/,49,),\hat{},3.$$

Standard precedence rules and associativity laws apply

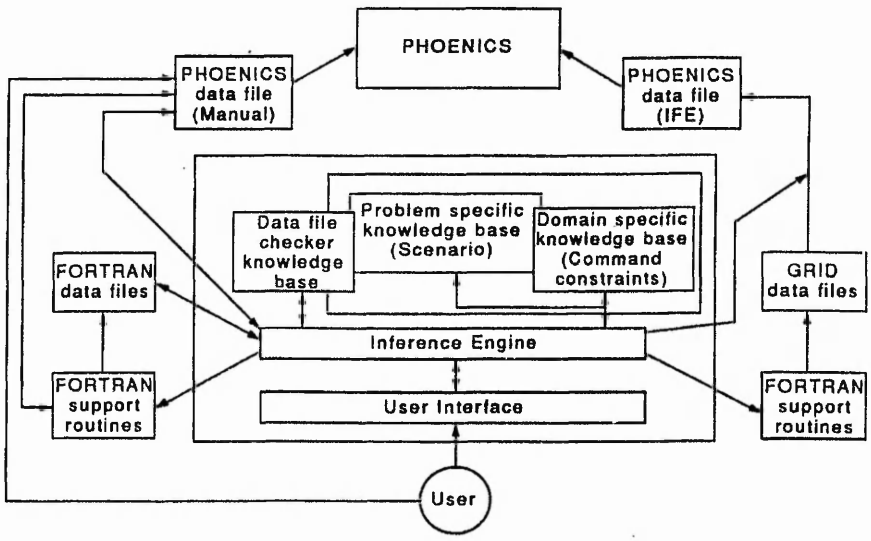


Fig. 3. Preliminary infrastructure of the PHOENICS IFE.

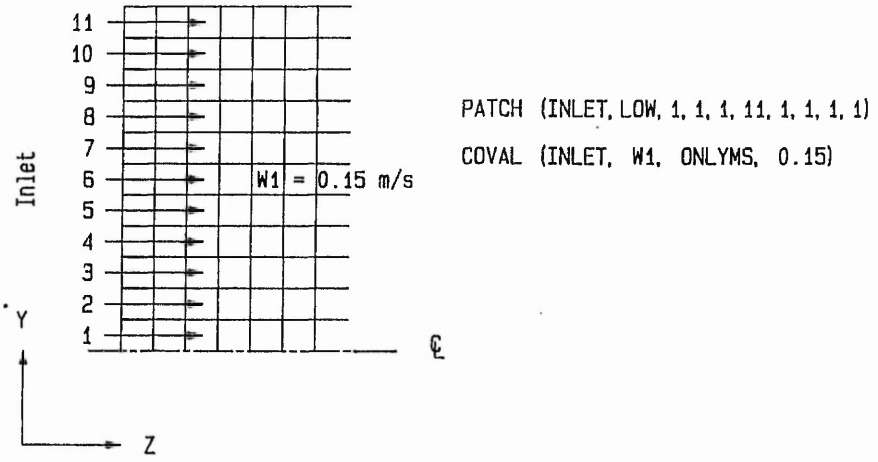


Fig. 4. Specification of an inlet boundary condition for a flat velocity profile.

to the evaluation process where the expression is collapsed into a single resulting value. Recursion would be the ideal technique to use on such a problem, which would be best implemented in C or LISP; however FORTRAN77 does not allow recursion. Therefore, subroutine looping was used, which enabled the calculations to be performed.

The parser highlights an operator and removes the associated operands from the expression, together with the operator, and calculates the result. The answer is then fed back into the expression in place of the extracted information. If parentheses are present then the innermost set will be determined first and a gradual outward growth leads to the final answer. In the example given the final value is determined in the following manner:

20,+, 0.5402,^,3

20,+, 0.15764

20.15764.

IFE information storage within list structures

It has been mentioned above that LEONARDO uses pseudo-lists. This, when compared to LISP, heavily restricts the developer in terms of flexibility and availability of potential information-storage techniques. Limited list-processing functions within LEONARDO reduce its flexibility, and complex list structures such as lists within lists are not available. These have proved to be invaluable for the LISP development.[27] Complex list structures can be used to store, in a modular form, the associated information for the boundary conditions. In order to effectively utilize the lists within LEONARDO it was necessary to establish LISP-like structures so that multiple data blocks could be stored in one list, thus establishing pseudo-lists within lists. This approach was adopted for storing information relating to the boundary conditions required for a fluid-flow problem specification.

Within PHOENICS each boundary condition requires a set of commands which (a) locate the named regions within the meshed domain using cell numbers and surface notation, and (b) specify the applied condition. For example, the boundary condition commands required to specify an inlet velocity of 0.15 m/s at entry to a mesh defined in Fig. 4 are as follows:

PATCH(*INLET*,*LOW*,1,1,1,11,1,1,1,1)

COVAL(*INLET*,*W*1,*ONLYMS*,0.15)

The cell numbers within the *PATCH* command, represented by the last eight arguments, are generated by the grid-generation routines. Information that is required by the routines is the absolute co-ordinates of the boundary and the following specifications:

Patch name : *INLET*

Patch type : *LOW*

Dependent variable : *W*1

PHOENICS coefficient : *ONLYMS*

PHOENICS value : 0.15

Information is then stored using the template shown in Fig. 5. The "Name" is the index for each boundary module within the list and it is possible to directly access information pertaining to the boundary conditions using the equations given in the Appendix.

Information stored within the lists is passed to the external grid-generation program which creates all of the necessary commands in order to specify the grid to be used and the associated boundary conditions.

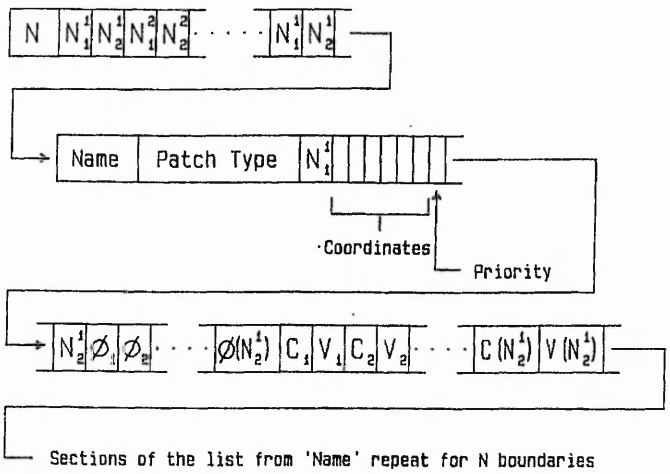


Fig. 5. Conceptual list structure.

Feasibility study of integrating the mathematical parser and information storage within list structures

A feasibility study into the possibility of storing equations of the kind shown in the Appendix and using them for locating information within lists was performed. The potential benefit would be to reduce the amount of code required to locate information from several lists, through developing generic routines to operate upon the stored equations. The concept involves generating an initial module at the front of the list which contains the equations for that list (Fig. 6). The equations in the Appendix would have to be modified in order to account for the length of the equation module. The equations would then be retrieved and used by the parser to calculate the position of a specific item of information. This technique was not implemented because of the inherent slowness experienced with LEONARDO, and the potential to exceed the allowed number of characters within the pseudo-lists.

CONCLUDING REMARKS

A preliminary Intelligent Front End (IFE) to a commercially available Computational Fluid Dynamics (CFD) package, PHOENICS, based on a commercially

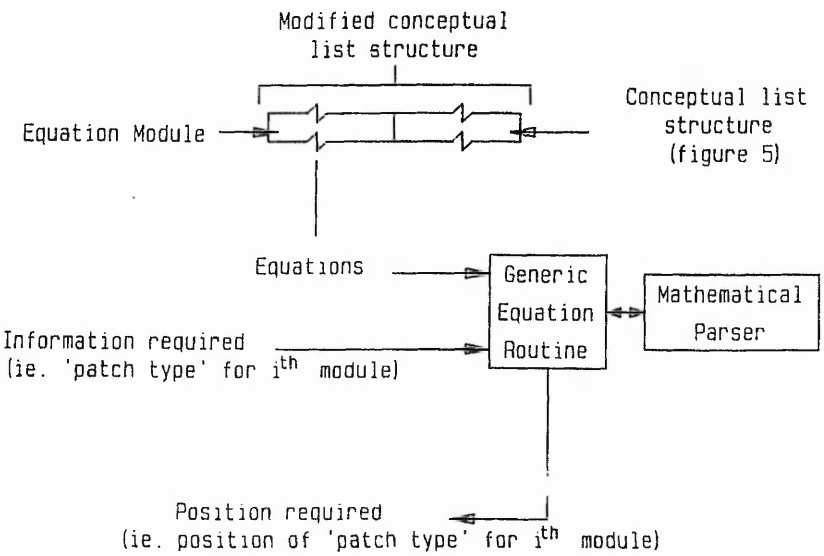


Fig. 6. Integration of the mathematical parser and information storage within list structures.

available Expert System (ES) shell, LEONARDO, has been developed. This development, whilst confirming the feasibility,[28] enabled the inadequacies of using LEONARDO for such applications to be highlighted. Inherent bugs, slowness and inflexibility in knowledge representation necessitated the abandonment of the concept of developing a practical IFE using an existing shell. Experience in the use of techniques such as a mathematical parser to handle algebraic expressions, and the use of pseudo-lists to facilitate modular information storage within LEONARDO, established a foundation upon which redevelopment of the IFE using C and LISP has commenced.

Creating pseudo-lists within the list structures of LEONARDO partially simulated the list-processing available within LISP. However, the amount of data required to perform this simulation was increased because of the necessary indexing values required.

The unreliability of LEONARDO proved to be a major problem because of the frequent occurrence of software bugs that were located. Reliance upon software support by the supplier is therefore critical. However, this problem can be eliminated due to the self-containment of software development using LISP and as such debugging facilities, when required, are available in-house.

To date the decision to use LISP for future development of an IFE instead of using an existing shell looks promising.

Acknowledgements—The work was carried out in the Department of Mechanical Engineering using computer facilities provided by the Polytechnic's Computer services. The research assistantship held by Mr S. L. Hartle is funded by the Polytechnics and Colleges Funding Council (PCFC), under their special initiative, and is gratefully acknowledged. Thanks are also due to Creative Logic Ltd. for their technical support. Figures 2 and 3 are used with the kind permission of Computational Mechanics Publications.

REFERENCES

1. MARC User information manual, available from MARC Analysis Research Corporation, Palo Alto, CA.
2. Bennett J. S. and Englemore R. SACON: A knowledge-based consultant for structural analysis. In *Proc. Sixth Int. Joint Conf. Artificial Inteligence*, Tokyo, Japan, pp. 47–49 (1979).
3. PHOENICS Versions 1.4, and 1.5.3. Parabolic Hyperbolic Or Elliptic Numerical Integration Code Series, CHAM Ltd, London.
4. Bundy A. *Intelligent Front Ends in Expert Systems*, Pergamon Infotech State of the Art Report "Expert Systems". Pergamon Infotech (1984).
5. Bundy A., Sharpe B., Uschold M. and Harding N. *Alvey IKBS Research Theme Workshop: Intelligent Front Ends*, Abingdon, England. IEE, Stevenage (1984).
6. Bundy A. (Ed.) *Alvey IKBS Research Theme Workshop: Intelligent Front Ends 2*, University of Sussex. IEE, Hitchin (1984).
7. Tangen K. and Wretling U. Intelligent front ends to numerical simulation programs. In *Research and Development in Expert Systems III* (Bramer M. A., Ed.), pp. 254–265 (1986).
8. Pang G. K. H. An intelligent front end for a control system design and analysis package. *Proc. Fourth IFAC Computer Aided Design in Control Systems Symposium*, Beijing, China, pp. 329–334 (1988).
9. Thomas G. B., Thomas R. C. and Lai C. C. An expert system interface to a suite of rotordynamic programs. *Inst. Mech. Engng Conf. Proc. Vibrations in Rotating Machinery*, pp. 621–626 (1988).
10. Clarke J. A., Rutherford J. H. and MacRandal D. M. An intelligent front-end for building energy simulation. *Working Conf. of Users of Simulation Hardware*, Ostend, pp. 165–171 (1988).
11. MacRandal D. The application of intelilgent front ends in Building Design. In *Artificial Intelligence in Engineering: Tools and Techniques* (Sriram D. and Adey R. A., Eds), pp. 361–370. Computational Mechanics Publication (1987).
12. Fink R. K., Callow R. A., Larson T. K. and Ransom V. H. *ATHENA AIDE: An Expert System for ATHENA Code Input Model Preparation*. Idaho National Engineering Laboratory EG and G Idaho Inc, Idaho Falls (1987).
13. Tong S. S. Design of aerodynamic bodies using Artificial Intelligence/Expert system technique. *Am. Institute of Aeronautics and Astronautics*, Aerospace Sciences Meeting, Reno, NV (AIAA paper 85-0112) (1985).
14. Uschold M., Harding N., Muetzelfeldt R. and Bundy A. *An Intelligent Front End for Ecological Modelling*, Research paper 223. Department of Artificial Intelligence, Edinburgh University (1984).
15. Metha U. B. and Kutler P. *Computational Aerodynamics and Artificial Intelligence*. NASA Technical Memorandum 85994, National Aeronautics and Space Administration (1984).
16. Mehta U. B. Knowledge based systems for computational aerodynamics and fluid dynamics. In *Knowledge Based Problem Solving* (Kowalik J. S., Ed.), pp. 183–212 (1986).
17. CRYSTAL. Intelligent Environments Limited, Richmond-upon-Thames.
18. EGERIA. Inference Europe, Slough.
19. Xi PLUS. Inference Europe, Slough.
20. LEONARDO. Creative Logic Ltd, Uxbridge.
21. Forsyth R. Software review: Leonardo. *Expert Systems* **5**, (2), 160–164 (1988).
22. FLOTRAN, STRUCOM—Structures and Computers Limited, Finite Element Consultants. UK and European Representatives for ANSYS and FLOTRAN, Croydon.
23. FEMVIEW Ltd. FEMGEN TO PHOENICS 1.5 INTERFACE PROGRAM AND PHOENICS TO FEMVIEW INTERFACE PROGRAM, Leicester.
24. Jambunathan K., Lai E., Hartle S. L. and Button B. L. Development of an intelligent front end for a computational fluid dynamics package. *Artif. Intell. Engng* **6**, (1), 27–35 (1991).
25. Jambunathan K., Lai E., Hartle S. L., Li H. and Button B. L. An intelligent data file checker for a computational fluid dynamics package. In preparation.
26. Clocksin W. F. and Mellish C. S. *Programming in PROLOG*, 2nd edn. Springer, New York (1984).
27. Andrews-Vogel A. *A Knowledge-Based Approach to Automated Flow-Field Zoning for Computational Fluid Dynamics*. National Aeronautics and Space Adiminstration, NASA Technical Memorandum 101072 (1989).
28. Uzel A. R., Edwards R. J. and Button B. L. A study into the feasibility of an intelligent knowledge based system (IKBS) in computational fluid mechanics (CFM). *Engng Applic. Artif. Intell.* **1**, 187–193 (1988).

APPENDIX A

Equations used for information location in pseudo-lists

$$N_1^i = 2 + 2(i-1) \tag{1}$$

$$N_2^i = N_1^i + 1 \tag{2}$$

$$\text{Index}_i = 2 + 2N + \sum_{n=1}^{i-1} (4 + \text{Min}(1, N_3^n) + 3(N_1^n + N_2^n)) \tag{3}$$

$$\text{Name}_i = \text{Index}_i \tag{4}$$

$$\text{Patch Type}_i = \text{Index}_i + 1 \tag{5}$$

$$\text{Priority}_i = \text{Index}_i + 3 + 3N_1^i \tag{6}$$

$$\phi_{ji} = \text{Priority}_i + 1 + j \tag{7}$$

$$C_{ji} = \text{Priority}_i + 2 + N_2^i + 2(j-1) \tag{8}$$

$$V_{ji} = C_{ji} + 1 \tag{9}$$

where: N = number of boundaries in the list; N_1^i = number of coordinates for the boundary; N_2^i = number of dependent variables specified on the boundary; $i = 1, 2, 3, \ldots, N$; $j = 1, 2, 3, \ldots, i-1$; $n = 1, 2, 3, \ldots, N_2^i$.

Development of an Intelligent Front End using LISP

K. Jambunathan, E. Lai, S. L. Hartle, B. L. Button

Department of Mechanical Engineering , Nottingham Polytechnic, Burton Street, Nottingham, NG1 4BU, UK.

ABSTRACT

Attempts to develop an Intelligent Front End (IFE) to a Computational Fluid Dynamics (CFD) package through a commercially available Expert System (ES) shell have proved to be unsuccessful. The inadequacies of the techniques available for knowledge representation and the manipulation within a shell environment rendered the use of the shell approach for such a development to engineering applications impractical. Nevertheless this valuable experience has helped in the redevelopment of an IFE using Common LISP. Implementation of existing techniques for customised knowledge representation which allows for the flexibility required for the IFE is described. Integrating variables, expressed through LISP structures, and facts, expressed as symbol lists, has enabled a combination of these techniques to be used effectively. Standard pattern matching techniques in conjunction with modified inferencing processes have been used to interact with multiple knowledge bases. Furthermore, integration of external C routines has enhanced the numerical computation of LISP for complex CFD mesh generation.

INTRODUCTION

The British coordinated development of Intelligent Front Ends (IFEs) stems from the early stages of the Alvey programme, discussed by Oakley and Owen [1], which commenced in 1983, and published its final report in October 1988. The Alvey programme of advanced information technology (IT) was a joint venture between three UK Government Departments (the Department of Trade and Industry, the Ministry of Defence, and the Department of Education and Science), British industry and academia. The programme was coordinated by the UK Science and Engineering Research Council (SERC). The objective was to stimulate British IT research into five key technologies, one

of which was Intelligent Knowledge Based Systems (IKBS), in response to increasing overseas competition in the field of information technology. Intelligent Front Ends was highlighted as one of the nine research themes within the overall key technology of IKBS and there has been two workshops, reported by Bundy et al. [2] and Bundy [3], on research projects related to IFEs.

IFEs can be developed for any kind of software package, examples of which are given in references 4 to 10, and are essentially designed to remove their inherent complexities and idiosyncrasies experienced by the user. To this end an IFE should interact with a user in his own language and ultimately synthesise the information obtained into the language required by the package. Computational Fluid Dynamics has progressed over the years to a stage whereby numerical flow analysis is becoming more readily available to solve complex problems. This increased availability has promoted the use of CFD, and in order to improve the user market has been a constant area of application of AI for many years, particularly in the field of grid generation, Vogel [11]. Comprehensive CFD programs have been developed which can simulate virtually any flow situation, consequently the packages are correspondingly complex especially for new/novice users. Experience in developing an IFE using an Expert System shell has been presented, Jambunathan et al. [12, 13]. The findings have led to the current research which implements a traditional Artificial Intelligence (AI) language, LISP, for the development of the IFE. Integration of C into the LISP code for complex numerical calculations reinforces the benefits of combining symbolic and numeric computation for CFD/AI applications presented by Mehta and Kutler [14] and Mehta [15].

Implementation of Common LISP, Steele [16], has shown to be a powerful language with which to develop an IFE. Established techniques for pattern matching and inferencing, introduced by Winston and Horn [17], have been used and extensively modified, where appropriate, to accommodate both variables and facts related to a specific application. Data-grabbing procedures form the information gathering operation which feeds the data driven inferencing process. An architecture of multiple rule bases allows efficient inferencing whereby unnecessary scanning of rules is minimised.

COMPUTATIONAL FLUID DYNAMICS

The processes of heat/mass transfer, chemical reactions and fluid flow pervade all aspects of human life. These processes can be observed in engineering, the natural environment, and in the human body. As a consequence of the enormous influence

the processes have on life it is essential to be able to predict the behaviour in order to deal with them effectively. Computational Fluid Dynamics (CFD) utilises the results of many years of research to aid engineers in predicting the effects of fluid flow with or without heat/mass transfer.

Commercially available CFD packages usually require the user to generate a data file to be read by the main program. Depending upon the versatility of the package and the complexity of the problem to be analysed, the size and intricacy of the data file can vary considerably. Some of the techniques used in the development of an IFE can be described with the aid of a simple CFD problem as defined in figure 1, and the corresponding CFD data file, figure 2. When defining a problem to be analysed using CFD it is possible to categorise the information to be obtained in the following manner.

Geometrical information - coordinates, connectivity.
Boundary conditions - inlet/outlet, wall boundaries.
Fluid properties - density, viscosity, turbulence model.

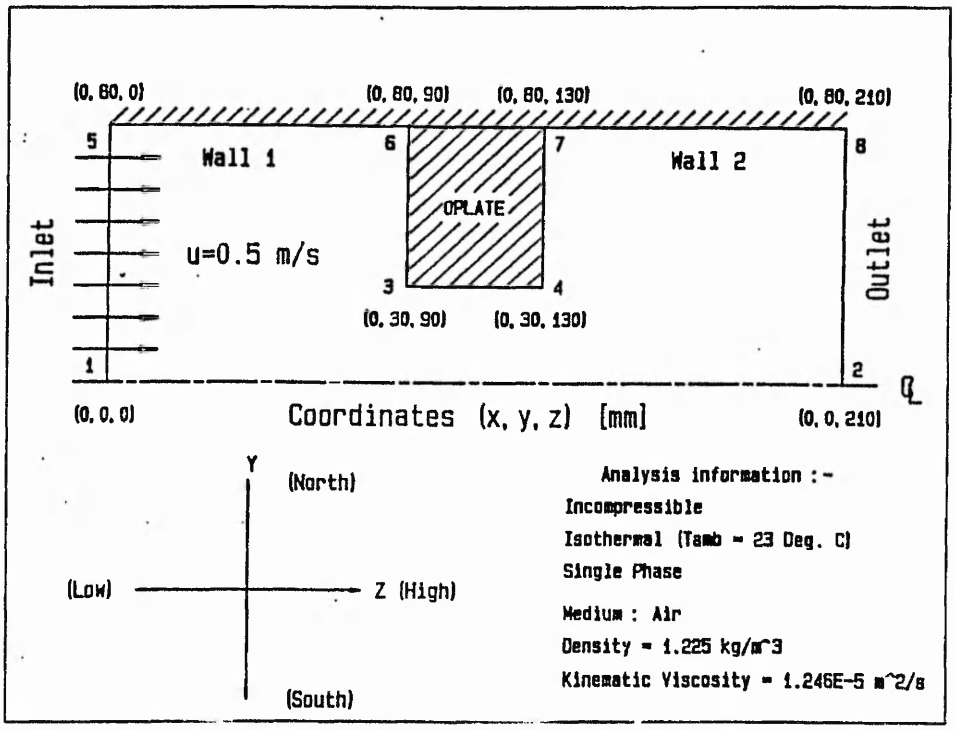


Figure 1: Simple CFD problem

```
Talk=f;Run(1,1);VDU=TTY
Text(Simple CFD problem)
Subgrd(y,1,5,0.03,1.0)
Subgrd(y,6,12,0.05,1.0)
Subgrd(z,1,10,0.03,1.0)
Subgrd(z,11,15,0.04,1.0)
Subgrd(z,16,26,0.08,1.0)
Solutn(P1,y,y,y,n,n,n)
Solutn(V1,y,y,n,n,n,n)
Solutn(W1,y,y,n,n,n,n)
Rho1=1.225
Enul=1.246e-5
Conpor(oplate,0.0,cell,1,1,-6,-16,-11,15)
Fiinit(V1)=0.01
Fiinit(W1)=0.5
Patch(inlet,low,1,1,1,NY,1,1,1,1)
Coval(inlet,P1,fixflu,Rho1*0.5)
Coval(inlet,W1,onlyms,0.5)
Patch(outlet,high,1,1,1,NY,NZ,NZ,1,1)
Coval(outlet,P1,fixp,0.0)
Patch(wall1,nwall,1,1,NY,NY,1,10,1,1)
Coval(wall1,V1,fixval,0.0)
Coval(wall1,W1,1.0,0.0)
Patch(wall2,nwall,1,1,NY,NY,16,NZ,1,1)
Coval(wall2,V1,fixval,0.0)
Coval(wall2,W1,1.0,0.0)
Lsweep=100
Relax(P1,linrlx,0.8)
Relax(V1,falsdt,0.5)
Relax(w1,falsdt,0.5)
Output(P1,y,y,y,y,y,y)
Output(V1,y,y,y,y,y,y)
Output(W1,y,y,y,y,y,y)
Iymon=10
Izmon=17
Nplt=1
Stop
```

Figure 2: CFD data file

Data-grabbing procedures form the basis for obtaining the geometrical information. Storage of nodal data and connectivity information is performed using global associativity lists (a-lists). Information such as the number of inlets, number of outlets, fluid compressibility, thermal requirements, density and viscosity are fixed for a specific application, and as such are stored as IFE variables which would be subsequently used within the rule bases. Facts relating to the geometry are stored as a list of symbol lists. The difference between variables and facts and how they interact within rules will be discussed later.

IFE ARCHITECTURE

Figure 3 shows the architecture of the IFE and illustrates the interaction of the inference engine, database, knowledge bases, C and LISP functions. The LISP functions contain all of the data-grabbing and data-manipulation procedures required to establish the initial information contained within the database. Since LISP is a symbolic manipulation language and only affords limited numerical processing power, C code was integrated into the system to allow complex CFD mesh generation, Hartle et al. [18]. The database consists of two forms of data storage: variables and facts.

Implementation of multiple knowledge bases reduces rule scanning, and allows the formation of an organised structure whereby blocks of rules relating to specific topics are self contained. Fundamentally, there are two categories of rules: data-establishing rules and data-synthesis rules. Data-establishing rules are those that enrich the database with information relating to the analysis. Data-synthesis rules are those that combine the information contained within the database to form the commands required by the CFD package, these are the last set of rules to be used by the IFE.

Once a problem has been defined, as shown in figure 1, the relevant information required for the analysis will need to be established. Information of this type is stored within the IFE as variables, and is obtained through inferencing performed on the rule bases. Data-grabbing routines are used to establish geometrical information and to assert preliminary facts. Typically, this procedure performed manually would consist of obtaining the relevant data, working through the reference manual and creating a data file using appropriate commands to fully define the problem. This essentially forms what is known as a data driven process, and is exemplified in the IFE through the primary use of forward chaining. Backward chaining is implemented to search through knowledge base inference networks.

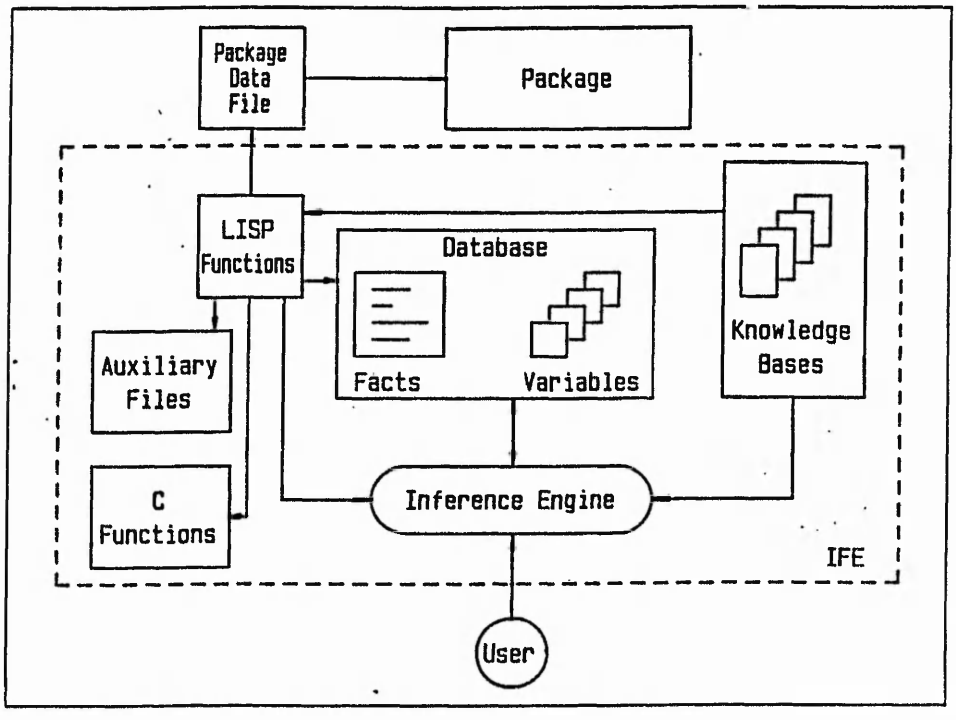


Figure 3: IFE architecture

VARIABLES, FACTS AND RULES

Variables

IFE variables contain data that is unique to a specific analysis, and as such is rigidly constrained, for example a fluid can be either compressible or incompressible, no other option is available. To constrain the user to enter certain values each variable is equipped with slots (structure keywords), one of which forms the AllowedValues. IFE variables are created from a LISP structure shown in figure 4, and are defined using a LISP macro, 'set-variable'.

Appendix A shows a selection of variable declarations. The structure keywords: Description, Type, Preface, ..., RuleBase, form the explicit declaration of the variables to be found in the rule bases. Use of the keywords allows the inference engine to establish whether, for example, to apply DefaultValues, ComputeValues or to perform inferencing upon further rule bases. Furthermore, it is possible for the rules to dynamically alter the contents of the slots. This is particularly useful under certain conditions whereby, for example, the AllowedValues of a variable need to be dynamically altered.

```
(defstruct variable
   (Description        nil)
   (Type               nil)
   (Preface            nil)
   (FixedValue         nil)
   (DisallowedValues   nil)
   (AllowedValues      nil)
   (DefaultValues      nil)
   (ComputeValue       nil)
   (Value              nil)
   (Prompt             nil)
   (Help               nil)
   (Status             nil)
   (RuleBase           nil))
```

Figure 4: IFE Variable LISP structure

Facts

Unlike IFE variables which contain unique information within a predefined structure, facts have common templates which when applied to individual data form new assertions. Factual templates contain assertion variables, $xxxx, indicating where data should reside. These assertion variables are created through a series of pattern matching routines with the antecedents and current assertions during inferencing on a specific rule. The variables are stored with their corresponding value in a LISP a-list. Considering the geometry shown in figure 1, each surface can be assigned a cardinal (North, South, High or Low) depending upon the position in space. Assuming that an a-list of data has been generated using LISP functions ...

```
(((($surface (5 6)) ($cardinal North))
                    .
                    .
 (($surface (4 7)) ($cardinal High)))
```

... which can be applied to a factual template ...

```
(Cardinal for surface $surface is $cardinal)
```

... gives the following assertions :-

(Cardinal for surface (5 6) is North)

.

.

(Cardinal for surface (4 7) is High)

Variables and facts are used simultaneously within the rule syntax.

Rules

Rules within the IFE reside in categorised rule bases. Traditional 'if...then' rules are supplemented by list quantification rules. The latter are used for list variables as opposed to class structures. A LISP macro 'remember-rule' is used to load the rules into the appropriate rule base. Antecedents within a rule are, by default, conjunctively combined, however disjunctive rules can also be introduced. The structure of the rules are shown in figures 5.1 and 5.2. The rules utilise variables and facts to assess whether to evaluate the consequents by either matching the antecedents with the facts, which reside in an assertions list, or to validate an antecedent with the variable value. Antecedents have to be syntactically accurate for the rule to be processed correctly. If the first operand in an antecedent is a variable then the statement is checked using a natural language parser (the parser is to be implemented at a later date as an interface to the IFE for experienced users). Conversely, an antecedent which can be regarded as a fact is checked using pattern matching techniques introduced by Winston and Horn [17]. Mathematical expressions within antecedents and consequents are evaluated, prior to any manipulation or evaluation, with the aid of a mathematical parser. Representing data using facts allows the system to generate many synthesised commands from a single rule. In order to fire a rule all antecedents must be correct. Appendix B shows a selection of rules from various rule bases.

Interaction of variables and facts within rules

Figure 6 shows a rule contained within a knowledge base for determining a datum parameter required for the CFD mesh generation code. The rule consists of three antecedents and a single consequent. A combination of variable and factual antecedents are used in conjunction with mathematical parsing to establish whether the rule should be fired. The first antecedent requires that the boundary-layer-thickness be greater th[illegible]. Inferencing will initially check to see if the variable boundary-layer-thickness is instantiated, if so the natural l[illegible]age parser will evaluate the antecedent. The system will prompt the user for a value if the variable is uninstantiated. The second antecedent is recognised as a fact and as such is interpreted by pattern matching routines. The routines try to symbolically match each previously made

assertion with the antecedent, and if one or more assertions are found to match then a binding list is created which contains the a-list variable and the corresponding value. For example, matching the second antecedent in figure 6 with the assertion 'minimum region size = 10' establishes an a-list consisting of (($MRS 10)). The a-list is then used for the third antecedent whereby the $MRS value is replaced and the evaluation commences. The right hand operand, being a mathematical expression, is evaluated prior to antecedent evaluation. A similar procedure is performed for the consequent provided that all antecedents are satisfied.

```
(name                          (name
 (if   (antecedent 1)           (if   (OR ((antecedent 1)
       (antecedent 2)                      (antecedent 2)
             .                                   .
             .                                   .
       (antecedent n))                     (antecedent n))
 (then (consequent 1)                (then (consequent 1)
       (consequent 2)                      (consequent 2)
             .                                   .
             .                                   .
       (consequent m)))                    (consequent m))))
```

Figure 5.1: Disjunctive and Conjunctive 'If ... Then' rules

```
(name                                (name
 (for all variable-name               (for all variable-name
    (if   (antecedent 1)                        (consequent 1)
          (antecedent 2)                        (consequent 2)
                .                                     .
                .                                     .
          (antecedent n))                       (consequent m)))
    (then (consequent 1)
          (consequent 2)
                .
                .
          (consequent m))))
```

Figure 5.2 List quantification rules

```
(remember-rule Delta-rb
'(delta-rb1
    (if   (boundary-layer-thickness > 0)
          (minimum region size = $MRS)
          ($MRS < (2 * boundary-layer-thickness)))
    (then (delta = (0.1 * $MRS)))))
```

Figure 6: Interaction of variables and facts within IFE rules

Data-synthesis rules do not assert new information or assign values to IFE variables, rather they write to an external package data file the commands necessary to completely define the problem to be analysed. The consequents within the rules have a specific structure relating to the type of command they have to synthesise. Essentially, the following template describes the synthesis consequents :-

(->Q1 *Command-template Command Argument-1 ... Argument-n*)

The symbol ->Q1 indicates that the consequent is to define a CFD package command which is to be written to a predefined file. Three type of commands can be accommodated within the system each of which is defined with the *Command-template*. Table 1 shows an example of each of the *command-templates* related to figure 2.

Command template	*Command*	*Arguments-1, ..., n*	*Example*
?[]	Solutn	P1 Y Y Y N N N	Solutn(P1,Y,Y,Y,N,N,N)
?[]=	Fiinit	V1 0.01	Fiinit(V1)=0.01
?=	Rho1	1.225	Rho1=1.225

Table 1: Example of command-templates

Progressing through the knowledge bases allows the system to obtain information relating to the analysis required by the user. This information, when used in conjunction with subsequent rules will assert further facts and instantiate or reinstantiate variables. Completion of the data-grabbing process initiates the writing of the CFD commands appropriate to the required analysis. This process would generate a data file similar to that shown in figure 2.

CONCLUSIONS

Experience using an Expert System shell has previously been shown to be inadequate for the development of an IFE to a CFD package, Jambunathan et al. [13]. The deficiencies were related to limited knowledge representation and data-manipulation facilities. To allow unlimited flexibility during the development of the IFE and the ability to create new facilities as and when required, Common LISP (Steele [16]), has been used to develop the IFE. Furthermore, information categorisation has been performed for nodal and regional data with the inherent ability to create compound list structures.

Multiple rule bases for data-establishing and data-synthesis rules are integral parts of the system which allows the establishment of self contained rules relating to categorised knowledge, and thus reduces the rule scanning required. Rule syntax using conjunctive and disjunctive antecedents incorporates factual and IFE variable information. Factual information, expressed as a list of symbol lists, has allowed command synthesis using single rules, while IFE variables have unique definitions that constrain the user to enter certain values. Furthermore, the system initially tries to evaluate a variable, through checking for fixed values, and executing any attached procedures indicated within the ComputeValue slot (structure keyword) before inferencing upon an attached rule base (if appropriate) prior to asking the user for a value.

As with any knowledge based system, rules are only fired if all of the antecedents are proved to be correct. Pattern matching techniques have been employed for assertions or factual antecedents, whereas the evaluation of antecedents that contain variables is performed using a natural language parser that interprets the contents of the condition. Mathematical parsing has been used for establishing the result of an expression prior to any evaluation of antecedents or consequents.

ACKNOWLEDGEMENTS

The work was carried out in the Department of Mechanical Engineering using computer facilities provided by the Polytechnic's Computer Services. The research assistantship held by Mr S L Hartle is funded by the Polytechnics and Colleges Funding Council (PCFC), under their special initiative, and is gratefully acknowledged.

REFERENCES

1. OAKLEY, B., and OWEN, K. *Alvey. Britain's Strategic Computing Initiative.* The MIT Press, London , England, 1989.

2. BUNDY, A., SHARPE, B., USCHOLD, M., and HARDING, N. *Alvey IKBS research Theme Workshop: Intelligent Front Ends,* Cosener's House, Abingdon, England, 26-27 September 1983, IEE. Stevenage, Herts., England, 1984

3. BUNDY, A., (Ed.). *Alvey IKBS Research Theme Workshop: Intelligent Front Ends 2,* University of Sussex, UK, 10-11 July 1984, IEE, Hitchin, Herts., UK, 1984.

4. BENNETT, J.S., and ENGLEMORE, R. 'SACON: A knowledge-Based Consultant for Structural Analysis.' *In: Proceedings of the sixth International Joint Conference on Artificial Intelligence,* Tokyo, Japan, 47-49 , 1979.

5. TANGEN, K., and WRETLING, U. Intelligent Front Ends to Numerical Simulation Programs. *In:* Bramer, M.A. (ed.), *Research and Development in Expert Systems III,* 254-265, 1986.

6. PANG, G.K.H. An intelligent front end for a control system design and analysis package. *Proceedings of the fourth IFAC Computer Aided Design in Control Systems symposium,* Beijing, China, 23-25 August, 1988, 329-334.

7. CLARKE, J.A., RUTHERFORD, J.H., and MacRANDAL, D.M. An intelligent front-end for building energy simulation. *Working conference of users of simulation hardware.* Ostend 6-8 September 1988, 165-171.

8. FINK, R.K., CALLOW, R.A., LARSON, T.K., and RANSOM, V.H. *ATHENA AIDE: An Expert System for ATHENA code input model preparation.* Idaho National Engineering Laboratory EG and G Idaho Inc., Idaho Falls (USA), 7p, 1987.

9. TONG, S.S. Design of aerodynamic bodies using Artificial Intelligence/Expert system technique. *AIAA paper 85-0112, American Institute of Aeronautics and Astronautics,* Aerospace Sciences Meeting, 23rd, Reno, NV, January 14-17, 1985.

10. USCHOLD, M., HARDING, N., MUETZELFELDT, R., and BUNDY, A. *An Intelligent Front End for Ecological Modelling.* Research paper 223. Department of Artificial Intelligence, Edinburgh University, Edinburgh, UK, 1984.

11. VOGEL,. A A. *A Knowledge-Based approach to automated flow field zoning for Computational Fluid Dynamics.* PhD Thesis, Stanford University, 1989.

12. JAMBUNATHAN, K., LAI, E., HARTLE, S.L., and BUTTON, B.L. Development of an Intelligent Front End for a Computational Fluid Dynamics Package. *Artificial Intelligence in Engineering, Volume 6 No 1, Special Issue: Intelligent Front Ends, ed. Clarke, J.A., 27-35,* 1991.

13. JAMBUNATHAN, K., LAI, E., HARTLE, S.L., and BUTTON, B.L. Development of an Intelligent Front End: An Experience. *Engineering Applications of Artificial Intelligence, Volume 4, No 5, 385-392,* 1991.

14. MEHTA, U.B., and KUTLER, P. *Computational Aerodynamics and Artificial Intelligence.* National Aeronautics and Space Administration, NASA Technical Memorandum 85994, 1984.

15. MEHTA, U.B. Knowledge based systems for computational aerodynamics and fluid dynamics. *In:* Kowalik, J.S. (ed.) *Knowledge Based Problem Solving,* 183-212, 1986.

16. STEELE, G.L., Jr. *COMMON LISP: The language.* Digital Press, 1990.

17. WINSTON, P.H., and HORN, B.K.P. *LISP.* Addison-Wesley Publishing Company, 1989.

18. HARTLE, S.L., JAMBUNATHAN, K., LAI, E and BUTTON, B.L. Aspect ratio dependent finite volume grid generation. *In preparation.*

```
APPENDIX A:  SELECTION OF IFE VARIABLE DECLARATIONS

(set-variable whole-field-variables
    :Type 'list
    :Value '(P1))

(set-variable slab-wise-variables
    :Type 'list)

(set-variable delta
    :Type 'real
    :RuleBase t)

(set-variable axis-1
    :Type 'Text
    :AllowedValues '(unused x circumferential))

(set-variable viscosity-thermal-dependence
    :Type 'text
    :AllowedValues '(required not-required)
    :DefaultValue 'not-required
    :Preface '(If you wish to simulate the change of viscosity
              within the domain depending upon the calculated
              temperatures  then  enter  <required>  at  the
              prompt.)
    :Prompt  "Viscosity   thermal   dependence   required   or
              not-required ? "
    :RuleBase t)

(set-variable number-of-inlets
    :Type 'integer
    :AllowedValues '(> 0)
    :DefaultValue 1
    :Prompt "How many inlets are within the domain ? ")

(set-variable flow-regime
    :Type 'Text
    :AllowedValues '(laminar turbulent)
    :DefaultValue 'Laminar
    :Prompt "Is the flow to be laminar or turbulent ? ")

(set-variable analysis-title
    :Type 'string
    :Preface '(The analysis title cannot be more than 40
               characters long.  The main purpose of this is
               to be able to identify the analysis.)
    :Prompt "What is the analysis title ? ")
```

APPENDIX B: SELECTION OF IFE RULES FROM VARIOUS RULE BASES

```
(remember-rule BC-RB
'((if   (Boundary name for inlet $number $nodes is $name)
        (axial inlet velocity for $name is uninstantiated))
 (then (ask axial inlet velocity for $name = real))))

(remember-rule Fluid-RB
'((if   (OR ((fluid-compressibility is compressible)
             (thermal-requirements is isothermal)
             (density is uninstantiated))
            ((thermal-requirements is thermal)
             (density-thermal-dependence is-not required))))
 (then (ask density))))

(remember-rule Fluid-RB
'((if   (thermal-requirements is thermal)
        (density-thermal-dependence is enthalpy))
 (then (density-equation AllowedValues ("A+BH" "1/(A+BH)")))))

(remember-rule Fluid-RB
'((if   (thermal-requirements is thermal)
        (density-thermal-dependence is enthalpy)
        (density-equation is "1/(A+BH)"))
 (then (ask rho1a)
        (ask rho1b))))

(remember-rule Grid-RB
'((if   (Aspect-ratio > 0)
        (Delta > 0))
 (then (run grid))))

(remember-rule G1-RB
'(grid1
 (if   (analysis-title is instantiated))
 (then (->Q1 ?[] text analysis-title))))

(remember-rule G3-RB
'(cartes
 (if   (coordinates are cylindrical))
 (then (->Q1 ?= cartes f))))

(remember-rule G7-RB
'(slab-wise-variables->Q1
    (for all slab-wise-variables
         (->Q1 ?[] solutn $value y y n n n n))))
```