1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Canadian Theses Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale
du Canada

Services des thèses canadiennes

CANADIAN THESES

NOTICE

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

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

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

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

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

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

THÈSES CANADIENNES

AVIS

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

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

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

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QU'ELLE
NOUS L'AVONS REÇUE**

**An Inferencing Procedure
for
Guaranteeing the Search Time
of a
Production-Rule Knowledge Base**

by

Haleh Vafaie, B.Eng. (EE)

**A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Engineering**

**Faculty of Engineering
Department of Systems and Computer
Engineering,
Carleton University**

June 1986

©copyright

1986, Haleh Vafaie

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-33423-1

The undersigned recommends to the Faculty of Graduate
Studies and Research acceptance of the thesis

"An Inference Procedure

for

Guaranteeing the Search Time

of a

Production-Rule Knowledge Base"

Submitted by Haleh Vafaie, B.Eng. (EE), in partial fulfillment of
the requirements for the degree of Master of Engineering.

Thesis Supervisor

Chairman,

Department of Systems and

Computer Engineering

Carleton University

July 1986

Abstract

In recent years, the trend toward the use of expert systems to solve a wide range of problems has increased. Problems requiring real-time responses have been largely left out, due to the unpredictable execution time for these systems as the description of the problem domain becomes complex. Therefore, an important issue in the design of real-time expert systems is to provide a guaranteed response time.

This thesis uses an algorithm which finds the maximum parallelism in a production-rule knowledge base. This in turn provides a means for calculating the maximum execution time, which becomes the guaranteed response time. The approach depends on a novel concept of a multiplicity of inference engines which simultaneously traverse the knowledge base. A logical architecture involving a set of concurrent inference engines for implementing the algorithm is provided.

The results of this thesis will demonstrate that the response time of the system can be guaranteed within a fixed period of time. That is, a response to the given input is achieved before this time has elapsed. This response time is dependent on the number of inference engines used, and decreases as this number increases until maximum parallelism possible is reached. Therefore, given the required response time, it is possible to evaluate the feasibility of achieving the required response, and the num-

ber of inference engines required. An analysis to show the speed-up and utilization of the architecture is also presented.

Acknowledgements

I would like to gratefully acknowledge all of the people whose assistance and support was invaluable to me in this thesis undertaking.

I would like to express special thanks to my thesis supervisor, Dr. B. A. Bowen for his relentless advice and guidance through the process of this research.

Thanks are due to my family and friends for their constant help and support. Special appreciation goes to Hamid and Houman, my wonderful brothers, and the graduate students in the Department.

Finally I would like to thank my fiance Fardiss for all his patience and support throughout this thesis.

I dedicate this thesis to my mother, Kayhan, and to the memory of my father, Ali Vafaie.

Contents

1	INTRODUCTION	1
1.1	INTRODUCTION	1
1.2	EXPERT SYSTEMS	9
1.3	DEFINITION OF REAL-TIME	12
1.4	REAL-TIME EXPERT SYSTEMS	14
1.5	THESIS RESULTS	15
1.6	EXPLANATION OF THE FOLLOWING CHAPTERS	17
2	PARALLELISM DEFINITION	19
2.1	INTRODUCTION	19
2.2	ACHIEVING REAL-TIME EXPERT SYSTEMS	21
2.2.1	PARALLELISM	23
2.2.2	METAPLANNING	25
2.3	ANALYSIS OF PARALLEL ALGORITHM	27
2.3.1	EXTENSION TO KNOWLEDGE BASES	37
2.3.2	PERFORMANCE PROPERTIES	50

2.4	ASSESSMENTS	59
2.5	SUMMARY	60
3	PARALLELISM IN A RULE-BASED EXPERT SYSTEM	63
3.1	INTRODUCTION	63
3.2	STRUCTURE OF A KNOWLEDGE BASE	65
3.3	ARCHITECTURAL CONSIDERATIONS	66
3.4	ACTUAL PROGRAM IMPLEMENTATION	71
3.5	EXAMPLE	75
3.6	SUMMARY	78
4	PERFORMANCE CONSIDERATIONS	80
4.1	INTRODUCTION	80
4.2	RESPONSE TIME	81
4.3	EXAMPLE	86
4.4	SUMMARY	91
5	SUMMARY AND DISCUSSION	92
5.1	SUMMARY	92
5.2	DISCUSSION	94
6	REFERENCES	99
7	APPENDIX A	102

List of Figures

2.1	An example data-flow graph	31
2.2	The precedence graph and table for the data-flow graph. . .	32
2.3	The precedence graph for the knowledge base in example 2.2.	45
2.4	The precedence table for example 2.2.	46
3.1	The proposed architecture for parallel inferencing	69
4.1	Graph of Speed-up and Utilization versus number of inference engines	90

Chapter 1

INTRODUCTION

1.1 INTRODUCTION

Considerable interest exists in exploiting the attributes of a knowledge-based representation for designing real-time controllers for physical systems. The knowledge base can represent the response based on a physical model of the plant, and in addition include heuristics to account for observed behaviour which is often difficult to model analytically. This knowledge base can be organized in several formats. One of the most common representations is rule-based knowledge bases or production-rules. If the knowledge base is implemented using production-rules the control algorithm can be easily modified by simply adding or deleting rules. This feature would permit both extensibility and rapid accommodation of unforeseen characteristics during commission of the software.

Examples of real-time expert systems have been applied in process con-

trol only over the last few years. The work on Hexscon (Hybrid Expert System Controller) by SRI International [10] resulted in development of a prototype real-time knowledge base control system running on a microcomputer. The hybrid Hexscon system uses both conventional logic and knowledge-based techniques. In this system, the knowledge base rather than the inference engine contains the knowledge of how various variables interrelate and how they are connected to the domain knowledge (as in PI-CON). This work showed that a knowledge based controller could be built that runs on a microcomputer and has enough capability to be effective for real-world problems.

A system that could be used for applications such as a nuclear power plant or for a NASA mission control room is the Real Time Causal Monitor (CAM) [11]. It is a special type of knowledge acquisition system used to develop causally-based real-time monitors for arbitrary physical sites. To achieve this, it requires models of knowledge elicited from the site expert, a frame-like model of the concept common to all physical sites, and an efficient and flexible solution to the problems of controlling a large symbolic system of production rule-like knowledge in real-time. Therefore, CAM is required to have an extremely high throughput run-time system. The Propagation Driven Machine (PDM) is the model used for this design. It exhibits the breadth and thoroughness of a classical production rule system without the usual problem of pattern matching and database main-

tenance.

The PICON (Process Intelligent Control) system [12] is designed to operate on a LISP machine, interfaced with a conventional distributed control system. It performs both forward and backward chaining inferences in a real-time environment. The heuristic knowledge base is built interactively using LISP. The key concepts in the design were to quickly recognize process conditions which are potentially significant, and to invoke relevant rule-sets and focus on these problem areas for diagnosis and procedural advice. Therefore, the inference engine was built to simulate the human expert's procedure. The following are some related work discussed in reference [12].

Another group from Foxboro [13] discussed the application of forward-chaining inference in a process control context. The knowledge rules consist of surface knowledge (consists of heuristics or knowledge that is experiential in nature) as well as deep knowledge (describes reasoning from the basic principles). The alarm-advisor application was considered in this research, and an example provided for the proposed application.

The work on an automated reasoning system at the Argonne National Laboratory [14] is very interesting. The system uses the deep knowledge or physical knowledge of the plant to develop the expert rules automatically. The potential use of the system is in nuclear plants. For example to permit

cooling to be directed to portions of the plant under various conditions of equipment failure.

Sakaguchi and Matsumoto [15] of Mitsubishi worked on the problem of restoring the normal operating state of an electric power system when a fault occurs. A production rule knowledge base structure was used in this research. The expert system applied surface knowledge (heuristics) to diagnose a system fault and also to determine the recovery procedure.

The work conducted by Hitachi Corporation [12] on a project concerning large scale systems encountered three main problems limiting the success of real-time expert systems. These problems were as follows:

- Lack of effective knowledge acquisition method and of a proper knowledge description scheme.
- Insufficiency of speed and shortage of memory of the computers used.
- Difficulty in detecting inconsistency in the knowledge base.

Based on the preceding, it can be seen that real-time expert systems are being explored, and the prototypes built were hybrid (use both conventional and knowledge-based techniques). Therefore, in general real-time expert systems will be hybrid, and a goal will be to maintain this modularity. The main goal will be to predict the response time of the knowledge based portions of the system [16].

A response is defined as a solution to the problem. Finding a suitable response in a set of rules can be formulated as a search problem where the task is to reach the solution node (goal node) in a graph or tree representation of the problem's state space. Arcs in the graph then represent the state transitions [3].

The two techniques used to search for this goal node are [4]:

Breadth-first : this search finds all rules containing the correct antecedents (therefore searching the branches of the tree first), then finds all the rules whose antecedents are members of the first consequence set. This method will find all the solution nodes.

Depth-first : this search finds the first rule containing the correct antecedent, then finds the first rule whose antecedents are members of the first rules consequence set. This method will find the first goal node reached.

The above searches can become time consuming and lead to time complexities of exponential order as the knowledge base grows [3]. In case of breadth-first searching for a solution node at depth 'd' with uniform k-way branching of the tree at each node, the execution time is bounded by [3]

d^{k-1} from below, and

d^k from above.

Therefore, if the depth of the solution (i.e. the goal node) is at lower levels and branching increases, the search time increases exponentially. Depth-first search has also time complexity of exponential order. Hence, using expert system design for problem domains that require real-time responses such as robotics, process control, signal and image processing would not be possible, unless some means for guaranteeing a fixed response time could be achieved.

The response time could be reduced if parallelism or concurrency could be introduced into the system. The two major ways of achieving parallelism are pipelining and multi-processing. Pipelining is achieved by separating the process into functional stages (processing circuits), called segments or processing units. Parallel processing is achieved by having independent processes in several segments at the same time [8]. Each segment only processes a portion of the data, and a final result is achieved when the data has passed through all the segments of the corresponding pipeline.

If 'm' parallel processors and an n-segment pipeline could be implemented such that all processors are busy executing, then 'mxn' concurrency is achieved in the system. If an architecture could be found that uses input to a given knowledge base as its input and runs by viewing the knowledge base as sets of rules with precedences that determine the sequence of firing, then all rules in the same precedence set are independent of each other and

could be executed simultaneously.

If the precedence and the associated sets could be found, then the maximum parallelism could be achieved. This is equal to the number of rules in the largest set. Thus, using a parallel pipeline, the whole rule-base could be traversed in a time equal to the number of precedence sets.

This thesis will provide an algorithm which finds these precedence sets. The guaranteed response time is then found from measuring the execution time of the system for the worst case (i.e. when the required solution node is at the bottom of the tree). Finally, a logical architecture for implementing this approach is proposed. The architecture produces all the possible responses (including inappropriate ones) in a fixed time. The correct response(s) could be selected by tagging the paths generated by valid antecedent-consequent links.

Implementing this design would allow us to guarantee the response time of a rule-based control algorithm. Also, using the parallelism, the response time for reaching any intermediate depth of the rule base can be predicted. In some cases, this would produce an adequate if not optimal response. This is important in many problems, for example, if conflict-avoidance advice in an airtraffic control environment is provided by an expert system, a delayed response time to obtain optimum response could be life threatening.

In this chapter, definitions of expert systems, real-time performance, and real-time expert systems are outlined. A brief explanation of the final results along with description of the approach used are presented .

At the end of this chapter the importance of real-time expert systems should be more clear, and as we go along the usefulness of such systems should become obvious. By the end of this chapter, the following will be accomplished:

In section 1.2, a brief description of an expert system and its functional parts are provided to give some common insight and also to characterize expert systems.

In section 1.3, a simple but formal definition of real-time performance is provided.

In section 1.4, a descriptive definition of real-time expert systems is generated so that a precise definition exist as a milestone.

In section 1.5, the approach used and procedure applied to achieve real-time expert systems is discussed.

In section 1.6, a short description of the remaining chapters is provided.

Chapter 2 will deal with the possible ways of achieving real-time expert systems.

1.2 EXPERT SYSTEMS

In recent years, there has been a growing interest in Artificial Intelligence. Many researchers became involved because they were interested in uncovering the amazing power of human mind, and simulating it [6]. This was approached by attempting to duplicate the thought processes of an expert as he exercised his expertise in a particular domain. A set of principles, tools, and techniques has emerged that forms the basis of knowledge-based expert systems and the corresponding field of knowledge engineering [6]. Expert systems investigate methods and techniques for building a man-machine system with a specialized problem solving expertise.

Expert systems could be considered as a subset of artificial intelligence. They restructure the problem domain, and create surrogates of a human expert by exploring some form of knowledge representation in the given domain [4].

An expert system usually consists of some form of user interface, a knowledge base, inference engine, and a global data base [4].

The function of the system can be described as follows [4]:

The inference algorithm scans the knowledge base in response to a user query according to a predefined procedure specified by the inference engine, and attempts to determine a suitable response. The global data-base is used to maintain records of successful search strings or some other information

depending on the design of the system to be used later on to facilitate the operation of the system.

The two most important elements of an expert system are the knowledge base and the inference engine, which will be discussed in more detail.

The inference algorithm or inference engine is the mechanism applied to explore the knowledge base to arrive at a conclusion. Many approaches to inference schemes can be used. However, if the knowledge base is rule-based, then the inference procedure could be forward or backward chaining depending on the definition of the input data. This means that a chain of reasoning starts with the input and moves forward or backward. These inferences could be defined as follows [4]:

Forward chaining : The chain starts from a set of inputs and moves toward a conclusion.

Backward chaining : the chain starts with a conclusion and moves backward to find the cause or input condition.

The knowledge base consists of a set of facts and relations, and if possible heuristics about a particular problem domain [4]. The facts or the formal knowledge is public information that is widely understood and agreed on by all experts (for example, the laws of physics). The heuristics are surface knowledge that is learned through experience and maturity in the field. The more heuristics included in the knowledge base the less the system

must search for a specific conclusion. The relations specify how these facts are related to each other.

* The knowledge base could be represented in several formats. The most widely used type is the rule-based knowledge base, which will be discussed in detail in later chapters.

An expert system is defined as [4]:

A computer program with at least two functional parts: a data base of relations and facts describing a particular domain of knowledge, and an inference procedure. The program is designed so that the inference procedure simulates the thought processes of an expert as he searches through his accumulated experience to formulate a conclusion.

In a well defined system, the knowledge base and the inference algorithm should model the performance characteristics of a recognized expert in the field. The knowledge base in this thesis represents the performance of a given physical plant. The expert system designed to operate a physical plant clearly has to be real-time, since most plants have unpredictable events that must have real-time responses to the environmental interactions.

The definitions of real-time systems are not unique and many are informal. Therefore, a formal definition is required. This is the subject of the

next section.

1.3 DEFINITION OF REAL-TIME

The informal and most common usage of the term "real-time" is fast, which refers to a measurement of relative speed; therefore, a system is real-time if it processes data quickly [3]. Another common usage of this term is based on recognizing that intelligent control reasoning can reduce systems processing time. This definition is gaining popularity among AI (artificial intelligence) researchers. Some other AI researchers use "real-time" to refer to systems that respond to a changing environment with changes in control strategies[3].

The preceding descriptions are only a few of the informal "real-time" definitions used by practitioners. These definitions vary in meaning and are not quantifiable. However; a quantifiable definition is required in order to establish provable results. The following is the strict definition of "real-time" based on the requirements of the above described systems.

A system acts in "real-time" if it produces a response to the environmental stimuli in a strict time limit regardless of the algorithm it employs. A high quality response, although important, is not a part of this definition, and is no more critical than any other factors[3].

The key characteristic here is that a "real-time" system must be able to

guarantee a response after a fixed time has elapsed, where the fixed time is part of the problem requirements (statement)[3].

More formally if

T_r = time at which the system requests a response

T_p = time at which the response must be completed

T_{r1} = amount of time response r1 takes to complete

Then response r1 is considered to be performed in real-time if and only if

$$T_{r1} \leq T_p - T_r$$

for any T_r and T_p where $T_p > T_r$

and if

r_{opt} = the optimal response

then if we have

$$T_{r_{opt}} \leq T_p - T_r$$

then the optimal real-time solution has been achieved.

The above definition indicates that for any system to be real-time, it has to provide a response before the expected execution time has elapsed.

The following section presents the definition of real-time expert systems based on the formal definition of real-time and expert systems.

1.4 REAL-TIME EXPERT SYSTEMS.

Many of the expert systems built to date do not require real-time response, mostly due to the fact that a design methodology was not available. It is of interest to provide a design methodology which could be used to implement systems that can provide the real-time performance.

This thesis defines a "real-time expert system" to be an expert system that provides a reasonable response (i.e. comes up with an adequate solution) in a fixed time and can predict the best solution's response time. Here the emphasis is on reasonable response or adequate solution, since in most (if not all) of the systems which require a real-time response, the response to an event may drastically change the subsequent actions. Therefore, the quality of the solution is also a question of interest.

In some systems, when the fixed time has elapsed and no adequate solution has been found, real-time response is provided by using a prestored solution [3]. This is the weakest real-time solution. This response is not the result of any plant interaction and does not provide any information

about the problem. Hence, the response is inadequate. Therefore, when referring to "real-time expert systems", it implies that an adequate response is guaranteed after a fixed amount of time has elapsed.

The goal is to achieve the strongest real-time techniques which will result or converge to an optimum solution and also provide access to their intermediate results in a fixed period of time. The technique applied to achieve this is presented in the following chapters.

1.5 THESIS RESULTS

The main problem in achieving real-time expert systems is due to the fact that usually there are several branches at each node (rules have more than one consequence). Therefore, deciding which rule to fire next to reach the desired answer is almost impossible. Hence, going through one path may not reach the desired answer and consequently the inference algorithm must backtrack to the point that the last decision was made. Therefore, if a design could be implemented to reduce or even eliminate this backtracking, then the execution time could be guaranteed, provided a suitable execution vehicle could be found.

The approach used in this thesis is to examine all possible paths at the same time. Thus, introducing parallelism into the system. The maximum parallelism is found by evaluating the maximum number of rules that could

be fired simultaneously (taking into consideration any existing certainty factors associated with the consequents and antecedents of the rules). If these rules are included in a set, there will be 'd' sets of rules (d is the depth of the knowledge base. This is equal to the longest path between any two nodes on the same branch of the graph or tree representation). The rules in these sets could be fired according to their precedences which is found by applying the algorithm presented in the following chapters.

Thus if maximum parallelism is implemented, then the backtracking problem is eliminated and the execution time is fixed. Applying the algorithm presented in the following chapters (thus, implementing maximum parallelism) and implementing a suitable architecture to execute the knowledge base, the following results could be achieved:

- The response time of the system is guaranteed by calculating the longest execution time possible for the system.
- The execution time to reach a given intermediate goal node can be calculated.
- An algorithm is provided for organizing the knowledge base for a sequential machine (one inference engine).
- Given the desired response time, the feasibility of the system could be established, and the required parallelism could be found. Implementing this parallelism, the utilization of the system and its cost can be

calculated. Therefore, the performance criteria could be determined before execution.

- The algorithm provides a means for finding redundant rules. These are the rules that are not contained in any precedence levels.
- If the system cannot reach an adequate response in the required period of time, an estimated portion of the process that could be executed in that time can be proposed.
- The precedences of and existence of goal nodes could be evaluated. This provides means for checking the knowledge base for completeness or any inconsistencies.

The following section describes the layout of this thesis for achieving the above described results.

1.6 EXPLANATION OF THE FOLLOWING CHAPTERS

Chapter 2 discusses the two design methodologies by which the real-time performance of a system could be guaranteed. These are parallelism and metaplanning. An approach to achieve real-time performance through parallelism is presented. Also an algorithm is devised to achieve maximum parallelism.

Chapter 3 deals with the implementation of a novel algorithm proposed based on the approach presented in chapter 2, and with defining the type of knowledge bases which could be used (rule-based), and a definition for these knowledge bases. A program is designed to obtain subsets of the given knowledge base that could be searched independently and in parallel. An example is provided to illustrate this achievement. Finally an architecture is proposed for implementing the suggested design.

In chapter 4, performance criteria and constraints are discussed. An example is presented to illustrate the concepts.

In chapter 5, the work is summarized and suggestions are provided for the problems and areas that this design is best applied.

Chapter 2

PARALLELISM DEFINITION

2.1 INTRODUCTION

The potential importance of real-time expert systems in general was established in chapter 1. In this chapter two global approaches for achieving real-time performance are discussed. Two design methodologies are presented that could predict the real-time response of a knowledge base. These are parallelism and real-time metaplanning [3]. Metaplanning applies meta knowledge (the ability to approach and organize a problem) to plan the operation of the system in advance. In the expert system world, metaplanning could be used to minimize the search time through a knowledge base by yielding partitions that could be independently searched for a response. The modular approach described in [16] is a pragmatic analysis to achieve this modularity. Even with this modularity only the worst case could be

predicted.

Metaplanner's most important requirement is for it to execute in real-time [3]. While this is an obvious requirement, it is difficult to achieve algorithmically in practical applications. This thesis will, therefore, only discuss this methodology as an alternative and will not go through details.

A decomposition technique has been found in [1] that divides an algorithm into independent subsections that could be executed in parallel. Parallelism can be implemented when an algorithm is independently decomposable. These decomposed parts could be executed independently to provide a reduction in execution time. An extension of this approach to rule-based knowledge bases is the design methodology that this thesis will follow to provide the guaranteed response.

In section 2.2, two global approaches for achieving real-time systems are presented. This leads to the definition of the above mentioned design methodologies. The material in this section was extracted from reference [3].

In section 2.3, the procedure that decomposes a given algorithm into independent sections is reviewed, along with its performance properties. This procedure is modified for application to knowledge bases.

In section 2.4, an assessment of the scope of this approach is presented.

This will provide us with the general cases.

Chapter 3 discusses an alternative approach for implementation of design and presents a logical architecture to execute a knowledge base in parallel.

2.2 ACHIEVING REAL-TIME EXPERT SYSTEMS

One way that can achieve real-time response is to provably reduce the complexity. That is, reduce the combinatoric cost of the algorithm (hence, the execution time) [3]. Therefore, to provide real-time response for an expert system it is required to reduce the execution time of the knowledge base to a predetermined maximum. The relations between the rules of a knowledge base constitute an algorithm. Therefore, in real-time expert systems it is required to provably reduce the complexity of this algorithm (i.e. the knowledge base's execution time). The emphasis here is on provable complexity reduction rather than the standard approach currently applied to make it faster. This approach usually involves debugging and designing hand-crafted, implementation dependent program modifications [3]. These modifications often involve programming tricks which can incidentally reduce the algorithm's complexity. However, it usually reduces only the execution time without changing the complexity.

The latter method should be avoided for the following reasons [3]:

- (a) The performance may vary if the problem specifications and/or the type and quantity of data changes.
- (b) This method is extremely time consuming, and
- (c) It lacks a formal basis and therefore, is accidentally real-time, i.e. cannot determine the performance criteria before running the system.
- (d) Since there is no defined methodology, it provides little or no insight into real-time system construction that could be used later on.

The probable approaches that could be used to achieve real-time expert systems are [3]:

Parallelism : This is achieved by executing processes (inferences) in parallel instead of sequentially.

Metaplanning : It uses the existing knowledge to approach and organize the problem in order to reach real-time response.

These are explored further in the following sections.

2.2.1 PARALLELISM

Parallelism is a method of program design that can or does satisfy the strict definition of "real-time" [3]. It is often possible to improve the performance by mapping the algorithm onto a multiprocessor architecture.

It is worth while mentioning that not all algorithms can be parallel-implemented. To meet the conditions an algorithm must be decomposable and independent [3]. An algorithm 'a' is decomposable if a mapping can be defined such that:

$$D : a \rightarrow (a_1, a_2, \dots, a_n)$$

from "a" into a set of more primitive components

$$(a_1, a_2, \dots, a_n)$$

such that for any set of data "d"

$$a(d) = C_{i=1}^n a_i(d)$$

for some composition "C" of the a_i .

An algorithm is "independently decomposable" if it is decomposable and the component operations are independent [3]. Thus, we must be able

to show that "C" is a set of union operators U, and not an operator that imposes temporal sequential constraints between the a_i 's.

Even if the algorithm is not independently-decomposable, it may be possible to change it into a parallel form. For example to calculate

$$a = 1 + 2 + 4 + 6$$

The conventional implementation would be to compute this sequentially. However, this can be decomposed into two parts that could be computed in parallel. Thus, we can perform

$$1 + 2$$

and

$$4 + 6$$

simultaneously, and then compute the sum of these calculations in a second step. This decomposes the calculation into two steps that are independent.

This example suggests that parallelism can be the means of achieving real-time performance, since the time complexity (execution time) is reduced when it is applicable [3].

Parallelism is the methodology applied by this thesis to achieve real-time expert systems. This is due to the fact that a procedure already exists for extracting parallelism from an algorithm. This will be extended to extract maximum parallelism in a knowledge base.

2.2.2 METAPLANNING

There are still some problems of great practical interest that no degree of parallelism is likely to provide a reduction of computation complexity to finite time [3]. A solution to these problems is often available through the use of metaplanning.

Metaknowledge is the main factor in metaplanning. Metaknowledge is an awareness of how to think, to approach and organize an unfamiliar problem such that all other forms of knowledge are effectively applied [4].

Metaplanning is defined as all meta level planning, control, and plan execution monitoring operations [3]. Meta level operations are characterized by their preoccupation with the state of system and not the state of the problem data. This means meta level planning involves evaluating how well a problem is solved, and by using this information, metaplanner plans and implements better approaches in problem solving.

The design requirements to provide real-time systems through

metaplanning are [3] :

- the metaplanner must be real-time itself. That is, the metaplanner must be able to react to external events (such as changes in the computational state of the system) in finite time. This is the most important requirement of a system, since if metaplanner is not real-time how could the system be real-time.
- metaplanner must maintain a model of the computational capabilities and cost of each non-real-time algorithm used, in order to predict its performance on a given problem set and check its performance as the problem solving progresses.
- the metaplanner must have access to an accurate source of time information on the scale proportional to the events it is attempting to monitor.
- the metaplanner must have process control privileges so that for example, when needed it can specify new tasks or shut the system down entirely.

Thus the metaplanner does the following tasks :

It uses its capabilities or techniques to establish an initial plan for computation, including the required milestones specification for the computation. It then starts one or more problem-solving activities and assigns

each a fixed time limit and then monitors the achievement of milestones and other aspects of the system state. When and if necessary, it takes corrective actions to bring the system's operations back into line with the expected outcome. It also replans whatever plan fragments for the given milestone that was not achieved.

The above design methodology provide us with another way of achieving real-time systems. Metaplanner also requires an adequate model of the behaviour of the system for proper design. Thus, for the metaplanning design methodology to be applied, the system has to be implemented and observed. This does not provide any information about the system's performance before implementation. If parallelism does not achieve the required response, metaplanning could be applied after parallelism in an attempt to achieve the real-time response.

The following section presents the required definitions and the algorithm for extracting maximum parallelism.

2.3 ANALYSIS OF PARALLEL ALGORITHM

This section provides the required definitions and expressions for modeling of an algorithm in terms of its data-flow and control. These have been taken from references [1] and [2]. These definitions are then extended to knowledge bases.

An algorithm is completely described by two parts, control and data, which are represented by precedence and data-flow graphs respectively. A data-flow graph consists of a set of nodes (containing operators P, and storage cells S) and a set of edges required by the operators to indicate the input and output storage cells. An operation can be represented as

$$F_p([A], [C]) \rightarrow B$$

This means that the contents of cells A and C are operated upon by operator 'p' and the result is stored in cell B.

A precedence graph consists of a set of nodes N_c , containing operator nodes, entry and exit nodes, and a set of directed edges between nodes indicating flow of control. An AND operator $\cap$ is used to synchronize parallel flow of control, where the flow can proceed to the next operator only when all operations prior to the AND have been completed. An OR operator $\cup$ is used to join the flow of non-parallel control edges, in which the control is passed to the next operator when any one of the inputs is activated. Therefore, a precedence graph gives information about the order of execution of the operators. A control expression, CE, is formed by combining a set of distinct operator names with a set of relational operators

$$(., \cap, \cup, (,))$$

as described below:

$\cup$ and $\cap$ operators are the same as above described operators.

The parentheses $()$ define the execution precedence for the control subexpressions. A parenthesised subexpression has a higher precedence over terms to its right.

The cascade operator $.$ is associative but not commutative and the operator on the left has higher precedence than those on the right.

A data expression DE_i is defined to be a set that specifies the input and output data cells required for operator i .

$$DE_a = (I_a, O_a)$$

where :

I_a is a set of input data cells

O_a is a set of output data cells

Example 2.1 : For the data flow and control flow graphs in Fig. 2.1 and

2.2 respectively, the data expressions are :

$$DE_a = [(A, B), E]$$

$$DE_b = [(B, C), F]$$

$$DE_c = [(D, E), G]$$

$$DE_d = [(E, F), H]$$

$$DE_e = [(G, H), K]$$

To realize alternate parallel/serial configurations that achieve the same results, it is desirable to establish a notation of computational equivalence.

DEFINITION 1: Two computations are data equivalent if

- (a) Their data flow have equivalent operator functions and data cells.
- (b) For any initial state, the same final state and the same history (data expressions) for all of the data cell is achieved by both computations.

In this definition it is required that the data portion of the computation be fixed.

Figure 2.1 : An example data-flow graph

	a	b	c	d	e
a	X	$\wedge$	$>$	$>$	$>$
b	$\wedge$	X	$\wedge$	$>$	$>$
c	$<$	$\wedge$	X	$\wedge$	$>$
d	$<$	$<$	$\wedge$	X	$>$
e	$<$	$<$	$<$	$<$	X

Figure 2.2 The precedence graph and table for the data-flow graph.

The following relations are defined for a pair of operators, (a,b), in a computation:

$a \wedge b$: a, b are independent, i.e.

$$O_a \cap (I_b \cup O_b) = 0$$

and

$$O_b \cap (I_a \cup O_a) = 0$$

$a > b$: "a" must precede "b" for correct operation, i.e.

$$(O_a \cap I_b) \neq 0$$

$a < b$: "b" must precede "a" for correct operation, i.e.

$$(O_b \cap I_a) \neq 0$$

$a \parallel b$: "a" and "b" are in parallel, i.e. neither $a < b$ or $a > b$

$a - b$: "a" and "b" are on alternative control paths, this occurs when

$$(O_a \cap I_b) \cup (O_b \cap I_a) = 0$$

the two paths are diverging when

$$(I_a \cap I_b) \neq 0$$

and are converging when

$$(O_a \cap O_b) \neq 0$$

If two operators are on alternate paths, they could be executed in parallel. This is due to the fact that the result of one operation is not required for the other operation.

Before presenting the algorithm there are a few points that should be considered:

THEOREM 1 : Consider the two parallel sets

$$A = [A_1, A_2, \dots, A_n]$$

and

$$B = [B_1, B_2, \dots, B_m]$$

Thus

$$A_1 \parallel A_2 \parallel \dots \parallel A_n$$

and

$$B_1 \parallel B_2 \parallel \dots \parallel B_m$$

then A must precede B ($A > B$) for correct operation if and only if

$$A_i > B_j \text{ for all } i \text{ and } j.$$

Proof : Obvious.

THEOREM 2 : The operations $A_1, A_2, \dots, A_n$ can only be performed in parallel if and only if

$$A_1 \parallel A_2, A_1 \parallel A_3, \dots, A_1 \parallel A_n$$

$$A_2 \parallel A_3, A_2 \parallel A_4, \dots, A_2 \parallel A_n$$

$$A_{n-1} \parallel A_n.$$

Proof : Obvious.

The following theorem is presented to prove the correctness on the extension made in this thesis to the algorithm presented in [2] for deriving the precedence table.

THEOREM 3 : If operators 'a' and 'b' are on alternate paths, they can be executed in parallel.

Proof : Operators 'a' and 'b' are on alternate path if

$$(O_a \cap I_b) \cup (O_b \cap I_a) = 0$$

then:

$$(O_a \cap I_b) = 0$$

and

$$(O_b \cap I_a) = 0$$

Therefore, neither $a < b$ or $a > b$. Thus, 'a' and 'b' are in parallel.

It also should be noted that a serial to parallel transformation, i.e. $(a.b \rightarrow a \parallel b)$ is possible if and only if

$$O_a \cap (I_b \cup O_b) \neq 0$$

and

$$O_i \cap (I_i \cup O_i) = 0$$

For any computation we can present the operator relations $>$ or $<$, and $\parallel$ with a precedence table P (a square $n \times n$ matrix with unique entry for each operator pair (a_i, a_j) , where a_i and a_j are the row and column designation, and $i \neq j$).

The following section presents an algorithm that uses the above concepts and definitions to provide maximum parallelism in a given knowledge base.

2.3.1 EXTENSION TO KNOWLEDGE BASES

This section presents an algorithm that extracts maximum parallelism in a rule-based knowledge base. It is an extension of the general model used to reconstruct a serial structure into a parallel form derived in [1], and the data independent version in [2].

The hypothesis is that knowledge bases are data independent. This is based on the assumption that relations among the rules in a given knowledge base are invariant. Therefore, the data independent model derived in [2] could be used to decompose the knowledge base into independent segments that could be executed in parallel. This algorithm does not consider

alternate paths, therefore, an extension was made to this algorithm to find the precedences of the operators when alternate paths exist.

In order to map the algorithm presented in [2] on to knowledge bases it was required to define operators and operations based on the definitions used in [2]. It should be noted that these are different from the usual or conventional definitions. If the execution of rules (IF ... THEN ...) are the operations, then any thing that results in an operation is an operator in terms of definitions in [2]. This implies that the consequences of the rules and the input to the system constitute the operators.

In order to draw the data-flow graph and the precedence table, it is required to evaluate the sequence of firing of the operations. Therefore, it is necessary to evaluate which operators result in what operations and visa versa. Hence, a methodology is required to give antecedents and consequents of the rules the proper operator names. This methodology is presented in steps 1 to 9 of the following algorithm. The inputs to the system could be excluded from the precedence table and data-flow graph, since the knowledge base is data independent. However, some inputs may exist which are invoked by a consequent of a rule and require some time for processing before they are available. These inputs affect the sequence of firing and therefore, are included in the data-flow graph, precedence graph, and precedence table.

Given a knowledge base, a set of operators are derived from the rules in the knowledge base. The data-flow graph and the precedence table are drawn for these operators. The operations that could be executed at any precedence level are found by searching for operators that are independent of each other, and only require the operators in the preceding precedences to be available.

The following are the detailed steps that should be taken to achieve the maximum parallelism.

1. For all the antecedents of the rules
 - (a) Remove all the initial input to the system (these are the inputs that are not invoked by a consequence of a rule).
 - (b) Derive a set from all the consequences and remaining antecedents of the rules given in the knowledge base.
2. If antecedents do not contain certainty factors, then assign an operator name for each member of the set found in 1b. Go to step 9.
3. For all members of the set:
 - (a) If it is an input to the system, give it an operator name, Go to step 8.

- (b) If a consequent has certainty factor, find the consequent with the lowest certainty factor and give it an operator name. Go to step 4.
- (c) If no certainty factors are involved, assign an operator name to the consequent being considered, and go to step 8.
4. Search through antecedents and find all the antecedents named the same as this consequent without considering the certainty factors.
 5. If the antecedent has an equal or lower certainty factor than the consequent being considered and is not already assigned an operator name, give it the same operator name as consequent in step 3b. If the antecedent doesn't have a certainty factor involved, give it the same operator name as the consequent in step 3b.
 6. Repeat step 5 for all antecedents found in 4.
 7. Find the consequent with the next lowest certainty factor, give it an operator name and follow step 5.
 8. Go to step 3 until all the members have an operator name.
 9. Draw the precedence graph and the precedence table for the assigned operators (a_i 's).
 10. Build a precedence table for operator pairs (a_i, a_j) according to the

following procedure :

REPEAT FOR all i and j ($i \neq j$)

IF $P(a_i, a_j) = 0$ (not yet specified)

THEN IF $(O_{ai} \cap I_{aj}) \neq 0$

THEN $P(a_i, a_j) = >$

END IF

IF $(I_{ai} \cap O_{aj}) \neq 0$

THEN $P(a_i, a_j) = <$

END IF

IF $(O_{ai} \cap I_{aj}) \cup (O_{aj} \cap I_{ai}) = 0$

THEN $p(a_i, a_j) = -$

END IF

IF $(O_{ai} \cap (I_{aj} \cup O_{aj}) = 0 \text{ AND } O_{aj} \cap (I_{ai} \cup O_{ai}) = 0)$

THEN $P(a_i, a_j) = \wedge$

END IF

END IF

IF $P(a_i, a_j) = < \text{ (or) } >$

AND

$P(a_j, a_k) = < \text{ (or) } >$

THEN $P(a_i, a_k) = < \text{ (or) } >$

END IF

IF $P(a_i, a_j) = \wedge \text{ OR } P(a_i, a_j) = -$

THEN $P(a_i, a_j) = \parallel$
END IF

Following the above algorithm will provide a precedence table. To obtain a set of parallel operators, the following steps should be taken:

1. Choose the row (or rows) with entries $>$ and/or $\parallel$ only. The corresponding operators consist of a parallel set. If there is no such row, then the remaining operators make parallel sets (each operator is the only member of the set).
2. Delete all the rows and columns of the precedence table P corresponding to the operators contained in the parallel set found above. Repeat step 1 until there are no operators left in the precedence table.
3. The precedence of the parallel sets is the order in which they were extracted from the table. This could be verified using Theorem 2.

The above algorithm can decompose large static knowledge bases (knowledge bases with stable tree representations) into as many as possible parallel executable segments. Therefore, the required time to execute the knowledge base could be reduced to its minimum.

Example 2.2 : In order to illustrate the application of the above algorithm, a simple example is now presented. Assuming that the knowledge

base contains the following rules:

IF A THEN C

IF (A AND H AND R) THEN B

IF B THEN E AND G

IF (B AND J) THEN D

IF J THEN P OR W

IF (D AND P AND T AND S) THEN M

IF D THEN N

IF (G AND C AND K AND N) THEN F

IF F AND W THEN L

and all the input are available before execution, then the operator set is

$$S = (B, C, D, E, F, G, L, M, N, P, W)$$

and assigning the same names for the operators, the data flow graph and precedence table are as shown in Figures 2.3 and 2.4 respectively.

The first parallel set found after step 1 is (B, C, P, W)

After applying the required steps, the parallel sets are found to be:

(B, C, P, W)

(D, E, G)

(M , N)

(F)

(L)

Therefore, the control expression CE is :

$$CE = (B \parallel C \parallel P \parallel W) (D \parallel E \parallel G) (M \parallel N) (F) (L)$$

The rules which could be executed in parallel are the rules that contain these operators as consequences. Thus, the parallel sets are :

set 1:

IF A THEN C

IF (A AND H AND R) THEN B

IF J THEN P OR W

set 2:

IF B THEN E AND G

IF (B AND J) THEN D

Figure 2.3 : Precedence graph for knowledge base in the example 2.2
Shaded nodes indicate the goal nodes.

	B	C	D	E	F	G	L	M	N	P	W
B	X		>	>	>	>	>	>	>		
C		X			>		>				
D	<		X		>		>	>	>		
E	<			X							
F	<	<	<		X	<	>		<		
G	<				>	X	>				
L	<	<	<		<	<	X		<		<
M	<		<					X		<	
N	<		<		>		>		X		
P								>		X	
W							>				X

Figure 2.4 : Precedence table for the knowledge base given in example 2.2.

set 3:

IF (D AND P AND T AND S) THEN M
IF D THEN N

set 4:

IF (G AND C AND K AND N) THEN F

set 5:

IF F AND W THEN L

and the maximum parallelism is achieved when three inference engines are used.

EXAMPLE 2.3 : This example is presented to illustrate the use of the presented algorithm when certainty factors are involved. The knowledge base in this example is very similar to that of example 2.2.

IF A THEN C

IF (A AND H AND R) THEN B(0.4)

IF B (c.f. > 0.1) THEN E AND B(0.6)

IF J THEN P(0.4)

IF B (c.f. > 0.5) AND J THEN D(0.2)

IF (D AND P AND T AND S) THEN M AND D(0.5)

IF $D_0(c.f. > 0.4)$ THEN N
 IF (AND C AND K AND N) THEN F
 IF F THEN L

assuming that all the required input are available before execution, then
 the operator set is

$$S = (B(0.4), B(0.6), C, D(0.2), D(0.5), E, F, L, M, N, P(0.4))$$

and assigning the names for the operators, then the operators are

$$S = (B1, B2, C, D1, D2, E, F, L, M, N, P)$$

After drawing the data flow graph and precedence table The first parallel
 set found after step 1 is $(B1, C, P)$

After applying the required steps, the parallel sets are found to be:

(B1, C, P)

(B2 , E)

(D1)

(M, D2)

(N)

(F)

(L)

Therefore, the control expression CE is :

$$CE = (B1 \parallel C \parallel P) (B2 \parallel E) (D1)(D2 \parallel M) (N) (F) (L)$$

The rules which could be executed in parallel are the rules that contain these operators as consequences. Thus, the parallel sets are :

set 1:

IF A THEN C

IF (A AND H AND R) THEN B(0.4)

IF J THEN P(0.4)

set 2:

IF B (c.f. > 0.1) THEN E AND B(0.6)

set 3:

IF B (c.f. > 0.5) AND J THEN D(0.2)

set 4:

IF (D AND P AND T AND S) THEN M AND D(0.5)

set 5:

IF D (c.f. > 0.4) THEN N

set 6:

IF (G AND C AND K AND N) THEN F

set 7:

IF F THEN L

and the maximum parallelism is achieved when three inference engines are used.

The usefulness of this algorithm becomes more apparent as the knowledge base expands.

The next section discusses performance properties of the system when parallelism is implemented.

2.3.2 PERFORMANCE PROPERTIES

After determining the maximum parallel form, it would be useful to be able to guarantee a maximum response time, or at least measure the difference between the execution time of the equivalent parallel and serial algorithms. To accomplish this, we use a method for determining the expected execution time of an algorithm [1]. This is then extended to calculation of the execution time of a knowledge base when the sequence of firing is known.

The following definitions are required for this calculation. Let each operator, a_i , be represented by a vector a_i [1]

$$a_i = \begin{pmatrix} t(a_i) \\ p(a_i) \end{pmatrix}$$

where :

$t(a_i)$ = expected execution time contribution of a_i

= execution time of one inference

$p(a_i)$ = conditional probability of a_i executing given that its immediate predecessor(s) (operators preceding a_i) have completed.

Each decision (execution of the next node) is modeled as a discrete probability distribution where any node " i " is assigned probability p_i and

$$\sum_i p_i = 1$$

In knowledge based systems the probability of a node (exit) being executed is dependent on the inference engine implemented. It is assumed that the inference engine does an depth-first search to find the proper response, and also follows the first branch from the node being considered. For example, if a rule results in 'n' consequences, then the probability of the first branch being executed is one, and the probability of remaining

branches being executed is zero. Therefore, for any node

$$P(a_1) = 1$$

and

$$P(a_i)_{i=2 \text{ to } n} = 0$$

Determination of operator probabilities :

1. If the immediate predecessor of "a" is a non-decision operator. That is, the rule being considered has only one consequent. Then :

$$p(a) = 1$$

2. If the immediate predecessor of "a" is a decision operator and "a" follows decision outcome i. That is, the antecedents of the rule "i" result in several outcomes (consequents). Then:

$$p(a) = P_i$$

Calculating the expected execution time :

The expected execution time of a computation is calculated by processing a control expression (CE) sequentially in reverse polish notation form [1]. To avoid special cases in the algorithm a "null operator" (zero execution time) is inserted immediately after each decision exit (this leads to a single $\cap$ or $\cup$ operators).

Now let

$$j = [0 \ 1] \quad \text{and} \quad k = [1 \ 0]$$

then, the expected execution time contributions for each subexpression can be calculated as follows:

1. Serial Control Expression:

Let $a_1, a_2, \dots, a_n$ be a serial control then

$$E(T) = \left[\prod_{i=1}^n j a_i \right] \left[\sum_{i=1}^n \frac{k a_i}{j a_i} \right]$$

2. Parallel Control Expression:

Let $a_1 \parallel a_2 \parallel \dots \parallel a_n$ be a parallel control expression if each operator contributes a fixed amount of execution time then

$$E(T) = \left[\prod_{i=1}^n j a_i \right] \left[\max_{i=1}^n \left(\frac{k a_i}{j a_i} \right) \right]$$

3. OR Control Expression

Let $a_1 \cup a_2 \dots \cup a_n$ be an OR control expression then

$$E(T) = \sum_{i=1}^n k a_i$$

To calculate the total expected execution time, apply the above equations. This results in an expression model that calculates the expected execution time for a given computation. This could be extended to evaluate the execution time for a given knowledge base. In a knowledge base if the execution sequence is known, then the probability of the given operators being executed is one. That is

$$p(a) = 1$$

and the probability of the remaining operators being executed is zero. Therefore, the expected execution time becomes the actual execution time. The following is an example to illustrate this concept.

EXAMPLE : Considering the knowledge base presented in the example 2.2 of the previous section, then the operators vectors are as follows:

$$i = \begin{pmatrix} t_i \\ p_i \end{pmatrix}$$

Where :

i = The operator name

t_i = The time it takes to execute i

p_i = Probability of i being executed

The total probability of leaving a decision is one, since the rules are fired one after another.

For example B's vector is

$$B = \begin{pmatrix} t_b \\ 1 \end{pmatrix}$$

For sequential execution let us assume that all the rules are fired before the goal node is reached. Therefore, all the operators have to be executed. In this case, all the operators are in series.

Thus

$$E(T) = \left[\prod_{i=1}^n j a_i \right] \left[\sum \frac{k a_i}{j a_i} \right]$$

and assume that the first consequent of the rule is the one to be considered, then for example, after operator B, the next operator to be executed

is E. Thus:

$$p(E) = 1$$

and

$$p(G) = p(D) = 0$$

The execution time of any rule varies with the number of antecedents and consequences. Backtracking is one of the factors affecting this time. For example, if the correct path to be followed is through G and not E, then it must backtrack to follow the branch through node E. In this case, the execution time of node E is insignificant (E is evaluated when G was evaluated). Thus, for the given knowledge base it is assumed that

$$t_e = t_g = t_{eg}$$

and

$$t_p = t_w = t_{pw}$$

After the required calculations the estimated execution time $E'(T)$ was found to be

$$E'(T) = t_b + t_c + t_d + t_{eg} + t_f + t_l + t_m + t_n + t_{pw}$$

The backtracking time has been left out, since this time is negligible compared to the execution time, therefore the execution time is slightly larger than $E'(T)$. If all inferences take the same amount of time to execute, then

$$E'(T) = nt$$

where:

n = Number of operators involved.

therefore,

$$E'(T) = 9t$$

It should be mentioned that $E'(T)$ is the estimated maximum execution time for a single inference engine (sequential machine).

If the maximum parallelism is implemented, then the maximum execution time is when the goal node is at the depth of the tree, i.e. when depth is:

$$d = 5 \text{ (this is achieved for goal node L).}$$

The longest execution time is calculated when executing the longest path, that is going through the path

$$GE = B.D.N.F.L$$

Then the estimated execution time is

$$E'(T) = t_b + t_d + t_n + t_f + t_l$$

If all inferences have the same execution time, then

$$E'(T) = 5t$$

Therefore, using parallelism, the execution time of the system is substantially reduced. In the above described situation the execution time when maximum parallelism is applied is almost half the execution time for the sequential machine. Since for both cases considered the execution path is known in advance, the estimated execution time is approximately equal to the actual execution time, which is the maximum execution time possible.

The following section provides an overview of the possible results and the systems that could use this method.

2.4 ASSESSMENTS

It is an appropriate time to discuss the possible systems that could achieve real-time expert systems having defined the different methods applicable and the milestones. The systems addressed here are those containing rule-based knowledge bases.

For a given rule-based knowledge-base and all input queries, it is possible to find the maximum parallelism possible and through it we would have the essential information to work with. For example, the feasibility of the system could be found by calculating the expected execution time when maximum parallelism is implemented. If this does not satisfy the real-time requirements, then the system is not real-time. Also given a set of requirements, for example, cost, execution time, and utilization of the processors, the ultimate solution may be obtained.

It is also possible to calculate the expected execution time, cost, etc. for the desired number of parallel processors.

The presented algorithm only deals with static knowledge bases. That is a knowledge base which only contains rules with predetermined relations. For dynamic knowledge bases (relations between the rules may change in the execution process) this algorithm is not valid, since this algorithm was designed based on the assumption that the relation between the operators are known prior to execution. This assumption is not valid for knowledge

bases with dynamic rules.

The following chapter presents a program implemented to extract maximum parallelism, and a logical architecture that could achieve this concurrency.

2.5 SUMMARY

The main issue in this chapter was to find a method to predict the execution time of the system and guarantee an adequate response before the desired execution time has elapsed. Two design methodologies were proposed to achieve this, parallelism and metapanning. Metapanning was eliminated because it is difficult to find real-time metaplanners for practical applications. Parallelism was used to provide real-time requirements, since the maximum parallelism possible in a knowledge base can be calculated. Therefore, the execution time could be estimated before the actual system is implemented. Hence, the feasibility of the system could be evaluated in advance. It also has the algorithmic criteria that metapanning seems to lack.

Parallelism is a method for organizing the knowledge base of a system into the most efficient form, and to execute it as parallel sub-knowledge bases. This is done by finding the sets of independent rules. Members of

these sets could then fire concurrently. The maximum parallelism is found by the number of rules in the largest set. The execution time is guaranteed by calculating the execution time for the worst case. That is, when the goal node is located at the depth of the knowledge base.

Applying the parallelism algorithm, allows a designer to evaluate the feasibility of the system before actual implementation. This could save time and decrease the cost of system design. The most important factor in designing real-time systems is the maximum execution time of the system which will determine the feasibility of the system. If the response time is less than or equal to the desired time, then the system is real-time. The approach permits the following calculations:

- the execution time of any intermediate nodes.
- the execution time of any existing goal nodes.
- the maximum execution time when maximum parallelism is implemented.
- the maximum execution time for the sequential machine.
- Guaranteed response time.

A logical architecture for implementing parallel inferences is presented in the next chapter.

Chapter 3

PARALLELISM IN A RULE-BASED EXPERT SYSTEM

3.1 INTRODUCTION

This chapter deals with a novel algorithm applied to achieve real-time expert systems. It also specifies the types of expert systems or knowledge bases applicable. A definition of the knowledge base is provided along with a logical architecture for implementing parallelism. Finally, an example is presented to illustrate the application of this algorithm.

The algorithm presented in chapter 2 provides the formal bases for finding concurrency in rule-based knowledge bases. This algorithm finds all the precedences and the links between the rules of a knowledge base. That is, it finds the links of the algorithm's topology. A heuristic approach based on the structure of knowledge bases and the above results is developed in this

chapter. This approach is easier to program since it is especially designed for knowledge bases. It provides additional results such as finding the goal nodes, and checking the knowledge base for inconsistencies, completeness, or redundant rules. The results from chapter 2 are used in this chapter for comparison.

In section 3.2, some background in the representation of knowledge is presented and the possible structure of the knowledge bases considered is described.

In section 3.3, a logical architecture which could be used to implement a decomposable knowledge base is proposed. This involves several inference engines in a unique configuration.

In section 3.4, a novel program that extracts maximum parallelism in a knowledge base is presented. This is based on the algorithm presented in section 3.3 for partitioning the knowledge base. This program provides additional information about the system. Results found here are compared with results found in section 2.3 for test cases.

Section 3.5 provides an example to illustrate this concept.

Chapter 4 presents the performance properties of the proposed design.

3.2 STRUCTURE OF A KNOWLEDGE BASE

The knowledge-base in any system represents all the information obtained from the expert about the problem domain. It contains facts, heuristics, and the formal knowledge. In a physical plant the knowledge base represents all the information obtained from the plant about the problem domain. It is assumed that it contains the essential information required to arrive at a solution. However, the representation of the knowledge in a form suitable for coding on a computer is of great importance. There are four different ways of organizing a knowledge base [4].

1. Rule-based
2. Formal knowledge
3. Frames or schemata
4. Semantic network

For our purposes it is assumed that a knowledge-base is in rule-based form, since rule-based systems are by far the most common and suited for presenting plants interaction [5]. Rules are normally of the 'situation-action' type (IF ... THEN). This is what is called 'production rules'. By combining them one to the other, we obtain a line of reasoning that provides a familiar structure in the form of IF ... THEN rules [5].

Rules of this kind have a considerable attraction because the system becomes strongly modular. If we wish to add a rule, the program does not have to be completely rewritten; the latest rule can be added to the earlier rules.

It is assumed here that the knowledge base used by the algorithm is a static knowledge base, i.e. consequents of the rules in the knowledge base do not change the chain of reasoning (i.e. the knowledge base has a stable tree or graph representation).

The following section describes the architecture that could execute these knowledge bases to reach a goal node.

3.3 ARCHITECTURAL CONSIDERATIONS

In this section, the suggested logical architecture for implementing parallel inferencing is derived from the general model in [4]. The purposed architecture is shown in Figure 3.1. In this architecture, a data-flow representation is assumed. The knowledge base is stored at the nodes (storage cells). The 'antecedent sort and match' and 'consequent dispatcher' form the inference engine of this design. An inference engine (also referred to as inference procedure) is the body of procedures used for traversing the knowledge base, in a systematic way, to arrive at conclusions based on an input query. In this design, the processing procedure is anything required for conversion of the input data to results that require decisions. These results are then

passed to every inference engine.

Each inference engine receives the results and finds the appropriate entry in the knowledge base. Therefore, it finds the rules that have these results as their antecedents. When all the antecedents for a rule have arrived, the consequence is dispatched to the appropriate processing unit for conversion to the next result requiring a decision, continuing until the algorithm is completed.

In order to achieve parallel inferencing, it is required to:

- Allocate m inference engines to the system, where m is the number of inferences that should be implemented for maximum parallelism, and
- partition the knowledge base into ' m ' sub-knowledge bases according to the following procedure and allocate each sub-knowledge base to an associated inference engine.

To obtain these partitions divide the rules in such a way that independent rules are allocated to different sub-knowledge bases. Build these sub-knowledge bases according to the following procedure:

1. Start with the original knowledge base

- (a) Collect all the rules with only initial inputs as antecedents (an initial input is the data given by the plants environmental inter-

actions). Allocate them arbitrarily and uniformly to one of 'm' partitions. Go to step 2b.

2. (a) Collect all the rules that are left with no antecedents, or only inputs as antecedents, and allocate them arbitrarily and uniformly to one of the m partitions.

(b) Remove these rules from the original knowledge base.

3. Take the consequences of the above rules and remove them from the antecedents of the remaining rules. If there are certainty factors involved in antecedents and/or consequents, only remove the antecedents with the same name and equal, lower, or no certainty factors. Repeat from step 2 until all the rules are removed from the original knowledge base.

It could be seen that these partitions are not unique and variations of these sub-knowledge bases could be found. Therefore, all the inference engines must be able to access every sub-knowledge base.

EXAMPLE : Considering the knowledge base for example 2.2 presented in chapter 2, one set of sub-knowledge bases are found to be:

1)

IF A THEN C

IF (C AND K AND N) THEN F

IF F THEN L

2)

IF (A AND H AND R) THEN B

IF B THEN E AND G

IF D THEN N

3)

IF J THEN P OR W

IF (B AND J) THEN D

IF (D AND P AND T AND S) THEN M

The above procedure also provides another way to find the precedences and the corresponding sets. This is illustrated as a program implemented in the following section.

3.4 ACTUAL PROGRAM IMPLEMENTATION

In a knowledge base the precedences of the rules are found by evaluating the links between the rules (searching for the links between antecedents and consequences of the rules). These links could be found by exhaustive search through the knowledge base. A rule may only fire if all the required antecedents are either inputs or consequents of the preceding rules. The first precedence level could be found by searching for rules that only require inputs to the system to fire. The remaining precedences could be found by searching the knowledge base for the rules that may fire as result of the rules in the higher precedences. This could be done by searching for a match between the consequences of the rules at higher precedences to the antecedents of any rule. If no match is found the the consequent is a goal node.

This section provides a program that extracts maximum parallelism in a rule-based knowledge base based on the above described approach. This is done by evaluating the rules that could be fired simultaneously at any precedence level. Goal nodes are found by searching for consequents that do not cause firing of a rule. The maximum parallelism possible at each level is also indicated by the number of rules that could be fired at that level. This program assumes that all the antecedents of a rule are related by an AND operator. Rules that have antecedents which are related by an

OR operator can be rewritten in the form of AND related antecedents. For example the rule

IF (A AND B) OR (C AND D) THEN F OR G

can be rewritten as set of rules

IF (A AND B) THEN F OR G

IF (C AND D) THEN F OR G

which has the same effect. This is similar to the example presented in section 2.2.1 . This program is designed for any static knowledge bases.

The pseudocode for this program is presented here. More detailed structure of the program can be found in the documentation of program in Appendix A, along with a few examples.

1. Pseudocode:

***This portion of program finds the goal nodes **

FOR all the rules in the knowledge base

FOR all the consequents in the rule

IF consequent is not antecedent of any rule

THEN consequent is a goal node

END IF

END FOR

END FOR

** This portion finds the links between the rules **

INITIALIZE precedence level to zero

REPEAT

FOR all the rules in the knowledge base

FOR all the antecedents in the rule

IF antecedent is not consequence of any rule OR
is consequent of a rule at higher precedence and
requires equal or lower certainty factor than that
consequent

THEN antecedent = BLANK (remove the antecedent)

END IF

END FOR

END FOR

** This portion finds the rules at the same precedence level **

INITIALIZE level of parallelism to zero

WRITE the precedence level

FOR all the rules in the knowledge base

IF all antecedents of the rule are BLANK (removed)

```

    THEN the rule can be fired at this precedence
    increment level of parallelism
    remove the rule by setting the index of consequences
    to zero
    WRITE the rule number
  END IF
END FOR
increment precedence level
WRITE the level of parallelism
UNTIL all the rules are eliminated

```

2. Inputs :

The programs input is basically the knowledge base. The antecedents (consequents) should be separated by a blank and a comma. The antecedents and the consequences are separated by a colon. The end of a rule is indicated by an asterisk. Any certainty factors for antecedents or consequents of the rules are indicated by a plus sign, and the certainty factor. A sample input is also presented in Appendix A.

3. Output :

The program prints out the number of rules in the knowledge base. For each precedence, it prints the rules that could be fired concurrently, and the maximum parallelism possible at that precedence level. If any goal nodes exists as a consequence of these rules, it is also printed out.

3.5 EXAMPLE

This example uses the knowledge base provided for the example 2.2 presented in section 2.3. After applying the algorithm presented in section 3.4 the following result was achieved.

There are 9 rules in this knowledge base

at precedence 0

the following rules can execute concurrently

rule 1

rule 2

rule 5

Maximum parallelism possible at this level is 3.

at precedence 1

the following rules can execute concurrently

rule 3

there is goal node E as a consequent of this rule.

rule 4

Maximum parallelism possible at this level is 2.

at precedence 2

the following rules can execute concurrently

rule 6

there is goal node M as a consequent of this rule.

rule 7

Maximum parallelism possible at this level is 2.

at precedence 3

the following rules can execute concurrently

rule 8

Maximum parallelism possible at this level is 1.

at precedence 4

the following rules can execute concurrently

rule 9

there is goal node L as a consequent of this rule.

Maximum parallelism possible at this level is 1.

It could be seen that the two approaches presented in section 2.3 and section 3.4 achieved the same results. The maximum parallelism is achieved when three inference engines are used.

3.6 SUMMARY

In this chapter a logical architecture was presented for implementing parallel inferencing. This was done by providing as many inference engines as required to implement maximum parallelism. An algorithm was provided which divides the knowledge base into partitions that could be executed concurrently by these inference engines. This was done by finding the precedences, and rules that could fire at that precedence. Rules at the same precedence are allocated to different sub-knowledge bases (partitions). This architecture introduces parallelism into the system and could reduce the execution time substantially.

A program was implemented based on the links between the rules of a given knowledge base which evaluates the precedences and rules that could fire simultaneously at that precedence. The goal nodes of the knowledge base were found by searching for rules with no links to the next precedence level. This could also be used for checking the knowledge base for any inconsistencies or completeness.

This chapter provides an approach to convert the knowledge base of a system to its most efficient concurrent form. This is done by executing the rules at the same precedence in parallel. Many problems requiring real-time response may be designed using expert systems software by applying this approach.

The following chapter deals with the expected execution time and the trade-offs associated with implementing parallel inferencing.

Chapter 4

PERFORMANCE CONSIDERATIONS

4.1 INTRODUCTION

In this chapter, we are concerned with the performance properties of the system. This involves calculating the maximum execution time for a given knowledge base, and the speed-up of the system after parallelism is introduced. Finally, the relation between the number of parallel inferences, utilization, and speed-up is presented. These will provide means for evaluating the performance of a given system. Standard architecture performance quantifications such as found in [8] are extended to calculation of execution time, and applied for evaluation of speed-up and utilization.

Section 4.2 provides the means for calculating the execution time, speed-up, and utilization for a given system. In order to tabulate the relations between the number of inferences, utilization, and speed-up actual numeric

values are required. This is shown in the example presented in the following section.

Section 4.3 provides an example to illustrate these concepts. The results are compared with those of section 2.3.2.

4.2 RESPONSE TIME

After determining the maximum parallelism, it is of interest to have an estimate of the execution time and to be able to guarantee a response within a period of time given the inputs are available before execution, or the waiting period is negligible with respect to the execution time.

For any system when maximum parallelism implemented, the execution time at any level is only dependent on the maximum execution time at that level. For example, at any level n if there can be 'm' inferences running at the same time and we have

$$t_{n1} > t_{n2} > t_{nm}$$

where:

t_{ni} = execution time for inference i

then

execution time at level $n = t_{n1}$

therefore, for any level i

$$T_{response} = MAX[t_{mi}]$$

and the response time of the system is

$$T_{resp} = \sum_{i=1}^d MAX[t_{mi}] \quad (4.1)$$

where:

d = depth of the goal node

t_{mi} = execution time of inference 'm' at level i

If due to some limitations, maximum parallelism can not be provided,
that is if

$$N < M$$

where:

N = no. of parallel inferences implemented

M = no. of parallel inferences achievable

then the response time would be

$$T_{response} \leq \sum_{i=1}^d \left[\frac{M_i}{N} \right] MAX[t_{mi}] \quad (4.2)$$

where:

M_i = maximum no. of parallel inferences achievable at level i

N = no. of parallel inferences implemented

The ceiling function is used here because an integer number of inference engines are required.

Therefore, for any level of parallelism, the maximum response time would be

$$T_{response} = \sum_{i=1}^d \left\lceil \frac{M_i}{N} \right\rceil MAX[t_{mi}] \quad (4.3)$$

The improvement in the processing time may be measured by the processors speed-up S_m . Speed-up is defined as the total execution time using the serial processor P_1 divided by the total execution time using 'm' parallel inferences P_m [8].

$$S_m = \frac{\sum_{i=1}^d M_i MAX[t_{mi}]}{\sum_{i=1}^d \left\lceil \frac{M_i}{m} \right\rceil MAX[t_{mi}]} \quad (4.4)$$

The efficiency with which the program utilizes the available facilities of P_m is measured by the processor utilization U_m [8]. This may be defined as the total space-time (number of processing elements multiplied by processing time) required in the sequential case, divided by the total space-time

available in P_m . Utilization can also be interpreted as the ratio of the actual speed-up to the maximum possible speed-up 'm' for any system [8].

$$U_m = \frac{S_m}{m} = \frac{\sum_{i=1}^d M_i \text{MAX}[t_{mi}]}{m \sum_{i=1}^d \lceil \frac{M_i}{m} \rceil \text{MAX}[t_{mi}]} \quad (4.5)$$

At this point it is worth while calculating the speed-up of the system when maximum parallelism " M " is implemented for the worst case. This occurs when goal node is at depth " d ", which is the depth of the knowledge base.

To calculate the corresponding speed-up, the time response for one inference engine (sequential) and for N parallel inference engine is required. These response times could be calculated by substituting for " N " in the following equation.

$$T_{\text{response}} \leq \sum_{i=1}^d \lceil \frac{M_i}{N} \rceil \text{MAX}[t_{mi}] \quad (4.6)$$

The maximum response time for maximum parallelism is achieved when

$$N = M$$

therefore,

$$T_{response} \leq \sum_{i=1}^{d'} \left[\frac{M_i}{M} \right] MAX[t_{mi}] \quad (4.7)$$

where :

$$M_i \leq M$$

therefore,

$$T_{min} = \sum_{i=1}^{d'} MAX[t_{mi}] \quad (4.8)$$

The maximum response time for a single inference engine is found when

$$N = 1$$

then

$$T_{max} \leq \sum_{i=1}^{d'} M_i MAX[t_{mi}] \quad (4.9)$$

Therefore, the speed-up is

$$speed - up = \frac{t_{max}}{t_{min}} = \frac{\sum_{i=1}^{d'} M_i MAX[t_{mi}]}{\sum_{i=1}^{d'} MAX[t_{mi}]} \quad (4.10)$$

If all inference engines take the same amount of time to execute one inference, that is

$$t_{m1} = t_{m2} = t_{m3} = \dots = t_{mN}$$

then

$$speed - up = \frac{\sum_{i=1}^{d'} M_i}{d'} \quad (4.11)$$

The utilization of the inference engines is

$$utilization = \frac{speedup}{M} \quad (4.12)$$

where:

M = number of inference engines implemented

Therefore, for any system, given the required cost, the corresponding utilization and speed-up could be found, and through these the number of parallel processors required can be calculated.

4.3 EXAMPLE

In order to illustrate the application of the above concepts the example presented in the previous chapters is considered.

From previous calculations it was found that after applying the parallelism algorithm, the control expression was found to be:

$$CE = (B \parallel C \parallel P \parallel W) (D \parallel E \parallel G) (M \parallel N) (F) (L)$$

and the maximum parallelism is achieved when three inference engines are used.

The maximum execution time for this system when maximum parallelism is implemented is:

$$T = \sum_{i=1}^d \text{MAX}[t_{m_i}]$$

where :

d = depth of the knowledge base or number of parallel sets achieved.

= 5

m = number of parallel inferences implemented.

= 3

Assuming that all inferences take the same amount of time to execute, then

$$t_1 = t_2 = t_3 = t$$

then

$$T_{resp} = \sum_{i=1}^5 t = 5t$$

2

OF/DE

2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

Thus, we can guarantee a response (reaching a solution node) within $5t$ after the input to the system is available.

Assuming that the backtracking time is negligible with respect to one inference execution time (as described in section 2.3.2), then the maximum execution time for the system using single inference engine is

$$T = \sum_{i=1}^d M_i \text{ MAX}[t_{mi}]$$

where :

$$d = 5$$

If all inferences take the same time to execute one inference, that is

$$t_{m1} = t_{m2} = t_{m3} = t$$

then

$$\begin{aligned} T &= \sum_{i=1}^5 M_i \text{ MAX}[t_{mi}] \\ &= 3t + 2t + 2t + t + t \\ &= 9t \end{aligned}$$

These results are in comparison with the results found in section 2.3.2. In both cases the guaranteed response time was found to be $5t$, when maximum parallelism was implemented, and $9t$ when a single inference engine was used.

Considering the above values for speed-up and utilization, i.e. applying maximum parallelism (three inference engines), then

$$\text{speed-up} = \frac{9t}{5t} = 1.8$$

and

$$\text{utilization} = \frac{\text{speed-up}}{m}$$

where :

$$m = 3$$

then

$$\text{utilization} = \frac{1.8}{3} = .60$$

The graph of speed-up and utilization versus number of inferences implemented is shown in figure 4.1. It can be seen that the speed-up seems to have a linear relationship with the number of inference engines, while utilization continues to decrease. However, there is usually a little increase in speed-up as the number of inference engines increases beyond the average value of parallel inference engines required to achieve maximum parallelism.

Figure 4.1 : Graph of Speed-up and Utilization versus number of inference engines implemented for example 4.1

4.4 SUMMARY

This chapter discussed the performance criteria of a given system. The maximum execution time of a system is calculated by considering the worst case. That is, when the solution node is at the depth of the knowledge base. For parallel inferencing the longest path must be searched to reach the goal node. In the sequential case, maximum execution time occurs when all other possible paths are exhausted before reaching the solution node.

The speed-up was found by calculating the ratio of the execution time of the system when parallelism is implemented to the execution time for single inference engine. Maximum speed-up is reached when maximum parallelism is implemented. Utilization is found by the ratio of this number to the parallel inferences implemented.

Implementing parallel inferences required to achieve minimum execution time possible, increases the cost to its limit. This also reduces the utilization of the inferences. Therefore, there is a trade-off between the speed-up of the system and its cost. The feasibility of the system could be evaluated given the required performance criteria and cost limitations before actual implementation.

Chapter 5

SUMMARY AND DISCUSSION

5.1 SUMMARY

The main focus of this work was to achieve a guaranteed response time for a given rule-based knowledge base. The knowledge base may include rules with attributes that contain certainty factors, as long as it remains static. That is, the relations between the rules do not change during execution. To achieve the guaranteed response time, a means for calculating the execution time and preferably reducing the time cost to its minimum is required. Parallelism was the system design applied to achieve this requirement. This was done by finding the rules in the knowledge base that could execute independently, and allocating them to precedence sets. Maximum parallelism at any precedence is achieved when every member of that precedence set is executed simultaneously. Therefore, all possible routes are executed

concurrently. This eliminates the backtracking problem. The precedence sets then partition the knowledge base in the most optimum way. These partitions then could be executed concurrently.

A special architecture is required to execute these partitions such that the performance of the system is maximized. The proposed architecture suggests implementation of multi-inference engines such that all the partitions could execute concurrently. This occurs when maximum parallelism is achieved. That is, there are as many inference engines implemented as the number of rules in the largest precedence set. The execution time of the system is then minimized and the maximum performance is achieved. Therefore, the guaranteed response time is found by measuring the maximum execution time possible for a given implementation.

The major contributions of this work are as follows :

Guaranteed Response Time : Maximum execution time for a given implementation can be calculated when the solution is at the depth of the knowledge base. This provides the guaranteed response time.

Execution Time of Intermediate results : This execution time can be calculated when maximum parallelism is implemented. This time is equal to it's precedence, times the average execution time of the executed sets. This provides a means for calculating execution time of any node, including the goal nodes.

Optimal Strategy : The development of an algorithm which finds optimum strategy for a given rule-based knowledge base. The algorithm partitions the knowledge base into its most efficient execution form.

Knowledge Base : The knowledge base of the system could be checked for inconsistencies and completeness by evaluating the goal nodes and their precedence levels.

Rule Redundancy : The implemented program provides a means for finding redundant rules. That is, rules which are never executed. These are the rules without precedences.

Architecture : The knowledge base of the system can be executed by implementing a parallel pipeline architecture.

Performance Criteria : A quantifiable measure for the utilization of the inference engines and speed-up of the systems are presented for the architecture proposed.

System Feasibility : Considering the time and performance cost, the system's feasibility could be evaluated before it is implemented.

5.2 DISCUSSION

A design applied for extracting maximum parallelism in an algorithm was extended to knowledge bases. If an architecture extendible to inference

engines could be found, then the design could be implemented. A data-flow machine is a suitable architecture, since the computation is data driven and this could simulate the rule execution process in a knowledge base. The data-flow concept is based on the fact that an event occurs only when all of the required input tokens are available. The results from an event are then used as input to the next event. This is very similar to the execution of rules in a knowledge base. That is, a rule could fire only if all the required antecedents are true, and available. The consequences are then used as antecedents of another rule. The data-flow machine suggested by Smith [9] is a feasible architecture for implementing this design.

Following the above approach and implementing a suitable architecture, many complex problems requiring real-time response, can now be solved using expert systems software. Some of these problems include applications in areas such as robotics, dynamic machine vision, fault detection and diagnosis, signal processing, and process control.

Since the knowledge base is static and relations among the rules are invariant, the developed algorithm is data independent. That is, given a knowledge base, the rules keep their precedence level regardless of the type and quantity of input.

The utilization of the inferences and speed-up of the system are important factors for choosing an appropriate measure of cost for the proposed

architecture.

Limitations :

The suggested design is not applicable to systems containing dynamic knowledge bases. Since it is based on the predetermined relations between the rules. Due to the same reason, the algorithm has to be reapplied every time the knowledge base is modified.

Suggestion for further research :

The design started by mapping an algorithm onto a knowledge base. Another approach would be to map a given knowledge base onto a finite state machine (FSM). Although this is possible, it is very complex and time consuming.

Guaranteed response time is the most important factor in evaluating the feasibility of a real-time system. If the required response time is less than the minimum guaranteed response time (maximum execution time possible when maximum parallelism is implemented), then the system cannot provide real-time response to the problem, and other means for achieving this response are required. Metaplanning may also be applied to this problems to achieve the required real-time response. In fact, parallelism and

metaplanning techniques could be very effective when applied together in a system.

This work did not consider dynamic knowledge bases. A different design may be applied to provide the guaranteed response time.

The algorithm in this thesis was designed based on the assumption of forward chaining as the inference algorithm. The design could be extended to compensate for backward chaining as well as forward chaining. This may be achieved by starting from the goal node and searching for the consequences that cause firing of the corresponding rules.

In our approach we started with the simplest representation of knowledge (rule-based) and implemented maximum parallelism in order to achieve real-time response. This could be extended to all possible knowledge representations. Although the algorithm may not reach the required response time, it does provide measures for reducing the complexity of the search time. In a frame-based knowledge base the antecedents are frames instead of Boolean. This contributes more overhead per inference than the rule-based knowledge bases in order to evaluate the validity of a frame. Thus, the same algorithm could be used to achieve minimum execution time and provide a guaranteed response time.

This algorithm could be applied to a knowledge base linked by a semantic-network. The precedence levels found then evaluate the minimum depth

of the knowledge base. The semantic-network could then be reorganized to result in minimum execution time. The precedence levels also provide a means for distinguishing between a class and its subsets. The highest precedence finds the class of data. An example is presented in Appendix A.

If the knowledge base is presented as formal logic, an important issue is to be able to index directly into relevant knowledge. This is done by searching the entire knowledge base for an entry point. Once the precedence levels are found, then the rules contained in the highest precedence set provide the required entry points. This reduces the search time which is exponentially expensive.

Chapter 6

REFERENCES

1. M.C. Wei and H.A. Sholl, "An Expression Model for Evaluation of Parallelism in Control Structures", IEEE Trans. Computers, Vol. C-31, No.9, pp851-863, 1982.
2. B.A. Bowen and K.O. Siomalas, "Synthesis of Efficient Architectures for Implementing DSP Operations", IEEE Trans. Acoustics, Speech, and Signal Processing, Vol ASSP-33 No. 9, pp 1499-1509, 1985.
3. C.A. O'Reilly and A. S. Cromarty, " 'Fast' is not 'real-time': Designing effective real-time AI systems", SPIE, Vol. 548, Applications of Artificial Intelligence II, pp 249-257, 1985.
4. B.A. Bowen, "An Introduction to the Design of Expert Systems", In preparation by Prentice Hall, 1986.
5. Alain Letendre, "Expert Systems", Office of Industrial Innovation Department of Regional Industrial Expansion, Government of Canada,

1985.

6. F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, "Building Expert Systems", 1983.
7. B.A. Bowen and Haleh Vafaie, "Real-Time Expert Systems", submitted to American Association of Artificial Intelligence (AAAI) for presentation at the Annual Conference, August 1986.
8. John P. Hayes, "Computer Architecture and Organization", McGraw-Hill computer science series, 1978.
9. D.R. Smith, "A Distributed Control Data-flow Machine", Ph.D. Thesis, Carleton University, 1986.
10. M. Lattimer Wright, Milton W. Green, Gudrun Fiegl, and Perry F. Cross, "An Expert System for Real-Time Control", IEEE SOFTWARE, pp 16-24, March 1986.
11. C. Rieger and C. Stanfill, "Real-time Causal Monitors for Complex Physical Sites", proceedings of the First Annual Conference on Artificial Intelligence, Stanford University, 1980.
12. Robert L. Moore, Lowell B. Hawkinson, Carl G. Knickerbocker, and Linda M. Churchman, "A Real-Time Expert System for Process Control", Proceedings of The First Conference on Artificial Intelligence Applications, pp 569-574, Denver, Dec. 1984.

13. D. A. Fortin, T. B. Rooney and E. H. Bristol, "Of Christmas Trees and Sweaty Palms", Proceedings of the Ninth Annual Advanced Control Conference, pp 49-54, West Lafayette, Indiana, 1983.
14. E. Lusk and R. Stratton, "Automated reasoning in Man-Machine Control Systems", Proceedings of the Ninth Annual Advanced Control Conference, pp 41-48, West Lafayette, Indiana, 1983.
15. T. Sakuguchi and K. Matsumoto, "Development of Knowledge Based System for Power System Restoration", IEEE Trans. on Power Apparatus and Systems , Vol. PAS-102, No. 2, pp 320-329, Feb. 1983.
16. B.A. Bowen, J.D. Beaton, and N.M. Standen, "A Conflict Resolution Algorithm for Sparse Airspaces Using A Hybrid Knowledge-Based System", presented to the Canadian Aeronautical and Space Institute in the Sixth Canadian Symposium on Navigation, Vancouver, B.C., May 1986.

APPENDIX A

The appendix contains the program written to provide the precedence levels and the sets of rules that could execute concurrently at these precedences. Two examples are provided to illustrate the results.

```

program config(input,output);
{*****}
{
{ this program finds the rules that could fire concurrently }
{ in a rule based knowledge base. If any rule contains goal }
{ node(s) as a consequent, it is also indicated. The maximum }
{ level of parallelism at each precedence is also found. }
{
{ Constants Definition }
{ ----- }
{
{ RULEMAX : Indicates maximum number of rules possible for a }
{ given knowledge base. }
{
{ MAXIMUM : Indicates maximum number of antecedents and }
{ consequents possible for each rule. }
{
{ WLENGTH : Indicates maximum length possible for an }
{ antecedent or consequent. }
{
{ COLON : Used to delimit end of antecedents in a rule. }
{
{ STAR : Used to delimit end of consequents in a rule. }
{
{
{ Variables Definition }
{ ----- }
{
{ consequence : Is an array of words, used to hold the }
{ consequences of rules in the knowledge base. }
{
{ antecedent : Is an array of words, used to hold the }
{ antecedents of rules in the knowledge base. }
{
{ tempant : Temporary memory to store antecedents of rules }
{
{ tempcon : Temporary memory to store consequents of rules }
{
{ maxirule : Used to indicate the number of rules in the }
{ given knowledge base. }
{
{ rulecount : Used as an index to the rules. }
{
{ indexx : Used to indicate that the rule has already been }
{ given a precedence. }
{

```

```

{ antcount, conscount : Are indexes to antecedent and      }
{      consequent arrays.                                   }
{                                                           }
{ i,j : Integers used for loop indexes.                     }
{                                                           }
{ maxcons, maxant : Are arrays that hold the number of      }
{      antecedents and consequents for each rule           }
{                                                           }
{ reducedrule : Is a boolean array that is used to evaluate }
{      whether all the rules have their precedences        }
{                                                           }
{ flag, concurrent : Are flags used in this program.       }
{                                                           }
{ Parralel : Used for evaluating maximum parallelism        }
{      possible at each precedence level.                   }
{                                                           }
{*****}
{                                                           }
{
const RULEMAX = 100;
    MAXIMUM = 15;
    WLENGTH = 15;
    COLON = ':';
    BLANK = ' ';
    STAR = '*';
    PLUS = '+';
{}
type string = packed array[1 .. WLENGTH] of char;
    words = packed array[1 .. RULEMAX, 1 .. MAXIMUM] of string;
    conarr = array [1 .. MAXIMUM] of integer;
    boolarr = array [1 .. RULEMAX] of boolean;
    intarr = array [1 .. RULEMAX, 1 .. MAXIMUM] of real;
var consequence, antecedent : words;
    goalnode : words;
    tempant, tempcon : string;
    rulecount, precedence, antcount, conscount : integer;
    rule, maxirule, parallel : integer;
    tempind : real;
    counts, i, j : integer;
    indexx, conscf, antcf : intarr;
    maxconseq, maxantec : conarr;
    reducedrule : boolarr;
    nextcharc : char;
    match, flag1, flag, concurrent, reduced : boolean;
{}

```

```

procedure readstring(var word:string);
{
{*****}
{
{ This procedure reads a word containing nonblank charaters. It}
{ is used to read antecedents and consequents of each rule in a}
{ given knowledge base.}
{
{*****}
{
var counter: integer;
    character : char;
begin
    word := BLANK;
    counter := 1;
    repeat
        read(character);
        word[counter] := character;
        counter :=counter + 1;
    until (character=' ') or (counter>WLENGTH);
    if counter > WLENGTH then read(character);
end;
}
}
procedure Writestring(var word :string);
{
{*****}
{
{ This procedure writes nonblank characters of a word}
{
{*****}
{
var counter:integer;
begin
    counter := 0;
    repeat
        counter :=counter + 1 ;
        if (word[counter] <>' ') then write(word[counter]);
    until (word[counter] = ' ') or (counter=WLENGTH);
    write(' ')
end;
}
}
procedure comparev(first,second :string ; var flag :boolean);
{
{*****}
{
{ This procedure check to see if there is a match between}
{ antecedents and consequence}
{
{*****}

```

```

{
    begin
        flag := false;
        if first = second then flag := true;
        end;
    {}
    {}
    {main program}
    begin
        {
            {this segment reads the file and stores the antecedents }
            {and consequences in the proper location }
            {
                rulecount := 0;
            {read all the rules in the given file }
            {
                repeat
                    rulecount := rulecount + 1;
                    antcount := 0;
                    {read the antecedents for this rule}
                    repeat
                        antcount := antcount + 1;
                        readstring (antecedent [rulecount, antcount]);
                        read (nextcharc);
                    {read the certainty factor}
                    if nextcharc = PLUS
                        then begin
                            read (antcf [ rulecount, antcount]);
                            read(nextcharc);
                        end
                        else begin
                            antcf [ rulecount, antcount] := 1
                        end;
                    {end if}
                until nextcharc = COLON;
            { get the number of antecedents in this rule }
            {}
            maxantec [rulecount] := antcount;
            conscount := 0;
            {read the consequences for this rule}
            repeat
                conscount := conscount + 1;
                readstring (consequence [ rulecount, conscount]);
                read (nextcharc);
                if nextcharc = PLUS
                    then begin
                        read( conscf [, rulecount, conscount]);

```

```

        read (nextcharc)
    end
    else begin
        conscf [rulecount, conscount] := 1
    end;
    (end if)
    until (eof) or (nextcharc = STAR);
    if not eof then readln;
    {get the number of consequents as result of this rule }
    []
        maxconseq [rulecount] := conscount;
    until (eof) or (rulecount = RULEMAX);
    {get the number of rules in this knowledge base }
    []
        maxirule := rulecount;
    {initialize}
    for i:=1 to maxirule do
    begin for j:=1 to MAXIMUM do
    begin
        goalnode [i,j] := BLANK;
        indexx [i,j] := 1;
    end; {end of for loop}
    end;
    writeln(' There are ',maxirule:1,' rules in this knowledge base');
    writeln;
    {end}
    {initialize }
    reduced :=true;
    for i:=1 to maxirule do
        reducedrule [i] :=false;
    {end of for loop}
    precedence := 0;
    {end}
    for i:=1 to maxirule do begin
    { find the goal node in the knowledge base}
        for j:=1 to maxconseq[i] do begin
            rulecount := 0;
            flag :=false;
            repeat
                rulecount := rulecount + 1;
                antcount :=0;
                repeat
                    antcount:=antcount + 1;
                    tempant := antecedent [rulecount, antcount];
                    tempcon :=consequence[i,j];
                    comparev(tempant,tempcon, flag);
                until (flag) or (antcount = maxantec[rulecount]);
            until (flag) or (rulecount = maxirule);

```

```

        if not(flag) then goalnode[i,j] :=consequence[i,j];
    end;
end;
rule :=0;
repeat
    reduced := true;
    parallel := 0;
    {}
    { find the precedence levels and the rules that could fire at }
    { each level }
    {}
    for i:=1 to maxirule do begin
        if reducedrule[i] =false
            then begin
                for j:=1 to maxantec[i] do begin
                    rulecount := 0;
                    match := false;
                    flag :=false;
                    flag1 := true;
                    repeat
                        rulecount := rulecount + 1;
                        conscount :=0;
                        repeat
                            conscount:=conscount + 1;
                            tempant := antecedent[i,j];
                            tempcon :=consequence [rulecount,conscount];
                            comparev(tempant,tempcon, flag);
                        if flag
                            then begin
                                match :=true;
                                tempind :=conscf[rulecount, conscount];
                                if (antcf[i,j] <= tempind) or (antcf[i,j]=1)
                                    then begin
                                        if indexx[rulecount,conscount] = 0
                                            then begin
                                                flag1 := false;
                                            end;
                                        end;
                                    else begin
                                        flag1 := true;
                                    end;
                                end;
                            until not(flag1) or (conscount=maxconseq[rulecount]);
                            until not(flag1) or (rulecount = maxirule);
                            if (not(match)) or (not(flag1))then antecedent[i,j]:=BLANK;
                        end;
                    end;
                end;
            end;
        end;
    end;
end;

```



```

writeln ('at precedence ',precedence:1);
writeln ('the following rules can execute concurrently');
writeln;
for i:=1 to maxirule do begin
    counts := 0;
    if reducedrule [i] = false
    then begin
        for j:=1 to maxantec[i] do begin
            concurrent := antecedent [i,j] =BLANK;
            if concurrent then counts:=counts + 1;
        end;
        if counts = maxantec[i]
        { this rule can be fired at this precedence level }
        {}
        then begin
            writeln ('rule ',i:1,' ');
            rule := rule + 1;
            parallel := parallel + 1;
        {}
        { check if the rule contains any goal node }
        {}
            for j:=1 to maxconseq[i] do begin
                if not (goalnode[i,j] = BLANK)
                then begin
                    write(' there is goal node ');
                    Writestring(goalnode[i,j]);
                    writeln( 'as a consequent of this rule:');
                end;
            { remove the rule by removing the consequents }
            indexx [i,j] := 0;
            end;
            reducedrule [i] := true;
        end;
    end;
    reduced := (reduced) and (reducedrule[i]);
end;
writeln;
write('Maximum parallelism possible at this level is ');
writeln(parallel:1,'. ');

precedence :=precedence + 1;
writeln;
writeln('-----');
writeln;
{ check if all the rules have been counted for }
    flag :=(reduced) and (rulecount=maxirule);
until (reduced) and (rule=maxirule);
writeln;
end.

```

Example 1: A sample knowledgebase with rules that contain certainty factors is presented.

The rules are:

```
IF A THEN C
IF A AND H AND R THEN B(0.4)
IF B(c.f. > 0.1) THEN E AND B(0.6)
IF J THEN P(0.4)
IF B(c.f. > 0.5) AND J THEN D(0.2)
IF D AND P AND T AND S THEN M AND D(0.5)
IF D(c.f. > 0.4) THEN N
IF C AND K AND N THEN F
IF F THEN L
```

Then the input and output of the program are as follows:

Input:

```
A :C *
A ,H ,R :B +0.4*
B +0.1:E ,B +0.6*
J :P +0.4*
B +0.5,J :D +0.2*
D ,P ,T ,S :M ,D +0.5*
D +0.4:N *
C ,K ,N :F *
F :L *
```

Output:

There are 9 rules in this knowledge base

at precedence 0
the following rules can execute concurrently

rule 1
rule 2
rule 4

Maximum parallelism possible at this level is 3.

at precedence 1
the following rules can execute concurrently

rule 3
there is goal node E as a consequent of this rule.

Maximum parallelism possible at this level is 1.

at precedence 2
the following rules can execute concurrently

rule 5

Maximum parallelism possible at this level is 1.

at precedence 3
the following rules can execute concurrently

rule 6
there is goal node M as a consequent of this rule.

Maximum parallelism possible at this level is 1.

at precedence 4
the following rules can execute concurrently

rule 7

Maximum parallelism possible at this level is 1.

at precedence 5

the following rules can execute concurrently

rule 8

Maximum parallelism possible at this level is 1.

at precedence 6

the following rules can execute concurrently

rule 9

there is goal node L as a consequent of this rule.

Maximum parallelism possible at this level is 1.

Example 2: This example presents the usefulness of the program in organizing a knowledge base linked by a semantic network. If the knowledge base is derived from queries, and presented in the form of a list of production rule, then the rules found at each precedence level provide the links in a semantic network.

The input to the program is shown below. The output achieves minimum depth possible for the knowledge base and the minimum links between each value. This is illustrated in Figure A.1.

Input:

feather :bird *
flies ,layegg :bird *
hair :mammal *
milk :mammal *
eatmeat :carnivore *
mammal ,hoofs :ungulate *
pointteeth ,claws ,forwardeye :carnivore *
mammal ,carnivore ,tawnycolor ,darkspots :cheetah *
ungulate ,longneck ,longlegs ,darkspots :giraffe *
ungulate ,blackstrip :zebra *
bird ,notfly ,longneck ,blackwhite :ostrich *
bird ,notfly ,swim ,blackwhite :penguin *
bird ,flywell :albartoss *

Output:

There are 13 rules in this knowledge base
at precedence 0
the following rules can execute concurrently

rule 1
rule 2
rule 3
rule 4
rule 5
rule 7

Maximum parallelism possible at this level is 6.

at precedence 1
the following rules can execute concurrently

rule 6
rule 8
there is goal node cheetah as a consequent of this rule.
rule 11
there is goal node ostrich as a consequent of this rule.
rule 12
there is goal node penguin as a consequent of this rule.
rule 13
there is goal node albartoss as a consequent of this rule.

Maximum parallelism possible at this level is 5.

at precedence 2
the following rules can execute concurrently

rule 9
there is goal node giraffe as a consequent of this rule.
rule 10
there is goal node zebra as a consequent of this rule.

Maximum parallelism possible at this level is 2.

END

0|8.0|8.8 7

FIN