Cogitator

A Parallel, Fuzzy, Database-Driven Expert System

THESIS

Submitted in Fulfilment of the Requirements

for the Degree of

MASTER OF SCIENCE

Rhodes University

by

Paul Baise

January 1994

Acknowledgements

No-one completes a thesis without help from others. I would like to thank both my supervisors, Peter Clayton and Dr. Murali for their financial support and useful comments. To Alfredo Terzoli for his helpful hints and the fruitful discussions that we had. To my fellow graduate students, their amicable characters made the atmosphere all the more pleasant. And lastly, to my dad, without whom I would not be able to read for this degree.

Abstract

The quest to build anthropomorphic machines has led researchers to focus on knowledge and the manipulation thereof. Recently, the expert system was proposed as a solution, working well in small, well understood domains. However these initial attempts highlighted the tedious process associated with building systems to display intelligence, the most notable being the Knowledge Acquisition Bottleneck. Attempts to circumvent this problem have led researchers to propose the use of machine learning databases as a source of knowledge. Attempts to utilise databases as sources of knowledge has led to the development Database-Driven Expert Systems. Furthermore, it has been ascertained that a requisite for intelligent systems is powerful computation. In response to these problems and proposals, a new type of database-driven expert system, Cogitator is proposed. It is shown to circumvent the Knowledge Acquisition Bottleneck and posess many other advantages over both traditional expert systems and connectionist systems, whilst having non-serious disadvantages.

Contents

Chapter 1

Introduction

Expert systems play a prominent role in AI. They have been successfully applied to problem solving in many fields including medicine, exploration, system configuration and planning. However, their continued progress and acceptance appears to be threatened by the classical *Knowledge Aquisition Bottleneck* (KAB). Current trends indicate that expert systems of any real use will have to posess large knowledge bases - for instance MCC's CYC knowledge base [Lenat & Sheperd, 1990] that seeks to embed all common sense knowledge in a knowledge base is to posess 10^8 facts and rules - exacerbating the problem. The resulting largesse of the KAB led researchers to suggest databases as alternative sources of knowledge in an attempt to circumvent the KAB. This has spawned the development of Database-Driven Expert Systems (DDES's). Large databases are commonplace today.

The recent great advances in concurrency have spawned the design and development of new hardware architectures and software designs and is generally accepted as the only viable alternative to increasing the performance of computer systems.

Fuzzy Logic has been conclusively shown to efficiently describe and control many complex real-world systems. It has also been shown to correlate closely with Human Information Processing (HIP) and there have even been arguments for its use in legal procedures [Kosko, 1992]. Furthermore, *similarity* is immanent to HIP and to complex systems in general. People react similarly to similar situations or stimuli, as do many natural systems. Similarity is thus a pervasive quality.

Consequentially, a new type of expert system that bypasses the KAB, precludes many other problems associated with traditional expert systems, utilises the ever-advancing power of concurrency, the descriptive capabilities of fuzzy mathematics and the immanence of similarity is proposed. Called *Cogitator* and deemed to be a dialect of Case-Based

Reasoning (CBR), the system seeks to emulate HIP according to the cognitive theory of Case-Based Reasoning (CBR) and the HIP descriptive capabilities of fuzzy mathematics. Furthermore, the use of concurrency to improve performance and by using databases, complete decomposability is possible simplifying the concurrency aspect and yielding a scaleably accurate system that is virtually database independent and therefore commercially viable.

The next chapter introduces the main ideas associated with concurrency. The different classes of concurrent architectures, performance and compiler issues, programming envirnoments and applications are briefly considered. Thereafter, a short chapter introduces the main characteristics associated with expert system such as knowledge and knowledge acquisition, representation, validation and management.

Chapter 4 illustrates an emerging field formed from the confluence of concurrency and AI : Concurrent AI, in particular, Concurrent Expert Systems. The different types of concurrent expert systems are introduced and briefly discussed. Emphasis is placed on the problems associated with concurrent expert systems as it forms a major thrust for the development of the concurrent expert system paradigm proposed in this thesis: *Cogitator*.

Chapter 5 introduces fuzzy logic. Some elemenntary fuzzy theory is initially introduced whereafter, fuzzy measures and membership functions, to be extensively utilised by the paradigm proposed in this thesis, are illustrated. Finally, fuzzy expert and control systems are introduced and some sample applications thereof are given.

In Chapter 6, a new dialect of Case-Based Reasoning, *Memory-Based Reasoning*, is illustrated. Memory-Based Reasoning (MBR) forms the basis for the system proposed in this thesis, in fact *Cogitator* is a fuzzy Memory-Based Reasoner. Initially, Case-Based Reasoning is introduced whereafter the main aspects of MBR are considered.

Finally, chapter 7 introduces and illustrates the system proposed in this thesis: *Cogitator*. The membership functions it utilises, the incorporation of fuzzy logic as well as its operation and an illustrative implementation using a popular concurrent programming paradigm, are considered. Finally, the advantages of *Cogitator vis-a-vis* traditional rule-based expert systems and connectionist systems, are given.

Chapter 2

Concurrency

Concurrent computing is here to stay. Developed extensively over the last two decades, it has influenced almost all aspects of computing. This chapter seeks to briefly introduce concurrency and illustrate its necessity for significantly improving performance. Later chapters illustrate how the confluence with AI has given rise to Concurrent AI, in particular, concurrent expert systems.

2.1 Origins

Despite convincing evidence that *Concurrency*[1] is a 20th century development, the earliest reference to it is believed to be in General L. F. Menabreas' publication in the *Biblioteque Universelle de Geneve*, in October 1842 entitled *Sketch of the Analytical Engine Invented by Charles Babbage*[2]. Listing the utility of the analytical engine, he writes:

"*Secondly, the economy of time: to convince ourselves of this, we need only to recollect that the multiplication of two numbers, consisting each of twenty figures, requires at the very utmost three minutes. Likewise, when a series of identical computations is to be performed, such as those required for the formation of numerical tables, the machine can be brought into play so as to give several results at the same time, which would greatly abridge the whole amount of the processes.*"

Nevertheless, it does not appear that the ability for concurrent execution was incorporated into the design of the Analytical Engine, but that the idea had occurred to Babbage

[1] Concurrency is used as a collective term to encompass the technically different descriptors Parallel Processing, concurrency and multiprocessing. These terms are defined later. In the sense used here, it refers to the traditional definition : the ability to execute many processes at once.

[2] In designing his Analytical and Difference engines, Charles Babbage pioneered the many notions of computing. His endeavours resulted from his desire to compile reliable astronomical tables.

about a hundred years before the technology matured [Hockney & Jesshope, 1981]. Nevertheless, in his designs, Babbage rejected serial arithmetic owing to the resulting long execution times and designed the analytical engine to perform arithmetic on fifty decimal digits in simultaneously is significant [Hockney & Jesshope, 1981]. Consequentially, the initial ideas and uses of concurrency concerned performing arithmetic on each digit simultaneously [3].

According to [Hockney & Jesshope, 1981], the architecture of most of the early and subsequent electronic computers may be traced to the concepts outlined by in a 1946 paper of Burks, Goldstein and von Neumann, entitled "Preliminary discussion of the logical design of an electronic computing instrument", the idea of a serial computing machine that subsequently became known as the "von Neumann Architecture" originated [4].

The fundamental characteristics of the von Neumann architecture were that both data and programs are stored in memory and that programs be able to alter themselves during execution, as is manifest in modern computers. The other main characteristic is that there be one main processor.

However, having only one main processor, and consequently one processor to memory interface through which instructions and data flow, creates a bottle neck that has became known as the "von Neumann bottleneck". Furthermore, the belief that many processors could perform a task or tasks quicker than a single processor heralded the advent of concurrent computers. This duplication of processors was (and still is) believed to preclude the limitations associated with the von-neumann architecture. From these initial ideas grew the area of concurrency.

The first concurrent computer was the Harvard Mark I or Automatic Sequence Controlled Calculator, a 52-by-8 foot electro-mechanical monster. First proposed by Howard Aitken in 1937, the rationale behind its design was that the automation of complex or extended computing could be achieved by linking a number of automata in a network and arranging the data to be passed around the network and operated on by the automata in a pre-set sequence. This is essentially the notion of pipelining. The chronological evolution of concurrent computers may be found in any of the multitude of books on concurrency.

Computer technology may be conceived of as developing in a series of generations, each

[3]This has been perpetuated in today's microprocessor architectures.

[4]Nevertheless, the first stored-program electronic computers, the EDSAC (1949) and the EDVAC (1952), performed bit- serial arithmetic. Considering Babbage's ideas and designs, this seems odd, but may be attributable to the need for a working electronic computer and consequentially simple circuitry used to implement bit-serial arithmetic.

characterised by some important development. The first generation was characterised by the development of vacuum tubes during the mid 1900's, the second, third and fourth generations were characterised by the development of, respectively, transistors, Integrated Circuits (IC's) or Large Scale Integration (LSI) and VLSI technologies. The fifth and recently announced sixth generations marked a general departure from hardware innovations to software, they are characterised, respectively, by thinking machines based upon formal logic (PROLOG) and biological/adaptive computing based upon soft (fuzzy) computing. See [Baise, 1993] for more information.

The 1980's heralded the advent of large scale parallelism as a principal innovation that has extended into the 1990's, and is bound to extend further.

2.2 Introductory Concepts

Firstly, it is significant to delineate between multiprocessing, parallelism and concurrency, since these terms are often incorrectly and interchangeably used.

Multiprocessing implies "*consisting of or having many*", with no restriction on the constituent parts. It refers to the use of two or more PE's with a common memory, but each executing a different program. Multiprocessing means one application per processor.

Parallel Processing or Parallelism means "*like in essential parts*" wherein a process is or processors are replicated at least once. It refers to the dividing up of one application among many processors. This may be accomplished in one of many ways:

- One processor doing many jobs
- N processors doing one job
- N processors doing M jobs.

Also, a job may be thought of as being performed on

- one processor
- one processor with many processing units
- many processors connected by links.

From these different ideas, various parallel architectures may be derived.

Parallel processors speed up the computation of a *single* job while multiprocesors handle *more* jobs at a given time. Multiprocessing improves productivity, while parallel processing accomplishes a single job quicker.

Concurrency encapsulates both parallel and multiprocessing, meaning that an application is concurrent if it may be implemented in a parallel or multiprocessing manner if a suitable algorithm may be found.

In *Symmetric Multiprocessing*, all the processors are homogenous and generally no one processor exerts control over any other one.

Asymmetric Multiprocessing is essentially the master-slave architecture where one (or more) master processors control all the processors in the computer system.

The move from sequential programming and computers to their concurrent counterparts is most certainly non-trivial. When parallelism is introduced, the problems of deadlock, livelock, race conditions and unwanted data sharing arise.

Deadlock occurs when two (or more) processes cannot proceed since they are interdependent and each need the other's next computed value in order to proceed.

Livelock occurs when two (or more) processors repetitively perform their (respective) same tasks since the flags or values needed to cause them to proceed to another task never occur.

When a shared memory scheme is implemented, *Race Conditions* occur when a process needs to use a value computed or updated by another process and fetches the value before it has been properly computed or updated. It thus fetches the wrong value leading to problems during execution.

Unwanted Data Sharing arises when data that should be local to a certain process is accidentally accessed by another process and, possibly, mistakenly modified.

Many techniques have been developed to detect program constructs that could lead to any of the aforementioned problems so that they may be precluded from occurring.

Parallelism may be introduced in either of three principal ways :

(1) Pipelining where assembly line techniques are used to improve the performance of an arithmetic or control unit

(2) Functional where several independent units that perform different functions such as arithmetical and logical. These units operate on different data, and

(3) An Array of identical processing elements under common control performing the same operation simultaneously on different data (i.e. in lockstep).

2.3 Architectures and Classifications

The notion of increasing processing power by replicating processors has spawned the development of different architectures and the classifications thereof.

Architecture refers to the *granularity* of the processing elements (PE's) , their topological organisation (interconnection network) and the distribution of control across the PE's.

The *Granularity* a PE refers to its processing power and capabilities, ranging from single bit processors (fine) to general purpose ones (coarse). *Topology* refers to the pattern and density of the interconnection pattern of the PE's which range from lightly connected (sparse) to heavily connected (dense). *Control* refers to the allocation of tasks and their subsequent synchronisation, ranging from loose to tight.

Attempts to classify concurrent aomputer architectures has led to three classification schemes, those of Flynn, Enslow and Shore of which Flynn's is the most popular.

[Flynn, 1966] devised a taxonomy outlining four classes of concurrent computers that have become standard terminology in concurrency. Based upon the nature of the instruction stream (the sequence of operators) and the data stream (sequence of operands), Flynn determined four classes according to the multiplicity of these two characteristics. The four classes are *Single Instruction Single Data* (SISD), *Single Instruction Multiple Data* (SIMD), *Multiple Instruction Single Data* (MISD) and *Multiple Instruction Multiple Data* (MIMD).

SISD refers to the traditional von Neumann uniprocessor computers, the decoding of only one instruction per execution cycle.

MISD refers to the simultaneous operation of many instructions operating upon a single duplicated datum, i.e. many processors operating on a single data stream. Applications exhibiting this type of organisation include Digital Signal Processors (DSP) where different solution algorithms operate simultaneously on the same data stream. Another definition of MISD includes pipelined processors which, like a production line, comprises stages each of which operates on the data and then passes it onto the next stage. The veracity of this latter definition is somewhat questionable, since pipelined machines tend to resemble

MIMD-type machines (see below) .

SIMD refers to a single duplicated instruction operating simultaneously on different data. These machines are also termed array processors in which the PE's are identical and are synchronised by a single master processor to operate in lockstep.

In *MIMD*-type organisations, a collection of instructions operates simultaneously on a collection of different data items. Each PE has its own independent instruction and data stream and is more general purpose than those of SIMD machines. The PE's may be synchronised or not and the amount of communication is minimal. MIMD is the most popular organisation.

Another taxonomy, based upon the degree of coupling between processors in multiprocessor arrays, is that of Enslow [Enslow, 1977] *Coupling* is the method by which PE's communicate, by network links, buses or shared memory. Coupling may be physical or logical ranging from loose to tight.

Loose Coupling implies a lack of direct sharing of process address space, no sharing of primary memory. On the physical level this means that communications are message-based , and on the logical level, that the processors are autonomous. Collective tasks are completed cooperatively.

Tightly Coupled systems permit shared memory access and inter- processor communications are word by word. On the physical level, there is an overlapping address space, while, logically, the processors are synchronised in that one processor may at its discretion exercise control over another.

It should be noted that it is possible to create systems that are physically loose, yet logically tight, for example, Transputer arrays with synchronised processors for real time applications.

Another taxonomy based upon a hierarchial classification according to how a machine is organised into its constituent parts, is attributable to [Shore, 1973]. Six classes of machines were identified and assigned a numerical designator. This classification scheme is not widely used, if at all. Furthermore, the numerical classes are somewhat archaic in that a biologist will not describe a certain plant as of class I; rather, a more descriptive linguistic or acronymic name would be chosen. Flynn's taxonomy is the most popular and informative and is used throughout this thesis.

The architectures of parallel and multiprocessors are numerous and varied. Figure 2.3 overleaf is a brief taxonomy of them. More information on each may be gleaned from any of the many books on concurrency.

Certain combinations of two of the four Flynn classes mentioned above, give rise to certain hybrid architecture classes. [Almasi & Gottlieb, 1989] describe a class called *MSIMD* (Multiple SIMD) Tree Machines and *MSIMD* Reconfigurable Machines. They also classify VLIW architectures as hybrid.

MSIMD Tree Machines are geared towards more massive parallelism than MIMD. They have very many simple processors (10^6 PE's for the two machines that have been built) [5] that cannot store their own programs. Two such computers are the Non-Von and the Cellular Computer.

MSIMD Reconfigurable Designs employ a circuit switched network to enable programs to have their resources dynamically reconfigured to fit the structure of (primarily scientific) problems. Two prototype machines have been built, the *TRAC* and the *PASM* having 8 and 16 processors respectively. However the goals are machines with more processors (1024 for the PASM). They may thus be considered to be in the experimental stage.

Considering this panoply of architectures, which one may be deemed to be the most apposite for all applications? It is generally accepted that no one single architecture offers the best solution for all applications. Some architectures are more appropriate for certain types of applications; e.g. array processors are efficient for spatial problems such as fluid flow analysis whereas a vector or systolic processor would be appropriate for computing Fast Fourier Transforms, which require many inter-processor communications. [Desrogers, 1987] states that

"There is ... no best parallel or multiprocessor organisation; each must be judged upon its benefits and weaknesses relative to the intended application".

2.4 Performance Issues

To ascertain the performance of concurrent machines is important since one needs to know whether these architectures actually do offer meaningful speedup. It is generally difficult to determine the performance of a uniprocessor machine, let alone a concurrent one, owing to a multitude of factors. Before discussing them, some preliminary concepts need to be introduced.

[5]Compare to the $6 * 10^3$ (64K) processors of the Connection Machine CM-2.

Architecture	Topology (Connectivity)	Control	Granularity	Classification (Flynn)
Multiprocessor	Light	Loose-Medium	Medium-Coarse	MIMD
A centralised operating system and autonomous processors				
Vector Processors	Medium	Tight	Medium	SIMD
Contains functional units each of which performs the same operation on all elements of the vector simultaneously. Has a vector and scalar unit.				
Pipeline Processors	Medium	Tight	Fine-Medium	MIMD/MISD
Applied to vector machines. Has a number of stages each of which performs a specified operation within a specified time period. Pipeline may be reconfigurable (multi-function) or not (single function). e.g. the Harvard Mark I.				
Array Processors	Heavy	Tight	Fine	SIMD
PE's arranged in a rectangular grid connected to the four nearest neighbours.				
Systolic Processors	Medium-Heavy	Tight	Fine	MISD
Pumps operands in one end, operates on them and pumps the results out the other end. Precludes I/O and compute bound problems. An extension to the concept of pipelining (multidimensional and multidirectional flows. Great for special purpose applications such as Fast Fourier Transforms.				
Cubes	Medium	Loose-Tight	Medium	MIMD
N processors where N is a power of 2. PE's located at the corners of the cube, the interconnections forming the cube edges. Each PE is located such that its binary numeric address differs from those of its neighbours by one bit. Hypercubes have a dimension greater then 3; i.e. at least $2^4 = 16$ processors.				
Associative Processors	Heavy	Tight	Fine	SIMD
Data is content addressable - no physical/logical address needed, retrieval is based upon the contents of the word. Operations performed on data in parallel. Associative memory array searched concurrently for a match. Ideal for implementing certain database searches since searches are O(1). e.g. Goodyear STARAN.				
Interconnected Network Processors	Light-Heavy	Loose-Tight	Fine-Coarse	MIMD
PE's are connected via some data routing network with a capacity to transfer many items simultaneously (eg butterfly and shuffle networks). Appropriate for problems comprising many interacting elements such as Fast Fourier Transforms. eg. Transputer networks.				
Data Flow	Light-Medium	Loose-Medium	Medium	MIMD
Based upon the idea that a sequence of data should control the machine and not the instructions. Operations occur when operands are ready to be operated on. Comprises separate functional units that may simultaneously operate on the data. Systems are functionally or single assignment programmed.				
Very Long Instruction Word (VLIW)	Light-Medium	Tight	Medium	MIMD (no general purpose PE's)
Rather than replicate processors, the instruction processing portion is widened by an order of magnitude relative to the usual instruction length. Conceptual to date owing to difficulties in programming such machines [Desrogers, 1987].				

Table 2.1: The Characteristics and Classification of Concurrent Architectures according to Flynn's Taxonomy

Concerning performance, computers may be classified as either *Compute* or *I/O bound.* A machine that is compute bound typically has its processor(s) unable to process data or instructions in accordance with the speed with they arrive along the buses or networks. For I/O bound machines, the bandwidth of the buses or networks cannot cope with the I/O capabilities of the computer's microprocessors.

The *Peak Rate* of a computer system (the one that manufacturers readily quote) is the maximum computational rate that may be theoretically achieved when all modules are fully utilised.

The *Sustained Rate* of a computer system is the rate achieved for the solution of a particular task.

The performance of computers is typically measured in *Floating Point Operations per Second* (FLOPS) and *Instructions Per Second* (IPS). For powerful computers, the terms Millions of FLOPS (MegaFLOPS) and Millions of IPS (MIPS) are often used.

The *Speedup* of machine I over machine II is given by the ratio $\frac{t_I}{t_{II}}$ where t_I and t_{II} are, respectively, the times taken to execute the same program on machines I and II.

It seems obvious that the amount of parallelism that may be extracted from a particular problem depends upon the inherent sequentiality in the algorithm designed to solve the problem. This notion is formalised in Amdahl's law [Amdahl, 1967] which states that the inherent sequentiality is the ultimate limiting factor of parallelism on any machine. There have been criticisms of this law as being unrealistic [Hockney & Jesshope, 1981]; nevertheless, it serves as a general indication as to the appropriateness of a certain architecture to a certain problem.

There are certain performance criteria that pertain to specific architectures. For instance, concerning pipeline computers, there are the notions of *Half Performance Length* and *Vector BreakEven Length*, and for interconnected network processors, there is latency.

Half Performance Length refers to the length of a vector that produces half the throughput while *Vector Break-Even Length* is the minimum vector length that makes operating in vector mode more efficient than operating in scalar mode. *Latency* refers to the time required to set up the communication between any two processors. It varies from computer to computer.

To ascertain the performance of various computers, programs that test certain computational aspects of computer systems have been developed. In addition, certain mathematical theories, such as that of Petri Nets have been developed ascertain and predict the performance of various computers.

A *Benchmark* is a program that, via a series of computations, operations and actions, seeks to provide a relative performance index of a computer's performance. These programs are often written in a high level language such as Fortran and contain code to test various aspects of a computer system such as floating point and looping.

Such programs must exhibit three characteristics: they must produce an intelligible result, the result should be repeatable and should state what aspects of a computer it tests.

The two most common benchmark programs are the *Dhrystone* and *Whetstone* which, respectively, test the integer and floating point capabilities of a computer. The Whetstone was the first benchmark and was succeeded by the Dhrystone.

The *Whetstone* program contains code for floating point, array manipulation as well as looping and subroutines.

The *Dhrystone* program contains code for pointer manipulation, array referencing, and string manipulation and record access. The relative performance returned by this program is measured in Dhrystones.

The latest benchmark is the *SpecMarks* program, which contains code to test many aspects of computers and then combines these into an aggregated relative index.

A common program to rate basic speed without procedure calls is the *Sieve of Eratosthenes*, based upon Erastothenes' algorithm for calculating prime numbers up to a certain integer, suchas 16 000.

However, the result of a benchmark program cannot be seen to give a true indication of a computer's speed. Owing to compiler efficiencies, operating system overhead, caching, the language used, the nature of the problemas well as its algorithm and the architecture of the computer, an accurate measure is difficult to obtain for uniprocessor systems, let alone multi and parallel processing systems. One solution is to use many programs and use a form of averaging to arrive at an aggregated index, or to disable all caching since benchmark programs are typically fairly small and may easily fit into the cache yielding an unrealistic result. Another solution is to develop specific benchmarking programs to test the efficiencies of various multi- and parallel processing systems on a range of problems with different natures should be designed. For instance, spatial or data-parallel problems such as fluid flow using cellular automata map very well to SIMD-type computers. Alternatively, problems exhibiting a large degree of locality and are non-uniform map better to pipeline or vector machines. There should be a collection of tests to test both spatial and local information-based problems.

Some people adopt the simplistic view that adding N processors should reduce the execution time by a factor of N. However, there are a multitude of factors that refute that idea, such as the nature of the problem, the target architecture, inter-processor communication overhead, the number of and the type of processors. Very seldom is a linear speedup achieved, and mostly the speedup is very sub-linear. Sometimes, usually owing to communications Overheads (eg. latency), there may even be a slowdown.

As may be deduced, measuring the performance of a concurrent computer is no easy task. Manufacturers and testers seem to still argue over the performances reached by certain concurrent computers.

2.5 Detection and Transformation

For programs to execute with a reasonable degree of efficiency on a concurrent computer, the concurrency in programs needs to be appropriately detected and utilised.

Concurrency in a program may be classified as either Explicit or Implicit. Explicit concurrency is the conscious placement by the programmer of certain language statements to indicate concurrency. Implicit concurrency is the concurrency that may be determined in a program without any conscious placement of specific language constructs to indicate parallelism.

The theory of dependency analysis was introduced by Professor David Kuck for the detection of concurrency and has subsequently been used in register allocation and memory hierarchy management.

Loops and arrays commonly occur in programs and form the main target for transformation.

Nevertheless, millions of lines of Fortran code written over the last 30 years for sequential machines. With the realisation that concurrency offers the only means of significant speedups, instead of rewriting all this code for concurrent computers, why not transform these sequential codes into parallel form? This notion has necessitated research and development in implicit concurrency detection and transformation.

Owing to the vagaries associated with concurrent programming, many techniques have been developed for the automatic detection of concurrency in programs such as loop unfolding and statement substitution. However, it is generally accepted that these techniques are limited by the language used, the nature of the problem, the language used and the target architecture and that only partial solutions may be expected in the near future [Zima & Chapman, 1991]. Implicit detection is only partially explicit parallelism is needed to really attain maximum performance therefore there is an emphasis on the programmer to consider these characteristics. This has led to the development of special-purpose concurrent languages such as Parallel C and Occam as well as parallel algorithms and programming environments that provide the construction of powerful tools for the detection of parallelism.

Automatic restructuring compilers (supercompilers) have been developed (cf. Cray Research's new CFT Fortran compiler). They are termed autotasking compilers since they detect the parallel/vector constructs in programs and transform them to an appropriate parallel/vector form thereafter generating parallel/vector code.

[Zima & Chapman, 1991] state that knowledge-based systems may play an important part in the detection of concurrency. On the relative difficulties of parallelism and vectorisation [6] Zima and Chapman [Zima & Chapman, 1991] state that

> *... while vectorisation has evolved into a well-understood technique and many of the principal problems of parallelisation have been successfully attacked, the variety and structural complexity of parallel systems has so far prevented a general solution of the parallelisation problem.*

and that

> *In general, the complexity of the relationships between the algorithmic structure and the transformation strategies prevents global optimal solutions ....*

2.6 Programming Paradigms and Environments

Methods for the representation and detection of parallelism have been described. In accordance with the difficulties associated with concurrent program development, certain concurrent programming paradigms environments for parallel program development and specification have been developed. Typically these environments contain support tools to allow detailed specification and coding of the tasks to be implemented as well as inter processor communication protocols and concurrent architecture desired.

[6] Zima and Chapman distinguish between parallelism and vectorisation to illustrate the relative difficulties of the detection and transformation methodologies pertaining to each. Usually, though, parallelism is thought to supersume vectorisation.

[Desrogers, 1987] mentions Snyder's POKER system and another system developed at Rice University. In POKER, a user specifies computations and maps them onto a specified parallel graph using a graphical tool to arrange the graph's nodes and edges. Rice University's system was designed to allow the development of large parallel programs without having to recompile the whole program when an addition or correction was made. It contains an intelligent Fortran editor, an intelligent debugger and an optimising compiler (as well as a vectoriser) to aid program development.

There have been a few concurrent programming paradigms that have been developed, the most popular of which appears to be the *Linda* paradigm developed at Yale University. Linda is a simple, yet powerful, concept in which certain simple functions are added to a traditional programming language to facilitate concurrent computation. Central to Linda is a Tuple space representing a shared memory through which communication between processes (sending variables or messages) occurs using the Linda constructs embedded into the traditional language. Versions of Linda have been ported to many platforms including Transputers and Hypercubes. *Linda* will be discussed more completely in chapter 5.

2.7 Applications

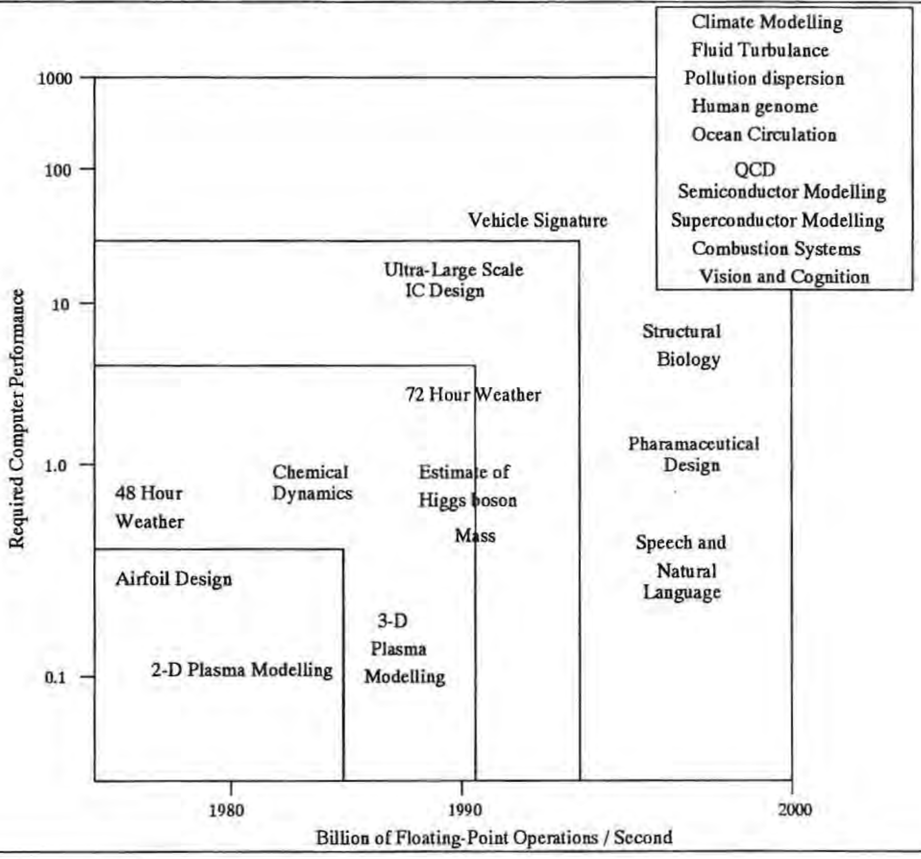
Almost 10 years ago, the Nobel physicist, Kenneth Wilson, suggested compiling a list of projects that presented grand challenges to researchers and their supercomputers.

Currently, the list is extensive including projects from areas such as molecular biology (designing protein structures), chemistry (understanding catalysis), physics (quantum chromodynamics), medicine (drug design and interaction), geography (weather prediction) and engineering (designing aircraft).

In 1991, the US Senate ratified the High Performance Computing and Communication (HPCC) initiative which is to investigate these grand challenges.

The key to these "grand challenges" is visualisation, to graphically see how the phenomena being studied appear and evolve. All of these "grand challenges" demand phenomenal computing resources. The convenience provided by a "teraflop" machine - one capable of performing one "trillion" floating point operations per second has been mentioned often [Zorpette, 1992b]. For instance, a TeraFlop machine would allow the visualisation of air flow over a "whole" aircraft (and not just certain parts of it) and the modelling of global weather. According to [Zorpette, 1992a], certain manufacturers claim that they will be able to achieve this performance by the end of the decade. Below is a figure illustrating the various grand challenges, their estimated processing requirements and the expected year of solution. Note the presence of speech and natural language processing as well as computer vision and cognition, major aspects of AI.

Figure 2.1 The Estimated Processing Requirements and the proposed date of solution of the Grand Challenges. (Source : [Grossmann, 1992])



2.8 Conclusion

Given the problems and difficulties associated with concurrency, why pursue the field so fervently? Why concurrent machines? Why won't very powerful serial machines suffice? The performance of serial computers is limited by the familiar "von Neumann Bottleneck" (serial path used to move instructions and data between memory and the CPU). Consider the state of present and a prognostication of future performance-enhancing technologies. The physical limits of semiconductor fabrication and design are being reached in that processors cannot be made dense enough such that the minimal time delays that allow for high speed may be facilitated. Furthermore, highly dense chips of exotic materials packed close together present difficult cooling and packaging problems [Walz, 1990].

Over the next decade, semi-conductor technologies such as Gallium Arsenide, the Quantum Flux Parametron (QFP) and the Josephson Junction (JJ) (Gallium Arsenide currently provides a speedup factor of three) and further miniaturisation (VLSI technology is generally acknowledged as nearing its limits concerning miniaturisation) will increase performance by a factor of five [Walz, 1990].

[Walz, 1990] also mentions that clever caching and instruction prefetch techniques will provide a speedup factor of two and the use of multiple functional units a factor of four thus theoretically increasing the fastest uniprocessors from the current 1 GFlop to about 40 GFlops. Further, compiler technologies could allow as many as sixteen such processors to be linked together thus producing a theoretical peak performance of 640 Gflops [Walz, 1990]. Also, higher clock rates facilitated by better cooling technologies and advances in semiconductor design could further increase processing power by a factor of two. Consider DEC's new *Alpha* [7] chip that is being designed to run at 200 MHz [IEEE, 1992].

The most powerful massively parallel machines already exceed the power of today's fastest sequential machines : the 65 536 processor CM-2 is realistically capable of speeds in the order of 5 GFlops with a peak performance of 28 GFlops. Further, the Defense Advanced Research Projects Association (DARPA) has targeted a massively parallel TeraOps (Trillion operations per second) machine by 1995 at a cost of less than $ 100 million

The main ideas and problems associated with concurrency have now been introduced and the reasons for the advent, continued use, research and development of concurrent computers have been stated and substantiated. At this stage, the reader should be convinced that concurrency is here to stay and does offer a viable alternative to serialism in computing for higher performance. A confluence of the notions developed here and those of Artificial Intelligence (AI) initiated the field of Concurrent AI. The main ideas associated with expert systems, in particular their functioning and the problems associated with building them. Thereafter, chapter 3 seeks to introduce the confluence of concurrency and AI - concurrent AI - with particular emphasis on concurrent expert systems and the problems

[7]DEC's new Alpha chip is listed in the Guiness Book of Records as the fastest microchip for 1992.

associated with building them.

Chapter 3

Expert Systems

Expert systems play a prominent role in AI. They have been successfully applied to problem solving in many fields including medicine, exploration, system configuration and planning. After a brief historical account of their origins, the aspects of expert systems' design and functioning, viz. architecture, the process of knowledge organisation, acquisition and validation, as well as their associated vagaries are briefly introduced and discussed.

3.1 Origins

In the late 1600's, Gottfried Wilhelm Leibnitz [1] sought to build an all-purpose calculating machine, becoming the first person to consider building a machine to mechanise thought. Leibnitz considered reasoning to be the identification of identities among ideas. His primary motivation was that, being a diplomat and lawyer, he encountered the problems that could arise from disagreement, especially on the political level. He ascribed the problems to the imprecision of natural language and wondered if, by mechanising the thought process, disagreement would be eliminated. As he wrote

> *"If controversies were to arise, there would be no more need of disputation than between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, to sit down to their slates and to say to each other (with a friend to a witness, if they liked): let us calculate"*

[Pratt, 1987a].

[1] Gottfired Wilhelm Leibnitz [1646-1716]. Mathematician, philosopher, lawyer and diplomat. Independent inventor of the calculus, he is certainly one of the greatest scientists.

This was the notion behind Leibnitz's project for the mechanisation of thought, the first ideas of an *expert system.* Leibnitz may thus be considered to have had the first ideas to build a *Thinking Machine.* This becomes apparent from his writing:

> *" ... how much better will it be to bring under mathematical laws human reasoning, which is the most useful and excellent thing we have. Logicians are not to be blamed because they have pursued these tasks, but that they have wearied boys with them"*

[Pratt, 1987b].

So Leibnitz believed that logic as the systemisation of reasoning can and should be mastered, and thereafter mechanised. He advanced the idea of syllogism inasmuch as he sought to apply it to the whole of life rather than to just shapes and lines like Euclid did. He also placed great emphasis on representation or notation and developed a representative scheme for *concepts.* He advanced a *universal polygraphy,* wherein a complex representation of an object was represented as a list of its simple attributes, used in modern-day list processing languages such as LISP.

Leibnitz built a calculator, but not for human reasoning *per se.* Sadly, towards the end of his life, Leibnitz' work sank into obscurity and his works left unread until the renaissance of mathematics in the nineteenth century. The denouement of Leibnitz' ideas on computation found a prologue in Charles Babbage.

The rise of mathematics in the nineteenth century, especially the advances made in extending algebra spurned the interests of Charles Babbage [2] in mechanisation. Babbage along with George Peacock [3] and others toiled at the notion of algebra for calculation leading to the axiomatisation of Algebra.

Algebra in this new form was conceived of as "*the science of general reasoning by symbolic language*" according to Peacock [Pratt, 1987c].

Owing to his design of the analytical and difference engines, Babbage is sometimes referred to as the *father of computing.* By their conceptions and designs, both machines were certainly marvels of engineering.

[2]Charles Babbage [1792-1871] Self-taught mathematician and inventor.He inherited money from his banker father which he used to finance his inventions. Among his inventions were the first speedometer, skeleton keys and the first actuarial tables. He also advocated the division of labour in factories the effectiveness of which Ford showed in his US car factory. His criticisms of the British method of postal charges led to the introduction of stamps with the Penny Black in 1840.

[3]George Peacock [1791-1858] English mathematician, son of a Curate. He was educated at home thereafter entering and graduating from Cambridge University. He published a text on Algebra in 1830.

Babbage and Ada, Lady Lovelace [4] believed that the ability to manipulate algebraic symbols would allow the Analytical Engine to use a *universal language* that could represent the laws governing the relationships between any two entities. In a similar vein, Ada stated that the Analytical Engine should be thus be able to compose music if only

> " ... the fundamental relations of pitched sounds in the science of harmony and musical composition"

were expressible in the universal language she envisaged,

> *" the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent"*

[Pratt, 1987c]

Countess Lovelace considered the Analytical Engine's ability for music, but concluded that original thought could not be imitated. The Analytical Engine was never built, yet the ideas and initial designs of Babbage and Lovelace were revolutionary.

After Babbage, mathematics continued to advance and eventually, in 1936, Alan Turing [5] and others saw what Babbage and Lovelace had done before: that arithmetic was not the only function of machines.

When in the 1930's, Turing and others entertained the idea of building brains, the developments in mathematical logic were of paramount importance. Turing's work in discrete state machines led him to conceive of the brain as such a machine performing the function of controlling behaviour. Ten years later, he proposed to exploit this discovery and his observations of living forms in the building of a brain with his famous 1950 paper *Can Machines Think?* and initiated the *Brain Project.* There, Turing entertained the idea that every human thought could be expressed by his universal machine if it was suitably programmed. He also proposed what became the *Turing Test* of intelligence, a method to determine whether a machine exhibits intelligence or not. To date, no-one has developed a computer to pass the Turing Test and it is still an active area of research.

[4] Ada Augusta, Lady Lovelace [1815-1852]. Daughter of Lord Byron. Her descriptions of the Analytical Engine preserved knowledge of it for posterity. A lady of brilliant intellect.

[5] Alan Mathison Turing [1912-1954]. British Mathematician. Showed an exemplary aptitude for mathematics during his early 20's. He proved the central limit theorem without knowing that it had been proved already. Worked in Numerical Analysis and noted for his major contributions to the early development of computing with his Turing Machines and notions of machine intelligences. Was concerned with the mechanistic interpretation of the natural world and during his later life, he attempted to determine a chemical basis for organic growth. Also noted for his version of chess : round the house chess.

3.2 Knowledge

Knowledge is fundamental to AI systems that are to exhibit any sort of intelligence. Early researchers believed that the approach was to develop general problem solvers (Newell and Simons' General Problem Solver) for particular domain classes. However, these systems almost invariably required considerable hand tailoring of the problem descriptions and an *ad hoc* guidance to solutions.

Intelligence requires the possession of and the access to knowledge : an important characteristic of intelligent people is that they possess much knowledge. This realisation heralded the prologue of domain-specific knowledge in computer programs, which are now termed expert systems.

Knowledge may be defined as the collection of facts and principles accumulated by mankind via acts of personal experience and 'knowing' (called *noumena*), or as having a familiarity with places, customs and entities coupled with an ability to utilise these notions effectively in different situations. Without this organisational ability, the facts and rules associated with experience are meaningless.

In biological organisms, knowledge may be stored as complex structures of interconnected weighted neurons. (Biological neurons are said to provide 10^{14} bits of storage). This representation is termed *non-symbolic. Symbolic* representations are articulated knowledge stored in some readable form.

Knowledge may be classified as either *procedural* or *declarative.* Procedural knowledge is knowledge compiled relative to the performance of some task (eg. steps to solve an algebraic problem). Declarative knowledge is passive knowledge expressed as statements of facts about the world. (eg. personnel data in a database).

Heuristic knowledge, which may be expressed both procedurally and declaratively, is commonly utilised by humans for problem solving. Knowledge about a problem is used to simplify it, to make good judgements and undertake appropriate actions. Although not always correct, it leads to quick solutions.

The difference between *knowledge* and *data* is that data concerns elements *per se*, whereas knowledge is the data with relations between them (i.e. knowledge is structured data). The relations may be either rules or symbolic links. For example, when a doctor treats a patient, he uses the patients' record (the data) and the facts, beliefs and heuristics (the knowledge) he learnt during his training and experience to determine a diagnosis.

To define knowledge, certain important concepts need to be introduced. A *Belief* is any meaningful or coherent statement that may be represented. *Hypotheses* are justified beliefs that are not known to be true. *Knowledge* may be defined as a collection of true justified beliefs, a collection of true hypotheses.

Epistemology is the study of the nature of knowledge. *Meta- knowledge* is knowledge about knowledge, ie. knowledge about what we know.

3.3 Expert Systems

The interests of mechanising logic should not be considered to be an esoteric concern of mathematicians and logicians. The appeal thereof is the ability to draw automatic conclusions from a collection of facts. This led to the notion of expert systems, computer programs that mimic the logical reasoning that a domain expert indulges in to solve a particular problem. They may also be considered to be theorem provers in that a proposition (goal) follows from any combination of propositions (facts and rules) that the system has been given. These programs marked a departure from algorithms and general search methods, such as hill climbing and means-end analysis (GPS) by using specialised domain knowledge and heuristics. Edward Feigenbaum, creator of the first expert systems, stated that the power of the expert system is in the *knowledge* it possesses, rather than the inference mechanism [Patterson, 1989].

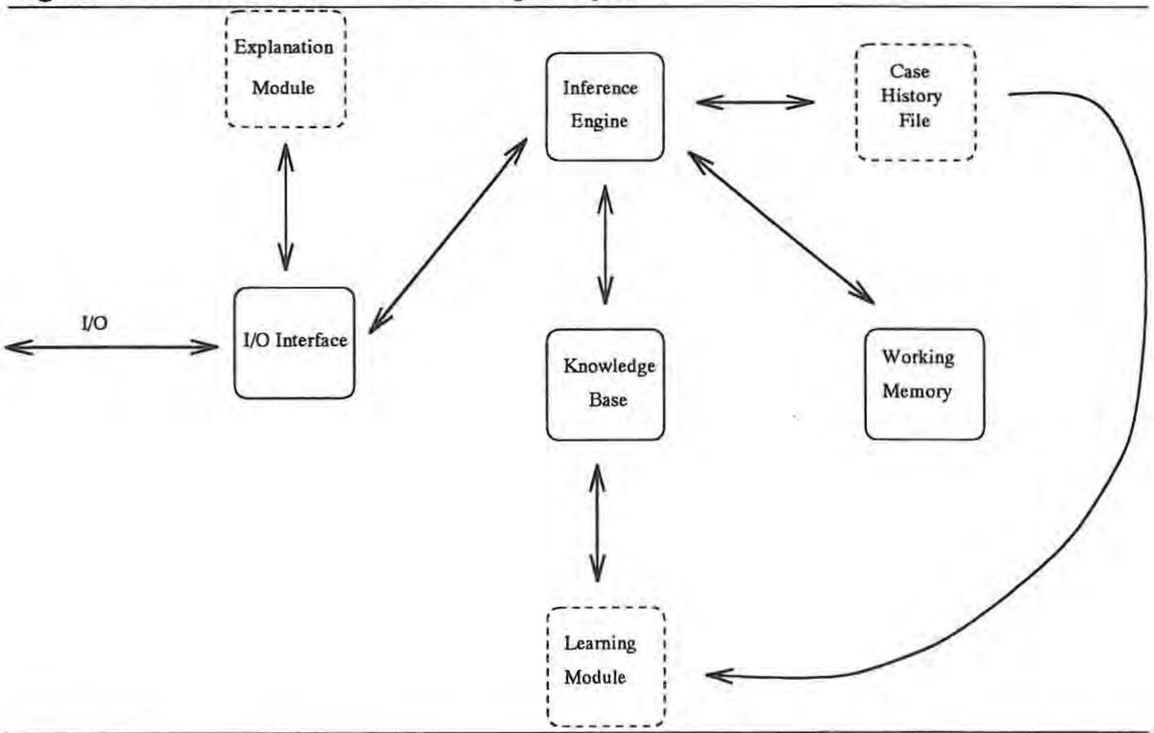
The first such program was DENDRAL developed by Lederberg, Feigenbaum and Djerassi of Stanford University in 1965. Dendral determined the molecular structure compounds from their constituent elements and mass spectral data. It was followed by META-DENDRAL, essentially DENDRAL with a learning function. This was done owing to the difficulty of knowledge acquisition. In 1968, MACSYMA was developed, solving a variety of mathematical problems. The next major expert system, and possibly the best-known, was MYCIN, also developed at Stanford. It diagnosed infectious blood diseases and prescribed a list of remedies. MYCIN's accuracy as opposed to that of a human expert was significant: 65% as opposed to 60% . MYCIN spawned the AI programs THEIRESIUS, a knowledge acquisition system, GUIDON, a tutorial system, and EMYCIN, the first expert system shell. After these initial systems, expert systems started to mushroom in size, complexity, number and in topology.

3.4 Representation

The representation of knowledge is of cardinal importance in AI. It may be via written text, character strings, binary numbers, or whatever. Certain representations are more suitable for particular applications than others. For example, for a poker game, cards may be represented as a simple character string : c6 for a six of clubs.

Expert systems are often characterised according to the representation of their knowledge, their *architecture*. The architecture of an expert system refers to the organisation of the knowledge and the subsequent methods of accessing it. Expert system architectures may be characterised as either *Production* (Rule-Based) or *Non-Production* Systems. Non-production systems include systems that are frame-based, associative (semantic) networks, decision trees, blackboard systems, object-oriented systems and neural networks. Other memory organisations include Scripts, Plans, Goals and Memory Organisation Packets (MOP's), particular to case-based reasoners. Production systems and neural networks [Gallant, 1988] are termed *unstructured* systems whereas, the other organisations which are essentially graph-like structures, are *structured.* The general layout of an expert system is depicted in figure 1 (dotted boxes indicate those modules not found in all expert systems).

Figure 1 A Schematic of a General Expert System



As may be deduced from the figure above, these systems comprise a *knowledge base*, an *inference engine*, a *learning module*, a *case history file*, *working memory*, an *I/O interface* and an *explanation module*. The general execution cycle proceeds as follows. Input to the expert system, a human query or a result from a system being monitored or controlled, is entered via the I/O interface and is then placed into working memory. The inference engine then uses the input to chain through the static information contained in the knowledge-base rules to reach an intermediate solution. During chaining, intermediate results are generated and stored in the working memory. These results, commonly called dynamic information, are then utilised to further the inferencing process performed by the inference engine. The final results are stored in the case history file for use by the learning module which adds information to the knowledge base. The explanation module attempts to explain how a conclusion was reached or why certain information is needed.

This chapter emphasises the process of acquiring knowledge, building a knowledge base and the problems associated therewith. The interested reader is referred to any of the many books on AI and expert systems, for instance [Patterson, 1989].

3.4.1 Rule-Based/Production Systems

Originally proposed as a model of human information processing, the production systems paradigm now occupies a prominent place in AI, having been used in expert systems and in human intelligence modelling. The first expert systems, viz. DENDRAL and MYCIN were all Rule-based. MYCIN initially possessed 200 rules, which grew to 600 by the early 1980's. These expert systems comprise a knowledge base of rules and facts. Rules are simple IF-THEN statements of the form

IF <Antecedents> THEN <Consequents>

The antecedents and consequents are, respectively, also termed conditions and conclusions. When there are more than one of each, they are separated by the logical connectives AND OR. For example
IF (lips are pink) AND (head is sore) OR (feels tired)
THEN (Carbon-Monoxide poisoning)
states that if a person has pink lips, a headache or feels tired, then carbon-monoxide poisoning is present.

Facts are observations that are assumed to be true or irrefutable. They may be represented in a variety of ways, but the most common method is to use First Order Predicate Logic (FOPL), for example PROLOG. FOPL is popular owing to sound mathematical theory, suitable expressive power and valid forms of inference. Facts may be represented easily as, for instance

Father(Paul, Debra) = Paul is the father of Debra

and the rules may be represented as Prolog conditionals. However, a problem common to FOPL is an inability to represent commonsense knowledge, which almost all people possess and is essential for any intelligent system.

Inference proceeds by eliciting an initial input from the user and then to use this dynamic information to chain through the static information contained in the knowledge-base rules to draw conclusions. This proceeds as follows.

Match : The inference engine matches the contents of the working memory with the knowledge base rules.

Select : If a match is found, the rules are added to the conflict set which resides in the working memory.

Execute : One rule is selected for execution from the conflict set. The system may ask for more information, fire the rule and thus place the new data into working memory or stop. By "firing a rule", the data in the consequent part is added to the working memory. When the last rule in the knowledge base has been matched, the data in working memory to be matched is removed ant the next datum is used.

The inference process may proceed as *forward* or *backward* chaining. In *forward chaining*, the left hand side of the rules (the antecedents/conditions) are instantiated first, i.e. the system is data-driven with the rule's right-hand side forming the sub-goals. For *backward chaining*, the rule's right-hand side is instantiated first, i.e. the system is goal-driven, an initial goal initiates a backward direction of chaining. Here, the antecedents/conditions form the sub-goals. Backward chaining is utilised when *what-if* questions are posed.

3.4.2 Non-Production System Organisations

Less common than production systems, non-Production Systems generally exhibit more structured knowledge bases. Included are associative/ semantic networks, frames, decision trees, blackboard systems, object-oriented systems and neural networks. It suffices to say that there has been a renewed interest in semantic networks (especially for parallel architectures) owing to the speed with which elements may be searched for and updated. Also, the interest in neural networks has surged over the last decade or so with a plethora of architectures being put forward and tested in various domains.

3.5 Uncertainty

Intelligent beings are continually required to make decisions under a veil of vague, incomplete, imprecise, contradictory and/or continually varying information. These concepts are collectively termed *uncertainty*.

With an aim of building intelligent systems, it is essential for computer systems to reason with uncertain information. Many AI systems use *monotonic* [6] logics to represent knowledge, thus assuming that all conclusions are valid and all information is given. This is clearly inadequate for the manipulation and representation of uncertainty. Uncertainty in expert systems is represented using probability, possibility and certainty theories and *non-monotonic* logics such as modal and fuzzy logics. Modal logics are extensions of classical logic allowing concepts such as "likely" and "possible" to occur. They were developed to better represent commonsense reasoning. Fuzzy logic, also an extension of classical logic, allows varying degrees of the presence of concepts, such as height and wind. Thus, the partial truth of concepts is allowed to facilitate the handling of uncertainty. Fuzzy logic and the other representational methods of uncertainty are discussed in chapter 4 on fuzzy logic.

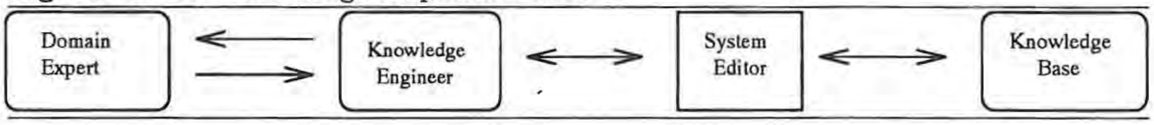
3.6 Knowledge Acquisition and Validation

Knowledge acquisition and validation are the most difficult and time-consuming aspects of designing an expert system. For acquisition, an expert is interviewed during which he is asked to solve typical problems specific to his field and to explain his conclusions. He has also to *re-state* his knowledge for the interviewers' understanding, whereafter the interviewer has to codify this re-stated knowledge into a representation suitable for a knowledge base. These demands of knowledge acquisition have created a new profession

[6] The difference between *Monotonic* and *Non-Monotonic* logics is as follows. In monotonic (traditional) logics, the addition of new knowledge increases the size of the knowledge base or axiom set linearly. For non-monotonic logics, this is not the case and the incorporation of new knowledge is easier.

for the interviewer: the *knowledge engineer*. Figure 3.1 below illustrates the knowledge acquisition process.

Figure 3.1 The Knowledge Acquisition Process



The knowledge gained from the domain expert is also used to test and validate the knowledge in the knowledge base. This acquisition-validation process is repeated until satisfactory results are obtained. This process takes the better part of a year and may consume tens of man years [Patterson, 1989]. [Patterson, 1989] states that

> " *Experience in building dozens of expert systems and other knowledge-based systems over the past fifteen years has shown this to be the single most costly and time-consuming part of the building process*".

Another quote from [Schank, 1991] concerning his development of an AI system called SAM, that evolved into FRUMP and then ATRANS, that was designed to summarise, paraphrase, translate and answer questions about stories, underscores this aspect.

> "... *there is one important fact to know about ATRANS. It took something like 30 person-years to make it work. This number is in addition to any of the SAM-FRUMP work. There is an important lesson to be learnt here*".

And that

> "*The lesson to be learnt from ATRANS is simple enough. AI entails massive software engineering. To paraphrase Thomas Edison, "AI is 1 percent inspiration and 99-percent perspiration". AI people will never build any real AI unless they are willing to make the tremendously complex effort that is involved in making sophisticated software work*".

[Schank, 1991] further states that a real test for AI systems is whether they easily scale up from toy problems to complex real world ones.

The aforementioned vagaries associated with the construction of a knowledge base are collectively termed the *Knowledge Acquisition Bottleneck*. Recognition of this problem has prompted researchers to propose specific knowledge base building tools, machine learning, the extraction of knowledge from databases and the construction of CYC, a large knowledge base of 10^8 commonsense facts [Microelectronics & Corporation, 1991].

3.7 Knowledge Management

Knowledge management concerns access to knowledge and is the key to the efficient processing thereof. For large systems, such as XCON, DEC's 12 000 rule expert system designed for the configuration of it's computer systems, it is essential that appropriate structures are located, accessed and retrieved quickly. For these systems, exhaustive searches are implausible and, consequentially, recourse to RETE-like matching procedures [Forgy, 1982], indexing and hashing methods or the use of structured (graph-like) representations for knowledge is made. The RETE method is an efficient method for processing production systems whereas the latter two methods simplify the search by performing localised searches. Since large knowledge bases often reside on secondary storage, another method is to cache storage device.

Allied to these methods are certain problems. The *Frame Problem* arises in systems that function in dynamic environments. It concerns knowing what changes have and have not occurred after the occurrence of some influential phenomenon, i.e. how frames and the relations between them are to be modified. This type of problem must also be considered by all non-production representational systems.

Furthermore, since knowledge changes periodically, the ability of the system to incorporate this new knowledge is desirable. Human memories are dynamic in that they have an inherent ability to learn and consequentially, concepts from human memories are used in designing AI systems. Learning is thus a quintessential aspect of intelligence - someone who makes the same mistake twice or more times is termed "dumb" or "stupid". That learning may be easily incorporated into MBR is another of it's advantages.

3.8 Conclusion

As has already been mentioned, the construction of knowledge bases is no trivial matter indicating the largesse of the *knowledge acquisition bottleneck.* This problem is apparent for both production and non-production systems. Allied to frame-based systems was the frame problem. Also, learning is a difficult task, especially for real (in Schanks' words) AI systems. The problems of generating appropriate rules and establishing their consistency *vis-a-vis* the other rules in the rule-base is extremely difficult and computationally expensive. A similar problem is apparent in non-production systems (cf. the Frame problem). These problems are a strong motivation for the development of *Memory-Based Reason-*

ing, of which *Cogitator* is an example. But before proceeding to memory-based reasoning, some other precedents need to be considered. The following chapter briefly illustrates the field of concurrent AI, in particular, the various types of concurrent expert systems. Theproblems associated with each type of concurrent expert system are highlighted with the aim of later proposing a system to overcome them.

Chapter 4

Concurrent AI

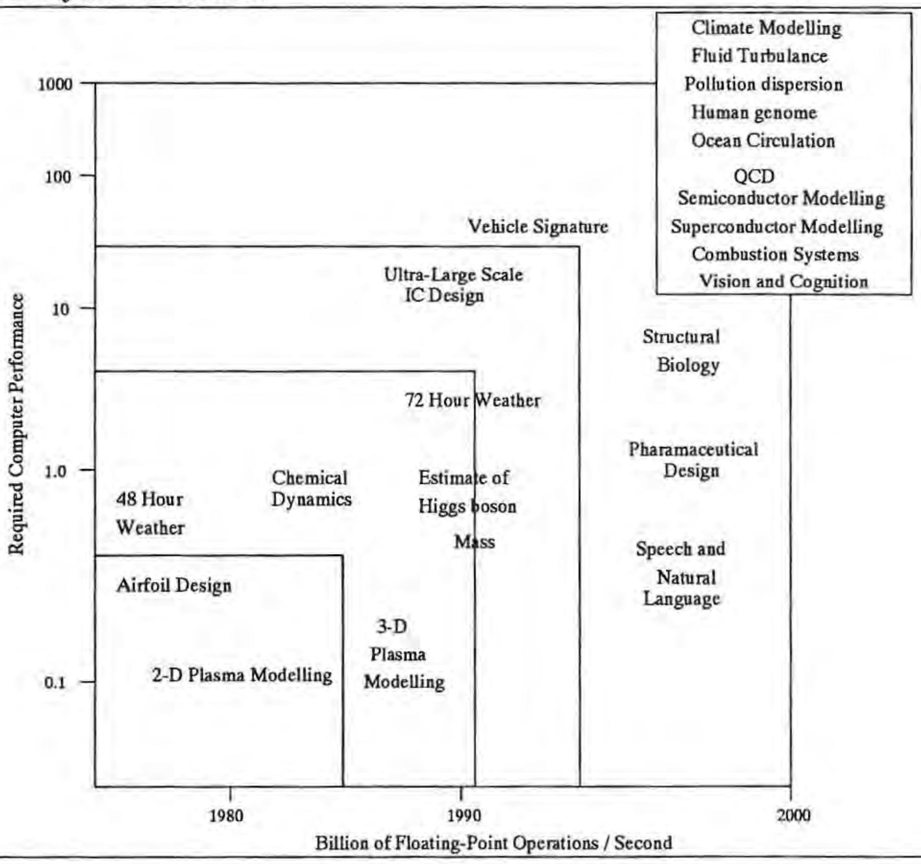
The recent advances in concurrency have influenced almost all aspects of computer science and electrical engineering: new microchips, computer architectures, programming languages, programming paradigms, algorithms, and so on have been designed and built. What effect has this exerted on AI? AI is acknowledged as being difficult: problems in very restricted environments work well, but when the scope is increased, they tend to fail. Also, AI has not lived up to the expectations and promises of it's early apologists and, AI systems are becoming more complex. With concurrency unanimously acknowledged as being difficult and complex (one researcher wryly remarked "Death, taxes and parallelism, no-one likes them, but they are here to stay"), why compound matters further by forming a confluence with AI? Consider the following.

The first chapter on concurrency mentioned the various *Grand Challenges* that were originally posed almost 10 years ago by the Nobel physicist, Kenneth Wilson, and extended by others since then. For the readers' convenience, the grand challenges are reproduced below in figure 4.1.

The reader possibly noticed the presence of speech and natural language processing as well as computer vision and cognition, major fields of AI. They form a cardinal reason for the development of *Cogitator*.

[Walz, 1990] provides a summary of parallel AI systems implemented on the Connection Machine and argues for massive parallelism in AI systems. He proceeds to state that truly intelligent machines would have to attain processing power and memory of at least four orders of magnitude greater than that which is currently available on the most powerful current machines. This sentiment is echoed in [Desbiens & Nault, 1992]. To elucidate this, consider the characteristics of a bee's brain [Sejnowski, 1992].

Figure 4.1 The various Grand Challenges, their estimated processing requirements and the expected year of solution.



The energy dissipation of a honeybee's brain is of the order of 10^{-6} watts (10 microwatts), about seven orders of magnitude better than current microchips. Furthermore, the operating speed is conservatively estimated to be approximately ten Teraflops (10^{13} operations), about three orders of magnitude better than today's fastest computers that (theoretically) peak at about ten Gigaflops. Also the size of a bee's brain is a few cubic millimetres, many orders of magnitude smaller than a supercomputer. Besides these capabilities, the bee can harvest nectar and return to the hive with it, recognise and remember high nectar concentration sites thus maximising foraging benefits, see, smell, fly, walk and maintain balance, they also navigate over long distances and predict changes in nectar location, recognise intruders and attack, recognise dead bees and remove them from the hive and when the hive becomes crowded, they fly off in a swarm in search of a new home. All these tasks they perform autonomously, ever heard of an autonomous supercomputer? Current robots and androids may be described as clumsy at best when compared to even the simple actions of humans and animals and the speeds that they are performed at.

Likewise, expert systems are still way behind in mimicking the type of everyday reasoning that humans perform. This is echoed in [Moldovan & Chung, 1992].

[Douglas & Mahowald, 1992] mention certain important performance characteristics of the human brain. The human brain contains 100 billion neurons (cf. the 1 million of a bee's brain) operating in the millisecond rather than the nanosecond range of their electronic counterparts. The speed of the brain is approximately 10^{16} operations per second and the power dissipation is approximately 10^{-15} joules per operation compared to 10^{-7} joules for electronic chips. [Douglas & Mahowald, 1992] proceed to mention the unsolved object perception problem and that the neuronal computations to facilitate object recognition are less than a hundred operations deep. This they attribute to the vast inter- neuronal connection network.

[Walz, 1990] also mentions a consensus in the supercomputing community that the supercomputers of 2000 will all be massively parallel. This view is echoed in [Zorpette, 1992b]

> *"On the verge of a gradual takeover, industry analysts believe, is the massively parallel processor (MPP), which is already winning for itself a place as the high performance architecture of choice".*

Even Cray Research, the bastion of vector processing is even moving towards MPP's.

Nevertheless, nearly all current AI systems have been designed to operate on single workstations or PC's. Considering this, there Are massively parallel machines or networks of cooperating machines necessary? Won't very powerful serial machines suffice? The performance limitations of serial computers is limited by the classical *von Neumann Bottleneck* (mentioned previously) as well as the rapid approach of the physical limits of VLSI. [Walz, 1990] offers a prognosis of future performance- enhancing technologies such as Gallium Arsenide, the Quantum flux parametron and the Josephson junction (Gallium Arsenide currently provides a speedup factor of three), clever cacheing and instruction prefetch techniques, the use of multiple functional units, new compiler technologies, higher clock rates facilitated by better cooling technologies and advances in semiconductor design (DEC's new 200 MHz *Alpha* chip, although cooling the chip is a major problem [IEEE, 1992]). Also, highly dense chips of exotic materials packed close together present difficult cooling and packaging problems) and further miniaturisation (VLSI and SLSI technology is generally acknowledged as nearing their limits concerning miniaturisation and switching speeds).

[Walz, 1990] thus concludes that serial technologies offer limited speedups. This sentiment is also echoed by [Desbiens & Nault, 1992] and [Herath, 1992] who states that

> *"Conventional uniprocessors cannot efficiently handle intensive computations with a high level of data dependency. Such processors are good at executing sequential computations that have been precisely formulated, but have difficulty processing tasks related to vision, speech and spacial and temporal pattern recognition in the presence of noise. Intelligent computations with irregular properties are suited for implementation on MIMD machines. Linear recursive structures are suited for implementation on SIMD machines".*

As the computing power of systems increases, more complex algorithms often need to be implemented to increase a system's intelligence. Generally, knowledge bases (KB's) of the size of 10^2 to 10^4 axioms (rules and facts) represent enough knowledge to be proficient in a certain restricted domain. Knowledge bases that are to contain enough information so as to be non-brittle and more widely applicable are to be four to five orders of magnitude bigger than those currently available [Lenat & Sheperd, 1990]. During the 80's, work began in Very Large Knowledge Bases (VLKB's) such as CYC [Lenat & Sheperd, 1990] and databases such as Japan's Electronic Dictionary Research (EDR) project, part of the Fifth Generation Computer Systems Project. Also, in the light of arguments that databases contain information.

[Walz, 1990] also advocates that massively parallel processors (MPP's) provide the best alternative concerning performance and cost. However, not many people nor organisations currently own or can afford MPP's, nor are they likely to. Rather, networked workstations and PC's are considerably more widespread and functional. Consequentially designing systems to execute on these networked machines is a more viable alternative, another motivation for the development of *Cogitator*.

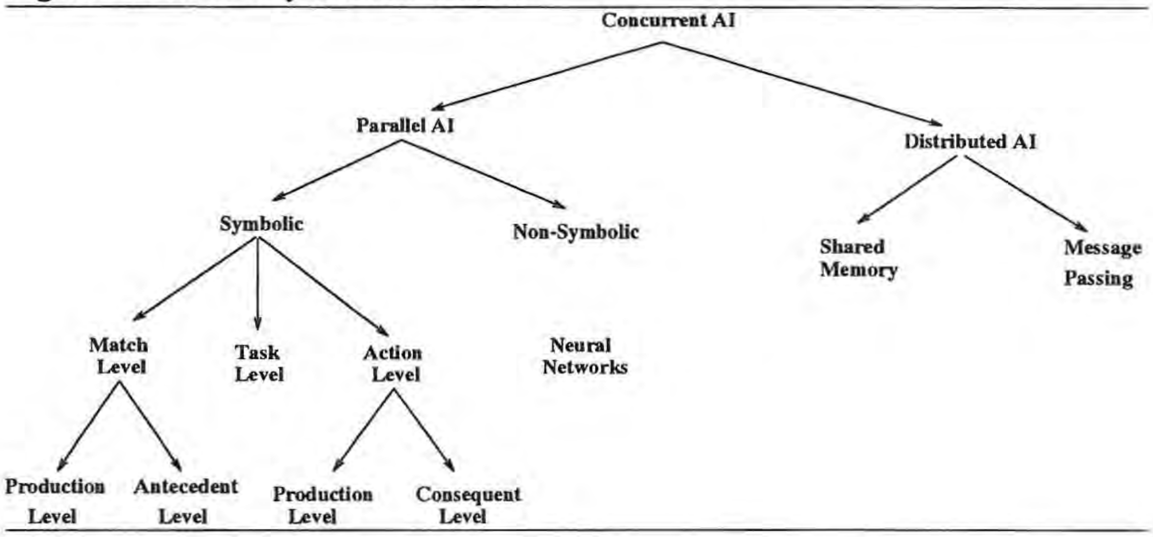
The Japanese were the first to recognise the benefits of merging concurrency and AI, quintessential to the fifth generation computer systems (FGCS) project. Initiated in 1982, the FGCS project concentrated on the development of concurrent inferencing techniques, computers (Parallel Inference Machines - PIM's) and logic languages. The FGCS project was terminated in June, 1992 shortly after it's successor, the Real World Computing (RWC) project was announced. The computing hardware of the FGCS project will form part of the computing base of the RWC project. For a brief summary on the aims, results and success of the FGCS project and a discussion of the RWC project, see [Baise, 1993].

In the light of the aforementioned revelations, that AI systems are space and time hogs, and that AI systems cannot handle suitable problems in real time (which may be attributed to a lack of computing power [Moldovan & Chung, 1992]), it is imperative that for AI systems of any decent nature to be constructed, efficient computational capabilities far beyond the capabilities of current systems and paradigms are needed. An attempt to solve these problems has led to the development of concurrent expert systems.

4.1 Concurrent AI Systems

Within concurrent AI, numerous paradigms may be identified. These are summarised in figure 4.2 overleaf and are briefly discussed in the ensuing pages, starting with parallel AI.

Figure 4.2 Taxonomy of Concurrent AI



4.1.1 Parallel AI

Judging from the dates of papers cited in the papers describing concurrent expert systems, the first such systems arose during the late 70's and early 80's with the Hearsay speech recognition system [Erman & Raj Reddy, 1980]. Work began in earnest during the mid to late 80's [Walz, 1990]. Currently, the field is still in it's nascent stages, but it is expanding.

Most work in parallel AI concerns the parallelisation of production (rule-based) expert systems, although work is being performed in parallel AI languages. Some work has also been performed in parallel algorithms.

Production systems are computationally very expensive and therefore slow, limiting their acceptance in industry [Kuo & Moldovan, 1992]. Furthermore, the need for intelligent systems, continued research and development in production systems is expected to lead to larger and more complex systems, exacerbating the performance problem. Therefore utilising the benefits offered by parallelism is essential.

Within parallel production systems exist three paradigms: actual parallel production systems, parallel production programming envirnoments and parallel production programming languages. Examples of each will be discussed in turn.

Faster Sequential Production Systems

Attempts to speed up sequential expert systems have led researchers to concentrate on speeding up the RETE algorithm [Forgy, 1982], the *de facto* matching algorithm for expert systems. Two examples of such expert systems are TREAT [Miranker, 1987] and YES/RETE [Highland & Iwaskiw, 1989]. The important aspects thereof are summarised in figure 4.3 below.

Figure 4.3 Sequential Match Algorithms. (Adapted from : [Kuo & Moldovan, 1992])

Name	Type	Speedup	Problems	Comments
TREAT	Sequential	4-15	Sequential (Limited Speedup)	Speeds up RETE significantly
YES/RETE	Sequential	Around 10	Sequential (Limited Speedup)	Speeds up RETE significantly

The realisation that sequential algorithms only lead to limited speedups has led to the development of parallel matching algorithms and systems.

Parallel Production Systems

Parallelism has been introduced to production systems at three levels: the *match*, *action* and *task levels*.

Match level parallelism arises when parallelism is introduced to speed up the matching of entities in production systems. Matching may be introduced at either of two levels: the *production* or *antecedent* level. At the production level, a single antecedent of *each* of the productions is simultaneously matched working memory elements, whereas, at the antecedent (condition) level, the antecedents (conditions) of a *single* production rule are simultaneously matched with the working memory.

At the *action level*, parallelism is introduced to speed up the execution of the actions (rule firing) of a production system. This may be accomplished by either firing all the actions of a single rule simultaneously, or by simultaneously firing a single action of each of the production rules. These two methods are referred to as, respectively, *production* and *action* level parallelism.

At the *Task level*, several homogeneous or heterogeneous tasks are performed simultaneously.

Of these levels of parallelism, match level parallelism is the most prevalent. Since production systems spend much of their time matching (around 90% [Forgy, 1982]), it was initially believed that match level parallelism should lead to vast speedups. However, it has since been shown that this is not the case and is attributed to three main reasons :

(a) Only a few productions are affected by changes to working memory (WM). Sequential algorithms such as RETE, may via appropriate bookkeeping, avoid unnecessary processing.

(b) Some productions are very expensive to match, while others are cheap. Therefore, some processors sit idle waiting for others to complete their tasks (the *small cycle problem*).

(c) The overhead arising from inter-processor communications.

Parallel Matching Algorithms and Systems

Changes to working memory only affects a few rule network nodes. Also, the number of node activations per change is independent of the number of rules in the rulebase. Therefore, neither small nor large production systems can be expected to execute quicker. Also, since the node changes are small, fine grained parallelism is appropriate.

Parallel match algorithms exploit this fine-grained parallelism by partitioning the RETE network and assigning each segment to a processor (node-level parallelism) and/or by breaking up the memory nodes themselves and assigning each to a different processor (intra-node level parallelism). The major issue is to partition the RETE network efficiently. Parallel Match algorithms have been proposed for shared memory, message passing and special purpose architectures. Examples of each are summarised in figure 4.4 below.

One of the main reasons for the very sub-linear (low) speedups is what is termed the small-cycle problem, that there is simply not enough work to occupy all the processors most or all of the time (the parallelism is limited - changes to WM are small). This is

Figure 4.4 Parallel Matching Algorithms.

Name	Type	Speedup	Problems	Comments
PSM-E	Shared Memory	2-11 (13 Processors)	Cross Product not reduced Resources and memory must be locked	Only Shared Memory System
PSM-M	Message Passing	8-12 (Simulated)	Cross Product not reduced Single control processor is limiting	Variant of PSM-E
DADO	Message Passing	2-31 (1023 Processors)	Under-utilisation of processors Specific Architecture must be locked	Pipelined Processors May operate in SIMD or MIMD mode
PESA-1	Message Passing	12.5 (32 Processors)	Under-utilisation of processors Specific Architecture must be locked	Dataflow Machine Distributed Memory and control Similar to
DRETE	Message Passing	9.3 (16 Processors)	Cross-product effect not reduced	DADO ans PESA-1
			Specific Architecture Overhead from a dynamic load balancer	Memory and control

also common in sequential systems in that only one rule may fire at a time. The small cycle problem may be addressed by incorporating the parallel execution (firing) of the production rules.

Multiple Rule-Firing Systems

Multiple Rule Firing Systems (MRFS's) attempt to increase the amount of available parallelism by parallelising *all* phases of the execution cycle, not just matching. By parallelising the act phase, the small cycle problem is alleviated, and by executing inference phases simultaneously, the variance of processing time is reduced.

Nevertheless, two significant problems, the compatibility and convergence problems arise when attempts are made to eliminate the sequential conflict resolution which prevents multiple rule firing.

The *compatibility* problem arises when rule instantiations are not independent. A set of rule instantiations may simultaneously fire only if they do not interfere with each other; ie. firing a rule in one set does not preclude any other instantiated rules from firing.

Rules may simultaneously fire if there are no inter-instantiation data dependencies. A set of non-interfering rule instantiations is termed a *compatible set*. Firing a compatible set simultaneously yields the same result as if they were sequentially fired. This is not the case for incompatible rule instantiations so they cannot fire simultaneously. The compatibility problem is thus concerned with determining compatible rule sets.

Compatible rule sets are determined by constructing and then analysing data dependence graphs (cf. data dependence analysis in parallelising compilers). This data dependence graph is constructed by analysing the inter-rule instantiation data dependencies and then combining the result into a directed graph.

Compile time analyses of the rule compatibilities are possible by analysing the left hand side of the rules since they determine the instantiations to be matched and the right hand sides represent the effects of firing these instantiations have on other rule instantiations. Therefore, analysing the rules using the cyclic or pairwise conditions yields a data dependence graph.

As is usual with compile time analyses, information is obtained off-line, the disadvantage is that only limited information is available. This disadvantage is remedied by utilising run time analyses.

Run time analyses are more complete since all variables are bound at run time and it is therefore possible to analyse the data dependencies between rule instantiations instead of just between rules. However, this process introduces run time overhead and consequentially, certain researchers have proposed a mixture of compile and run-time analyses [Kuo & Moldovan, 1992]. Compile time analyses offer limited information and run-time analyses increase the computational overhead.

As compile and run-time analyses are complementary, both are utilised to reduce the processing time : the run-time analyser has some initial information from the compile-time analyser to work from.

As the compatibility condition presents a local view of the problem solving process (the rule instantiations that seem to be right at one stage may be incorrect later on), it is insufficient to guarantee the correctness of the final solution. This is termed the *convergence condition.*

Two methods for addressing this problem include non-deterministic execution and parallel conflict resolution. In non-deterministic execution systems, rules are chosen arbitrarily and fired without the help of conflict resolution.

By executing randomly chosen rule instantiations, many different execution sequences arise and consequentially, many different solutions are possible, albeit not all correct.

Correctness in non-deterministic production systems is guaranteed if it may be proved that, starting from an initial state INIT, all possible execution sequences satisfy a post condition POST.

Parallel conflict resolution strategies may be encoded as a collection of meta-rules, the inputs to which are the matched rule instantiations [Kuo & Moldovan, 1992]. By carefully coding the meta-rules, a non-interfering set of rule instantiations may be selected and fired.

Various MRFS's have been proposed and some implemented as well, including parallel production languages (CREL [Miranker & Browne, 1990], SWARM [Cunningham & Roman, 1990] and PARULEL [Stolfo & Ohsie, 1991]) parallel programming envirnoments (IRIS [Pasik, 1989] and Ishida's system [Ishida, 1991]) and parallel production programs (PARS [Schmolze & Goel, 1990] and RUBIC [Moldovan, 1989]). Parallel production languages were proposed to facilitate the non-deterministic execution of production systems. They are summarised in figure 4.5 below.

Other systems have also been implemented on more general purpose machines. [Walz, 1990] briefly describes AI systems that have been implemented on the Connection Machine (r) and are described in detail in their respective papers, including CIS : a marker-passing parallel expert system which has one instantiated rule per processor [Blelloch, 1986] and a massively parallel document retrieval system [Stanfill & Kahle, 1986] which led to the 1989 commercial product DowQuest(r). Natural Language Processing (NLP) and computer vision systems are also mentioned. [Waltz & Stanfill, 1988] also describe the early applications of massively parallel AI applications on the Connection Machine.

Nearly all the systems implemented to date have been implemented on concurrent computers, some special-purpose, others more general. The raison de etre behind these systems is that there is enough parallelism to be utilised by the architectures, in the case of the Connection Machine, SIMD-type computation is deemed to be appropriate.

4.1.2 Non-Production Systems

A central issue in the design of expert systems is the representation of the knowledge base. As was mentioned, rule- based systems are difficult to construct and slow. For these reasons, other knowledge base representations have been proposed and used. Recently, two interesting non-production expert systems have been designed and implemented. Two systems, SNAP [Moldovan & Chung, 1992] and IXM2 [Higuchi & Kokubu, 1991], respec-

Figure 4.5 Multiple Rule Firing Systems.

Name	Type	Speedup	Problems	Comments
CREL	Parallel Production Language Language	5-12	Programmer must write "correct" Programs - difficult for large systems	Convergence and Compatibility problems solved problems solved
SWARM	Parallel Production Language Language	N/A	Programmer must write "correct" Programs - difficult for large systems	Convergence and Compatibility problems solved problems solved
PARULEL	Parallel Production Language Language	2-21	Programmer must write "correct" Programs - difficult for large systems	Convergence and Compatibility problems solved problems solved
IRIS	Parallel Production Environment Language	5.6-90.8 (Simulated)	Programmer must write "correct" Programs - difficult for large systems	Reduces software Complexity and increases parallelism problems solved
Ishida's System	Parallel Production Environment Language	5.11-7.57 (Simulated)	Programmer must write "good" Programs - difficult for large systems	Comprises an analyser, a simulator and language constructs
PARS	Parallel Expert System	N/A	Need to assign appropriate Priorities to Rules	Solves the Compatibility and Convergence Problems
RUBIC	Message Passing	3.84-4.35 (8 Processors)	Specific Architecture Mapping rules to Processors difficult	Implemented on a hypercube Addresses the Compatibility and Convergence Problems

tively, utilise the fast propagative and lookup capabilities of semantic networks and the benefits of associative memory to infer conclusions. They are summarised in figure 4.6 below.

Associative memory, a simple and efficient methodology of attaining efficient massive parallelism, allows association and set intersections to be performed in O(1) time. It also embodies extremely powerful logical and arithmetical computing power in that operands may be stored at each node or word and an operation may operate on all words simultaneously. According to [Higuchi & Kokubu, 1991], this method allows nana-seconds of execution time per datum. Furthermore, parallel write and the search functions of associative memory allow marker propagation from a node to proceed concurrently, independent of the number of fan outs (i.e. O(1)). This powerful feature is termed parallel marker

Figure 4.6 Non-Production Systems.

Name	Type	Speedup	Problems	Comments
SNAP	Semantic Network System	up to 1000	Specific Architecture Slow communications	Designed for Natural Language Processing
IXM2	Associative Memory System Language	Over 1000	Specific Architecture for large systems	64 T800 processors 9 T800's for communication Overcomes communication bottleneck

propagation.

These impressive speedups indicate three important aspects. Firstly that sequential machines (SUN-4/330 and CRAY) are unsuitable for semantic net processing and, secondly, that communication overheads and routing are of vital importance in semantic network as well as other types of processing. The latter is especially true if the fanout is large, although the CM-2 prevailed for small fanouts. Thirdly, and most importantly, AI systems may be reformulated using semantic nets to increase the inherent parallelism and parallel systems to exploit this could easily exhibit impressive speedups.

All the systems discussed thus far were all parallel systems, systems in which all the processors operate as a single entity, each performing their bit of work to collectively attain a certain goal. Distributed AI systems differ in that each entity represents a single system in it's own right. These various systems are often termed agents.

4.2 Distributed AI

Distributed Reasoning Systems (DRS's) are systems are composed of different modules or agents each expected to act as a problem- solving entity in it's own right, and an interconnection network connecting them. DRS's range from loosely- to tightly-coupled systems, respectively, systems with or without a central controller.

As intimated by Herbert Simon, large complex systems tend to naturally break themselves down into relatively independent sub- problems within which communication is frequent and quick, while between which communications are rare and slow. This introduces the notion of decomposability.

Nearly Decomposable Problems are those system that when divided up lead to relatively independent sub-problems. Complete decomposability is analogous to linear separability in that they lead to totally independent sub-problems when divided up. For instance, databases are completely decomposable, a characteristic that is deemed to be of cardinal importance and forms the major thrust of this thesis.

That problems tend to decompose themselves, coupled with the power of concurrency and that expert systems have limited domains of applicability (extending their scope seems doomed to failure), leads to the notion that large problems may be solved by a collection of co-operating entities (expert systems), Distributed Reasoning Systems (DRS's).

DRS's exhibit certain advantages over large monolithic systems:

(a) Greater Modularity - easier to build than non-modular systems

(b) Greater efficiency- easier and quicker to design owing to the greater modularity.

(c) Fast computer architectures - for more speed, there is a tendency towards processors with local memory (multi-computers).

(d) Heterogeneous reasoning - certain knowledge representations are more apposite for certain types of reasoning than others (eg. backward verses forward chaining).

(e) Multiple perspectives - many agents better than one since often many perspectives of a problem are needed to devise it's solution.

(f) Distributed problems - many problems are physically distributed, such as weather systems.

(g) Reliability - one agent may die without the whole system crashing.

These architectures must provide the following :

(a) A mechanism ensuring that the collective activities of the various agents solve the problem (ie. the co-operation and co- ordination of agent's activities),

(b) inter-agent communication links, and

(c) since operate on different knowledge bases, rather than a global one, distributed versions of reasoning techniques are necessary.

These are briefly discussed in turn below.

Coordination and cooperation of the processors in the DRS is of cardinal importance and may assume any one of the following four forms :

(a) Master-Slave, wherein one processor assumes the role of master controller, partitioning problems and designating the sub- problems to the agents or slaves,

(b) Pseudo Master-Slave, wherein the master controller partitions problems as before, but it negotiates with the agents as to who will receive what sub-problem,

(c) Cooperative Non-Master-Slave wherein there is no master controller and all the agents cooperate to attain a certain goal, and

(d) Non-Cooperative Non-Master-Slave, wherein a master controller is absent and agents do not cooperate so that the goal is not guaranteed. The agents may even compete with each other.

Although all these organisations differ considerably, the necessity for models of the agents is immanent. The master controller often needs a model of the agents to ascertain their capabilities for certain types of work, their goals and their status (busy or not). In non-master-slave systems, the agents often utilise models of themselves as well as other agents in the system for the purposes mentioned above so that it knows when to give and request help. State-based models, wherein each action by an agent alters either the global or local model state, are useful.

Master-Slave systems represent the least distributed form of reasoning, called multi-agent planning. Unless all the slave agents are homogenous, access to the models of the slaves by the master is necessary to facilitate the efficient allocation of tasks to them. Even if the slaves are identical, the master must perform load balancing to ensure that the goal is attained as soon as possible. Once the tasks have been distributed, unless all the tasks are independent, the slaves must be synchronised. In single agent schemes, static allocation of the tasks is often sufficient, however in multi-agent systems, dynamic schemes are also required owing to the often unpredictable (within limits) nature of the agents.

To illustrate these notions, consider a simple example. Assume documents must be spell-checked before they may be printed. using many spell-checking processors and one printing agent is sufficient. Synchronisation schemes may be simple wherein the master controls all the agents (printing does not proceed until all the spell-checker agents have finished and inform the master accordingly), to more complex systems in which agents cooperate with each other (each lets the print agent know of their job completions).

Pseudo Master-Slave systems (PMSS's) accomplish planning and negotiation by utilising contract nets. In contract nets, there an agent may assume the status of either a manager or a contractor. Managers decompose the problem, look for contractors and monitors them. A contractor accepts a job and completes it either by completing it itself or by partitioning the job further and subcontracting the sub-jobs to other contractors. Negotiation between the managers (masters) and the contractors (agents) occurs by means of bidding. The master announces a task, the contractors evaluate it vis-a-vis their capabilities and thereafter make bids to the master which allocates the task to the most suitable candidate. Managers may also undertake work instead of sitting idle while the contractors are busy.

Both the above systems contained a single controlling agent. Non-Master-Slave systems (NMSS's) attempt to attain their goals without a central controller by utilising distributed control and communication, a method termed distributed planning. In distributed planning, since all agents must perform more complex communication tasks than simply corresponding with the master, they are assumed to be rational.

Planning occurs either utilises communication or not at all.

Planning without communication proceeds by assuming rationality of the agents and utilising techniques from game theory, in particular, payoff matrices. Each agent utilises these matrices to maximise his gain or payoff *vis-a-vis* the choices of other agents.

Agents planning via communication create their own plans and either know or don't know the states of the other agents. Communication is either via shared memory or an interconnection network and sometimes messages are directed towards agents that are efficient at performing certain tasks. Also, incomplete plans may be initially formulated and may become complete as execution proceeds.

Communication between agents may be divided up into two classes.

Blackboard (BB) systems permit communication via a shared knowledge structure called a blackboard to which agents or modules may post to and read from. Blackboards are similar to shared memories and are thus perfectly suitable for shared memory architectures.

In *message passing systems*, an agent or module sends or receives messages from other agents whose names are explicitly known.

These two systems appear completely different, however one may be remodelled to simulate the other.

The first blackboard system was the HEARSAY-II speech recognition system [Erman & Raj Reddy, 1980]. It comprised a set of independent modules or agents called knowledge sources (KS's) that contain the domain-specific knowledge, a blackboard through which the knowledge sources communicate, and a control system which determines the order that the KS's may operate on the blackboard entries.

In blackboard systems, an agent is ignorant of all the other agents in the system and messages are issued by a central controller (the scheduler). In message passing systems, agents possess knowledge about other agents and messages are sent to specific agents via message passing. Inasmuch, agents may reallocate work to those agents more adept at a certain task.

An example of a multi-agent system is *Aurora* [Desbiens & Nault, 1992], developed on a Transputer cluster. Aurora simulates the crew of an anti-submarine attack aircraft in various situations. The crew duties include navigation, piloting and weapons armament. Each crew member is represented by an autonomous agent corresponding with other agents via message passing. Each agent has it's own local knowledge base which it uses to make deductions commensurate with the crew member it represents. They also formulate their own goals and sub-goals and perform their own tasks. The duties of the agents are the support and analysis of situations which they accomplish by using their local knowledge bases. They are not purely asynchronous as informational messages and orders need to be exchanged between them. All the agents (crew members) co-operate to collectively complete a "mission", a global goal all the agents are aware of.

While DRS's have many advantages, they also posess some serious disadvantages:

(a) How to partition and assign the rule base effectively (rule bases are not decomposable or separable),

(b) how to reconcile the different types of expertise in the system,

(c) how to reconcile subject field overlaps,

(d) the synchronisation of the agents (if needed), and

(e) truth maintenance.

As may be deduced, there is no real dichotomy between blackboard and message passing systems, rather a continuum. At one extreme, a controller broadcasts to all (tightly-coupled - blackboard systems), whereas in the other, any agents may broadcast to any other one. Neuter systems, wherein the agents are ignorant of an other agent, but knows it's class (to which it directs it's messages). In such a way, class hierarchies may be implemented, with more general-purpose KS's or agents higher up in the hierarchy. This saves memory and achieves greater modularity.

4.3 Conclusion

As has been illustrated, parallelising traditional expert systems is no trivial matter. Problems arise, such as the need to lock resources and require developers to write correct programs or assign appropriate priorities to rules so as to avoid inter-rule conflicts. Furthermore, concurrent versions of production systems offer only limited speedups. This may be ascribed to the fact that the production systems architecture is inherently sequential and therefore not amenable to simple parallelisation. It appears that significant speedups are only possible when specialist hardware, such as in IXM-2 and SNAP, is utilised. This is not a commercially viable as such specific architectures are often very expnsive and their general functionality is restricted. Furthermore, knowledge bases are not easily decomposable owing to their strong inter-connectedness, and as it is now generally acknowledged that real AI systems will have to possess huge amounts of knowledge (commonsense knowledge, etc - eg. [Microelectronics & Corporation, 1991]), it becomes infeasable for sequential machines to process these large knowledge bases.

As will be illustrated, the concurrent expert system paradigm argued for in this thesis , *Cogitator*, bypasses all the problems associated with the above systems, and with building knowledge bases, especially large ones. It also utilises fuzzy logic, a generalisation of classical mathematics that has been conclusively shown to model human cognition very closely. It is fuzzy mathematics that is considered in the next chapter, before moving on to discuss *Memory-Based Reasoning* and then *Cogitator*.

Chapter 5

Fuzzy Mathematics

The real world is a dynamic environment with uncertainty and vagueness being immanent. This has been known for many years: almost two millenia ago, Heraclitus stated that " *We never step into the same river twice*".

Fuzziness was a descriptor coined during the 1960's to describe matters of degree and to mathematically mean multivalence. The initial beginnings of *fuzzy mathematics* may be traced to the 1920's and 1930's.

From his investigations into quantum mechanics, Werner Heisenberg observed what was to become *Heisenberg's Uncertainty Principle*, that it is impossible to measure two complimentary properties of an entity to arbitrary accuracy. Consequentially, a third or middle truth was to be introduced into the duality framework of classical mathematics to allow for this *indeterminancy*.

Logical paradoxes, such as the *Sorites* and *Zeno* paradoxes, have been around for many a year. Paradoxes such as

- Does the Liar from Crete tell the truth when he says that he is lying?
- Who shaves the barber if he shaves all those people that do not shave themselves?
- Is A a member of itself if it contains all sets that do not contain themselves?

The concept common to all these paradoxes is that of a *self-referential statement.* They are found in music (Bach's), art (Escher) and in mathematics (Gödel). [Hofstadter, 1980] is a wonderful book highlighting these paradoxes.

It is easy to show that a paradox corresponds to a half-truth (indefinite). Consequentially, ternary and higher-order logics are able to represent the solutions to paradoxes.

Logical paradoxes and Heisenberg's principle led to the development of multivalued logics in the 1920's and 1930's. During the 1930's, the Polish mathematician, Jan Lukasiewicz developed a ternary logic (with truth values 0, $\frac{1}{2}$and 1). He later extended his logic to the rational numbers between 0 and 1 and thereafter to all the numbers in the closed interval $[0,1]$. Other three-valued logics were subsequentially developed. Figure 5 illustrates the various three-valued logics. Also during the 1930's, the quantum philosopher Max

Figure 5.1 Table of Some Three-Valued Logics

Inputs		Lukasiewicz				Bochvar				Kleene				Heyting				Reichenbach			
a	b	$\cap$	$\cup$	$\Rightarrow$	$\Leftrightarrow$	$\cap$	$\cup$	$\Rightarrow$	$\Leftrightarrow$	$\cap$	$\cup$	$\Rightarrow$	$\Leftrightarrow$	$\cap$	$\cup$	$\Rightarrow$	$\Leftrightarrow$	$\cap$	$\cup$	$\Rightarrow$	$\Leftrightarrow$
0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	$\frac{1}{2}$	0	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	1	$\frac{1}{2}$	0	$\frac{1}{2}$	1	0	0	$\frac{1}{2}$	1	$\frac{1}{2}$
0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
$\frac{1}{2}$	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	1
$\frac{1}{2}$	1	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$
1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
1	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Black applied continuous logic componentwise to sets or lists of elements. Each element behaved as a statement in continuous logic. Historically, Black drew the first fuzzy membership functions [Kosko, 1992], calling the uncertainty associated with these structures "*vagueness*".

In 1965, systems scientist Lofti Zadeh published the paper 'Fuzzy Sets' [Zadeh, 1965] thereby formally developing multivalued (infinite-valued) set theory and introducing the term *Fuzzy* [1].

The current surge of interest in fuzzy logic, spurned by Japan's successes with fuzzy logic control systems proves it's suitability as a means of interpreting complex real world systems. The Japanese have recognised this more than anyone else and have made fuzzy computing the major theme of the Real World Computing Project (the Sixth Generation Computer Systems project), which began last year.

Recently, there have been arguments for fuzziness in legal procedures. Comparing a rule-based judge to a fuzzy judge, [Kosko, 1992] illustrates that while the rule-based judge uses a rule-book and chains through it, the fuzzy judge does not use *rules*, but legal *principles*, citing legal precedents and cases to enunciate their relative importance. The

[1]Interestingly, Zadeh's initial ideas concerning *fuzziness* arose out of an argument with a friend about the beauty of each man's wife. Zadeh realised that beauty is not a dicotomous descriptor, but that beauty has degrees associated with it

fuzzy judge then determines a conclusion by combining and *weighing up* these fuzzy facts and principles.

The distinction between rule-based and fuzzy systems then reduces to *rules* verses *principles*. Certain legal theorists have focussed upon this, one aptly summing up the difference as

" *[principles] have a dimension that rules do not - the dimension of weight or importance*"

Furthermore, rules are strict, they come and go as civilisation evolves. Principles are more fundamental, they evolve as civilisation does. Legal principles are rarely excised, but they do evolve.

[Schmucker, 1984] summarises the work of many researchers to conclude that fuzziness is an inherent quality of Human Information Processing (HIP). He cites an example of natural language processing wherein people were tested in assigning linguistic variables to concepts (such as cool or warm to temperature) and numerical values (such as 23°C). The results conclusively indicated the use of linguistic rather than numerical values.

[Schmucker, 1984] proceeds to quote further results illustrating that people represent and recall concepts in accordance with the structure and recall procedures of fuzzy semantic memory [2].

Probability theory was conceived of to explain gambling odds. The rules of gambling are *strict* and consequentially, extensions to Pascal's initial ideas built probability theory atop classical mathematics, in particular measure theory. Probability theory has been applied to many problems, including those which have non-strict governing rules. However, considering Pascal's initial ideas relating to the formulation of probability theory, the application of probability theory to problems and systems which have non-strict governing rules may be deemed to be inappropriate. Examples of such systems include weather, air/fluid flows and economics. For these purposes, *possibility theory*, based upon fuzzy theory is deemed to be more appropriate and has been proposed and used. One might ask what would be the current state of prediction methods if probability theory had been formulated to explain or predict systems with non-strict rules of operation.

In fuzzy theory, entities may have partial memberships in two complementary classes which corresponds closely with HIP. An example may serve to illustrate.

[2] *fuzzy semantic memory* corresponds to semantic memory wherein the relations between entities are not necessarily crisp

The following sections introduce fuzzy mathematics as well as fuzzy expert systems and fuzzy controllers. The emphasis is on merely illustrating some of the main concepts of fuzzy mathematics that are utilised in expert systems, in particular a fuzzy database-driven expert system called *Cogitator*.

5.1 Introductory Theory

Mathematics comprises three main areas, *Set theory*, *Boolean Algebra* and *Propositional logic*. Between any two of these areas, an isomorphic mapping exists, so any theorem in one area has a counterpart in both of the two other areas. Figure 5.1 below illustrates the equivalent symbols used by each of the three areas. Isomorphisms between the three areas are derived using these equivalent symbols. The same organisation applies to fuzzy mathematics. Let X represent the *Universal Set* and x be an element of this set. Zadeh

Figure 5.2 The Equivalent Symbols used in each of the Three Branches of Mathematics

Set Theory	Boolean Algebra	Propositional Logic
$\wp(X)$	B	$Im(V)$
$\cup$	$+$	$\vee$
$\cap$	$+$	$\wedge$
-	-	-
X	1	1
$\emptyset$	0	0
$\subseteq$	$\leq$	$\Rightarrow$

[Zadeh, 1965] extended the *Indicator Function* I_A of a non-fuzzy (crisp) subset A of X which, in classical mathematics is defined as

$$I_A = \begin{cases} 1 & \text{if x} \in \text{A} \\ 0 & \text{if x} \notin \text{A} \end{cases} \tag{5.1}$$

to a *Multi-Valued Indicator Function*

$$I_A : X \mapsto [0,1] \quad A \in X \tag{5.2}$$

This allows the extension of the common operators of classical mathematics as follows

$$I_{A \cap B}(x) = min(I_A(x), I_B(x)) \tag{5.3}$$

$$I_{A \cup B}(x) = max(I_A(x), I_B(x)) \tag{5.4}$$

$$I_{\bar{A}}(x) = 1 - I_A(x) \tag{5.5}$$

$$A \subseteq B \;\; iff \;\; I_A(x) \leq I_B(x) \quad \forall x \in X \tag{5.6}$$

The *Membership Function* $\mathcal{M}(x)$ measures the degree to which an element x belongs to set A, formally

$$\mathcal{M}_A(x) = Degree(x \in A) \tag{5.7}$$

Just as the indicator functions act as statements in classical propositional calculus, membership values $M(x)$ correspond to statements in a continuous logic. Typically, membership and indicator functions map to the closed interval $[0, 1]$. Traditional indicator functions form step functions whereas fuzzy ones form continuous curves over the range.

This extension to multi, rather infinite, valued sets causes classical mathematics to become a special case of fuzzy mathematics. The principle that applies to allow the generalisation of crisp to fuzzy mathematics is termed the *Extension Principle.* Since Zadeh introduced the aforementioned fuzzy union, intersection and complementation operations, other unions, intersections and complementation operations have been developed. They need only obey a collection of axioms pertaining to fuzzy union, intersection and complementation. Some of these fuzzy union, intersection and complementation functions are reproduced in figure 5.1 below.

Figure 5.3 Table of some Fuzzy Union and Intersection Operators

Name	Fuzzy Union	Fuzzy Intersection	Parameter
Yager	min[1, $(a^w + b^w)^{\frac{1}{w}}$]	1 - min[1, $((1-a)^w + (1-b^w))^{\frac{1}{w}}$]	$w \in (0, \infty)$
Dombi	$\frac{1}{1 + [(\frac{1}{a} - 1)^{-\lambda} + [(\frac{1}{b} - 1)^{-\lambda}]^{-\frac{1}{\lambda}}}$	$\frac{1}{1 + [(\frac{1}{a} - 1)^{-\lambda} + [(\frac{1}{b} - 1)^{-\lambda}]^{-\frac{1}{\lambda}}}$	$\lambda \in (0, \infty)$
Dubois & Prade	$\frac{a + b - ab - min[a,\, b,\, 1-\alpha]}{max[1-a,\, 1-b,\, \alpha]}$	$\frac{ab}{max[a,\, b,\, \alpha]}$	$\alpha \in (0, 1)$

All the union, intersection and complementation operation definitions illustrated above reduce to classical mathematical intersections and unions when x is restricted to $\{0, 1\}$ (by definition). The intersection, union and complementation operators (5)-(7) above may be termed the *Standard Operations* of fuzzy set theory. When used to define a fuzzy set theory, they define *Possibility Theory*, a contingency theory supersuming both *Probability* and *Dempster-Schafer* theory.

This generalisation of crisp mathematics violates the classical laws of *Excluded Middle*

$$A \cup \bar{A} = X$$

and *Non-Contradiction*

$$A \cap \bar{A} = \emptyset$$

The *fuzzy Indicator Function* has since also been extended to

$$I_A(x) : X \mapsto \mathcal{L} \tag{5.8}$$

where $\mathcal{L}$ is any set that is at least partially ordered. $\mathcal{L}$ is frequently a lattice, an when it is, fuzzy sets defined by above are termed $\mathcal{L}$ - *fuzzy sets*. Again, typically $\mathcal{L} = [0, 1]$.

Fuzzy sets are commonly represented as follows:

$$A = \sum_{i=1}^{n} M_A(x_i)/x_i \tag{5.9}$$

where $x_i \in A$ and A is a fuzzy set. For instance,

5.1.1 Definitions

Some of the main definitions pertaining to fuzzy set theory are reproduced below, the interested reader is referred to any books comprising the vast literature on fuzzy logic, such as [Klir & Folger, 1988].

Definition 5.1.1 The *Support* of a fuzzy set $A \in X$ is the crisp set containing all elements of X that have a non-zero membership grade in A. Formally,

$$supp : I^A \mapsto 2^A \tag{5.10}$$

where 2^A is the power set of A and

$$supp(A) = \{x \in X \mid \mathcal{M}(x) > 0\} \tag{5.11}$$

Definition 5.1.2 The *Height* of a fuzzy set A is the largest membership grade attained by any member of that set.

Definition 5.1.3 A fuzzy set is said to be *Normalised* if at least one of the members attains the maximum grade, which is typically 1 (i.e. it's height is 1).

Definition 5.1.4 An $\alpha - cut$ of a fuzzy set A is the crisp set A_α of elements that attain a membership of at least α. Formally,

$$A_\alpha = \{x \in X \mid \mathcal{M}(x) \geq \alpha\} \tag{5.12}$$

Definition 5.1.5 A *level set* of A is the set of all distinct $\alpha - cuts$ of the fuzzy set A. Formally,

$$\bigwedge_A = \{\alpha \mid \exists x \in X;\ \mathcal{M}(x) = \alpha\} \tag{5.13}$$

Definition 5.1.6 $|A_\beta|$, the *Cardinality* of an α-cut β of a fuzzy set A is defined as the number of members in A_β.

Definition 5.1.7 The *Scalar Cardinality* of a fuzzy set A defined on a finite X is the summation of the membership grades of those elements of X in A. Formally

$$|A| = \sum_{x \in X} \mathcal{M}_A(x) \tag{5.14}$$

Uncertainty is a naturally universal phenomenon associated with real-world and very complex systems. It is the main aim of *Cogitator* to overcome uncertainty by utilising fuzzy methodologies and techniques. It is these that are now considered.

5.2 Uncertainty

For centuries now, man has attempted to characterise uncertainty. His initial attempts led to the development of probability theory and, subsequently, to Dempster-Schafer and possibility theory. Some theories are based upon classical bivalent mathematics (probability theory) while others are based upon the more general theory of fuzzy mathematics (possibility theory). Each of these theories occupies its own little niche in uncertainty theory and has been applied to a variety of problems. This section seeks to illustrate how fuzziness has been introduced to interpret uncertainty and to illustrate some of the methodologies used to characterise the various types of uncertainty. The system proposed in this thesis, *Cogitator*, is designed to utilise these fuzzy uncertainty theories for the interpretation of data.

5.2.1 Types of Uncertainty

Consulting a dictionary for a meaning of uncertainty yields terms such as vague, indefinate, doubtful, ambiguous, varying, inconsistent, unreliable, not known for certain, and so on.

From these, two main areas may be identified: *vagueness* and *ambiguity*. *Vagueness* implies not sharply delineated, indistinct or hazy, whereas, *ambiguity* signifies one-to-many, variety or a choice between many alternatives. The basic mechanism for characterising vagueness and uncertainty are, respectively, *fuzzy sets* and *fuzzy measures*.

Vagueness is characterised by *Measures of Fuzziness*. Typically, a measure of fuzziness takes the form of a function:

$$\mathcal{F} : \mathcal{P}(X) \mapsto \Re$$

where $\mathcal{P}(X)$ represents the power set of X

There is also a collection of axioms that all measures of fuzziness must obey; they are not reproduced here but may be found in any book on fuzzy mathematics. Measures of fuzziness are characterised by a distance function, such as the Hamming, Euclidean or Minkowski metrics. The metrics are in terms of a fuzzy set and the nearest crisp set (an *Index of Fuzziness* or between a fuzzy set and its complement. The general form of measures of fuzziness is represented as

$$\mathcal{F}(A) = D_c(Z, \overline{Z}) - D_c(A, \overline{A})$$

where Z denotes an arbitrary subset of X such that $D_c(Z, \overline{Z})$ is the largest possible distance in P (X) for some complement function,. $A \subseteq X$ is a fuzzy subset, and $D_c(A, \overline{A})$ represents the distance between A and its complement (for some complement function).

Some examples are reproduced below ($\mu_c(x)$ = membership of x in a crisp set).

$$\text{Hamming} : \mathcal{F}(A) = \textstyle\sum_{x \in X} \mid \mu_A(x) - \mu_c(x) \mid$$
$$\text{Minkowski} : \mathcal{F}(A) = (\textstyle\sum_{x \in X} \mid \mu_A(x) - \mu_c(x) \mid^{w})^{\frac{1}{w}}$$

Analogous formulae exist when, instead of the closest crisp set, the complement of a fuzzy set is used. The formulae for measures of fuzziness illustrated above may be extended to infinite sets by considering X to be an interval and replacing the summation by an integral with appropriate limits.

Ambiguity may be further resolved into 3 types. When the *size* of the subsets of X are being considered as a measure of ambiguity, the measure is termed a *Measure of Nonspecificity*. As the size grows, the amount of specificity decreases increasing ambiguity. When subsets of X that do not or partially overlap are considered, conflicts between the evidence supporting the sets may arise. In this case, a *measure of Dissonance* is used.

Finally, if a number of fuzzy subsets are designated as possible locations of a concept y, then problems arise owing to the multitude of candidate sets. Measures of ambiguity using this method are termed *Measures of Confusion.* Figure 5.2.1 contains the general forms of these three measures of ambiguity [3].

Figure 5.4 The General Forms of Measures of Ambiguity

Nonspecificity :

$$V(m) = \sum_{A \in \mathcal{E}} m(A) Log_2 \mid A \mid$$

Dissonance

$$E(m) = -\sum_{A \in \mathcal{E}} m(A) Log_2(Pl(A))$$

Confusion

$$C(m) = -\sum_{A \in \mathcal{E}} m(A) Log_2(Bel(A))$$

Fuzzy measures of uncertainty were preceded by other measures based upon classical mathematics and probability theory. The most common class is *Entropies*, which arose in physics (theory of heat) in the middle of the 19'th century. The most popular entropy is *Shannon Entropy.* The general form of Shannon entropy is

$$H(p) = -\sum_{i=1}^{n} p_i Log_2(p_i)$$

where $p = (p_1, p_2, \cdots, p_n)$ is a probability distribution. There are also other entropies, but they generally form soecial cases of Shannon entropy as may be deduced from figure below.

Figure 5.5 The General Forms of Entropies

Name	Formula
Reyni Entropies	$H_\alpha(p_1, p_2, \cdots, p_n) = \frac{1}{1-\alpha} Log_2(\sum_{i=1}^{n} p_i^\alpha$
Order β Entropies	$H_\beta(p_1, p_2, \cdots, p_n) = \frac{1}{2^{1-\beta-1}}((\sum_{i=1}^{n} p_i^\beta - 1)$
R-Norm Entropies	$H_R(p_1, p_2, \cdots, p_n) = \frac{R}{R-1}(\boxed{1 - (\sum_{i=1}^{n} p_i^R)^{\frac{1}{R}}}$

Shannon Entropy was restricted to finite sets which need not be the case. The *Boltzmann Entropy* has a similar form to the Shannon one, but is defined for infinite sets and has the following general form:

[3](m is a *Basic Probability Assignment*, E is the set of Focal Elements, Bel(A) and Pl(A) are *Belief* and *Plausability* measures respectively)

$$B(q(x) \mid x \in [a,b]) = -\int_a^b q(x)Log_2(q(x))dx$$

Another lesser used entropy measure is the *Hartley Information*, a measure based upon classical set theory. It assumes the general form

$$I(N) = Log_2(N)$$

where N is the number of possible alternatives (one or a sequence of selections).

Cogitator is a general paradigm for a new type of fuzzy expert system, therefore any of the aforementioned measures of uncertainty may be used by the inferencing process utilised. The choice is the discretion of the system builder.

5.3 Membership Functions

Quintessential to fuzzy inference and retrieval is the notion of a *Linguistic Variable*, which may be defined as follows.

Definition 5.3.1 [Zimmermann, 1986]
A *Linguistic Variable* is a fuzzy set characterised by the quintuple

$$< X, T(X), U, G, \tilde{M} >$$

where

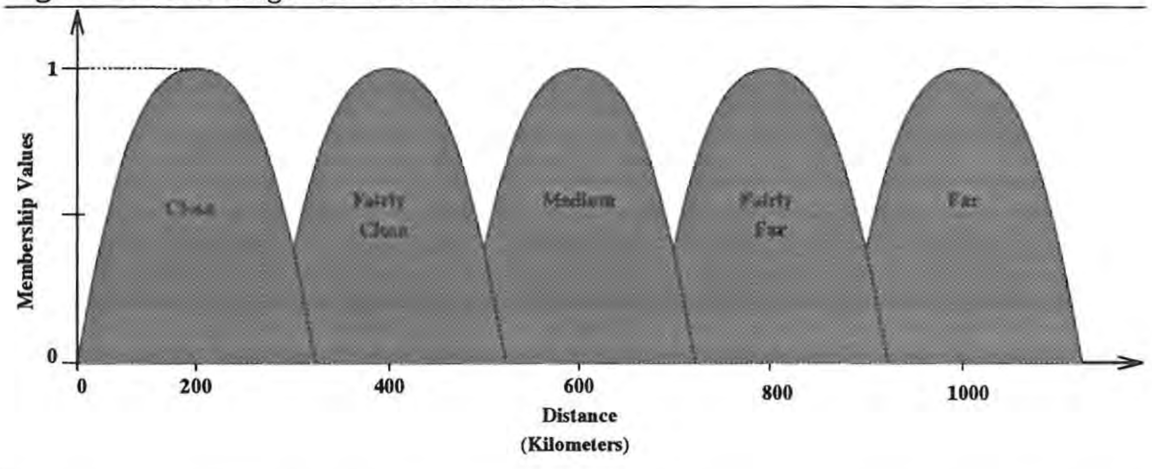
- X is the name of the linguistic variable,
- T(X) is the term set, the set of linguistic *values* of X,
- U is the *Universe of Discourse* associated with the base variable u,
- G is a *syntactic Rule*, often a grammar, for generating the name, X, of values x
- $\tilde{M}$ is the *Semantic Rule* for associating each x with its meaning; $\tilde{M}(x) \subseteq U$.

The element x generated by G is called a *term*.

To illustrate this definition, consider an example. Figure 5.6 below illustrates the linguistic variable *Distance*.

For this example, X is the Linguistic variable *Distance*, the universe of discourse, U, may be taken as [0, 20 000]. The terms or linguistic values, x, of this fuzzy set X may be

Figure 5.6 The Linguistic Variable Distance



Very Close", *Close*, *Far* and so on. i.e. T(x) = (very close, close, medium, far, very far). $\tilde{M}$ (x) assigns a meaning, a fuzzy set, to these terms x; in this case,

$$\tilde{M}(x) = \{(u, \mu_x(u) \mid u \in [0, 20000]\}$$

where $\mu_x(u)$ is referred to as the *Membership Function* associated with the linguistic variable x; for instance $\mu_x(u)$ may be given as

$$\mu_{Close}(u) = \begin{cases} (1 + e^{-\frac{x}{10}})^{-1} & u \in [0, 400] \\ 0 & Otherwise \end{cases}$$

G(x) generates the labels of the terms in the term set u, the grammar encapsulating the terms close, far, etc.

The two linguistic variables of particular interest are *Truth* and *Probability*. For more information, the interested reader is referred to [Zimmermann, 1986] for more details.

Two more definitions before rounding off.

Definition 5.3.2 A *Linguistic Hedge* or *Modifier* is an operation that modifies the *meaning* of a term or a fuzzy set. If F is a fuzzy set, then the linguistic modifier M generates the fuzzy set G = MF.

Two important linguistic modifiers are:

- Concentration : $\mu_{Con(A)}(u) = (\mu_A(u))^2$
- Dilation : $\mu_{Dil(A)}(u) = (\mu_A(u))^{\frac{1}{2}}$

For a term or fuzzy set A,

- Very A = Con(A)
- More or Less A = Dil(A)

Definition 5.3.3 A Linguistic Variable X is termed *Boolean* if all the terms (linguistic values) are boolean expressions of the form x_B, $M_B(x_B)$ where x_B is a primary term and M_B a modifier.

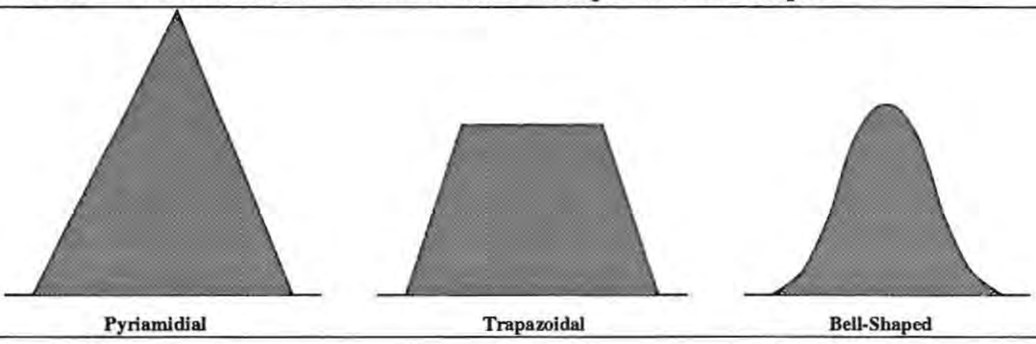
For example,

- Young $\rightarrow$ Not Young $\Rightarrow$ m_B = "Not"
- Old $\rightarrow$ Very Old $\Rightarrow$ m_B = "Very"

From these, terms such as "very (not Old)" and "more or less (not old)" may be derived.

There are three main forms that term set elements assume (i.e. their membership functions), *Bell-Shaped*, *Trapazoidal* or *Pyriamidial*, all convex forms. They are depicted in figure 5.7 below. Nevertheless, variations on these are possible.

Figure 5.7 The Three Most Common Membership Function Shapes



The choice of membership functions and their associated shapes as well as the number of linguistic vallues or terms to have, are of cardinal importance. Which types of membership functions are suitable for which types of problems is a still unsolved problem. Concerning the number of terms to use, five or seven represent a general consensus, although five seems to be more common.

membership functions play a cardinal role in fuzzy expert and control systems, which are discussed in the next section.

5.4 Fuzzy Expert Systems, Control Systems and Applications

Recently, expert systems and control systems based upon fuzzy logic have balooned in number. Various methodologies for incorporating fuzzy logic into these systems have been proposed and successfully implemented in various commercial and private projects. What follows is a brief descrription of fuzzy expert systems, fuzzy control systems and the equivalences between them and neural networks. Finally, some applications of these fuzzy systems are given.

5.4.1 Fuzzy Expert Systems

Most expert systems are represented in the form of a collection of rules, R_i, called *Production Rules*, rules of the form

$$R_i : \text{IF} < \text{Antecedents} > \text{THEN} < \text{Consequents} >$$

where the antecedents and consequents are collections of statements (at least one) of the form

$$X_i = A_j$$

where X_i represents the variable and A_j the value of the statement, linked by the logical operators AND, OR and NOT. The collection of rules in a fuzzy expert system is known as the *Rulebase* or *Knowledge base.*

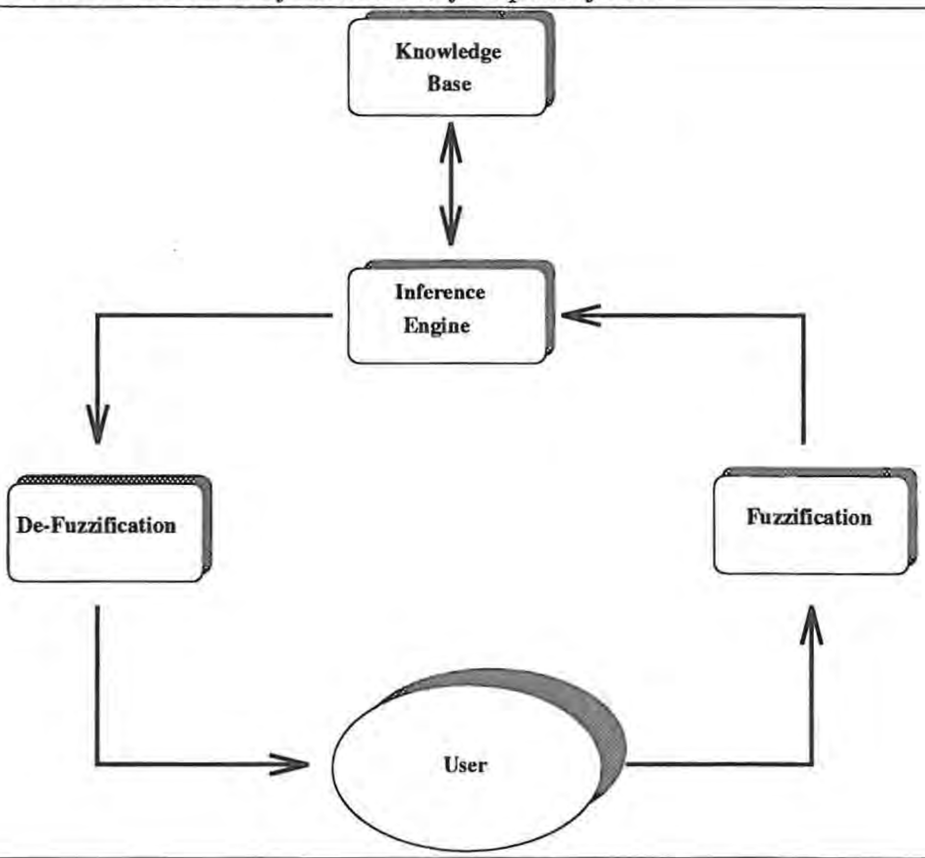
Fuzzy expert systems are represented as such except that the variables and values used in the antecedent and consequent statements are considered to be fuzzy sets as opposed to singletons, and are referred to as, respectively, linguistic variables and linguistic values (as mentioned earlier). Furthermore, the logical operators are not Boolean ones, but fuzzy. The antecedent (the rule's premise) describes to what degree the rule applies, while the conclusion (the rule's consequent) assigns a membership function to each of one or more output variables. For example, a collection of three rules for controlling the current to an air-conditioner Might be

- IF temperature is high AND humidity is high THEN current = high
- IF temperature is low AND humidity is low THEN current =low
- IF temperature is high AND humidity is low THEN current = medium

where*temperature*, *humidity* and *current* are considered to be the linguistic variables and *high*, *low* and *medium* the linguistic values or labels.

The execution cycle also differs, comprising the stages: *Fuzzification*, *Inference*, *Composition* and *De-Fuzzification*. The execution cycle is represented in figure 5.8 below [4].

Figure 5.8 The Execution Cycle of a Fuzzy Expert System



- [1] *Fuzzification* entails translating a crisp value into a fuzzy one; for instance for temperature, $43^o c$ translates into the fuzzy value *High*.
- [2] Under *Inference*, the truth values for each antecedent statement of each rule is determined. This results in one fuzzy subset to be applied to each consequent value for each rule. Commonly, the *MIN* and *Product* functions are used as inference rules In *MIN* inferencing, the output membership function is clipped off at a height corresponding to the rule premise's computed degree of truth (fuzzy logic AND). In *PRODUCT* inferencing, the output membership function is scaled by the rule premise's computed degree of truth.

[4]In this figure, the inference engine comprises the inference and composition stages

- [3 Under *Composition*, all of the fuzzy subsets assigned to each output variable are combined together to form a single fuzzy subset for each output variable. Again, usually MAX or SUM are used. In MAX composition, the combined output fuzzy subset is constructed by taking the pointwise maximum over all of the fuzzy subsets assigned to variable by the inference rule (i.e. fuzzy OR). In SUM composition, the combined output fuzzy subset is constructed by taking the pointwise sum over all of the fuzzy subsets assigned to the output variable by the inference rule.

- [4] Finally is the (optional) *De-Fuzzification*, which is used when it is useful to convert the fuzzy output set to a crisp number. There are many defuzzification methods, however two of the more common techniques are the *Centroid* and *MAX* methods. In the *Centroid* method, the crisp value of the output variable is computed by determining the center of gravity of the membership function for the fuzzy value. In the *MAX* method, one of the variable values at which the fuzzy subset has its maximum truth value is chosen as the crisp value for the output variable.

Fuzzy expert systems have exhibited great success in the modelling of complex processes, whether for inferencing or for controlling them.This may be attributed to the use of fuzzy sets to allow a certain degree of tolerance.

The most popular use of fuzziness in decision-making is in the controlling of complex processes. This gives rise to fuzzy control systems which are now (briefly) considered.

5.4.2 Fuzzy Control Systems

Fuzzy control systems may be considered to be a slight variation on fuzzy expert systems, in fact fuzzy control systems may be considered to be an example of a fuzzy expert system. As in fuzzy expert systems, fuzzy control systems also have a rule-base of rules with fuzzy variables and their associated values, as well as the four stages of operation that fuzzy expert systems comprise. Most of the techniques developed for fuzzy expert systems have been extended for use in fuzzy control systems.

The use of fuzzy control systems was first popularised by the Japanese under the auspiuces of their Fifth-Generation Computer Systems project (FGCS). Since then numerous applications of fuzzy control techniques to complex processes have occurred.

The latest foci in fuzzy control systems is to construct *adaptive* fuzzy control systems. The characteristice of systems to be controlled may vary over time, for instance as a machine operates, it's parts are subject to wear and tear. For these types of reasons, adaption has been introduced into fuzzy expert systems to compensate for this temporal behaviour. Two main methods for accomplishing this are the modification of certain weights associated with the rules in the rule-base, and to widen or narrow the membership functions corresponding to certain linguistic values of a linguistic variable. Adaptive fuzzy control sysetms have been shown to be extremely robust.

By considering continuous processes as continuous systems with a vector input and a vector output, [Buckley, 1992] shows that neural networks, fuzzy expert systems and fuzzy control systems may all be deemed to be equivalent (with some minor considerations).

The use of fuzziness in control and expert systems may be deemed analogous to the use of the complex domain, fuzzy expert and control systems translate inputs and operate upon them in the fuzzy domain, to analyse difficult integrals, it is convenient to jump into the complex domain and make use of residues and complex numbers.

5.4.3 Applications

Fuzzy expert and control systems have been utilised to interpret and control a plethora of real-world systems. Fuzzy expert systems working within domains such as medicine, geology and finance have been successfully developed. Of note is that the Japanese Securities firm, Yamaichi Securities, deploys a fuzzy expert system to manage clients' stock portfolios and yields an average return of 35% on investments!

Fuzzy control systems are far more numerous, having been applied to aeronautics (flight control systems and shuttle docking), railway train driving (the Sendai Railway), image stabilisation (panasonic's video recorder). Figure 5.9 below gives a brief summary of some of the applications recently developed.

5.5 Conclusion

At this stage, fuzzy mathematics, expert systems and their fuzzy counterparts have been illustrated. The reader may ask whether the problems associated with traditional (non-fuzzy) expert systems highlighted earlier, extend to fuzzy expert systems. The answer is yes, albeit to a lesser degree. By allowing fuzzy variables, a single fuzzy expert system rule encodes two or more traditional rules, i.e. a fuzzy expert system utilises *fewer* rules to encode the same knowledge as a traditional expert system. However, the knowledge

Figure 5.9 Some Applications of Fuzzy Control

Product	**Company**	**Fuzzy Logic Role**
Elevator Control	Fujitec/Matsushita	Evaluates Passenger Traffic to reduce waiting times
Video Camcorder	Sanyo Fisher / Canon	Auto focus and exposure when several when several objects are in the picture
Washing Machine	Matsushita	Senses quality and quantity of dirt, load size fabric type and adjusts wash cycle accordingly
Vacuum Cleaner	Matsushita	Senses floor condition and dust quantity and adjusts the cleaners' motor power accordingly
Air Conditioner	Mitsubishi	Determines optimum constant operating power level to preclude power-consuming on-off cycling
Television	Sony	Adjusts screen brightness, colour and contrast
Handheld Computer	Sony	Interprets (Japanese) handwritten input for data entry
Auto Transmission	Subaru	Monitors driving style and engine load to select the best gear ratio
Stock Trading Program	Yamaichi Securities	Manages stock portfolios (returns 35% annually)

acquisition bottleneck is still present and the problems of consistency and completeness of a knowledge base still abound, if to a lesser degree.

For these reasons, the expert system argued for in this thesis, while utilising fuzzy mathematics, is not a fuzzy expert system. Rather the quintessential aspect of *Cogitator* is that it seeks to circumvent the knowledge acquisition bottleneck and as many of the other problems associated with traditional expert systems, as possible. It utilises certain aspects of fuzzy mathematics, in particular fuzzy measures of uncertainty and similarity with the aim of modelling human cognition (problem-solving) more closely.

It is therefore appropriate to introduce an alternative methodology, *Case-Based Reasoning* and then consider a dialect thereof, *Memory-Based Reasoning*, the reasoning scheme upon which *Cogitator* is based.

Chapter 6

Memory-Based Reasoning

There is a plethora of reasoning paradigms that seek to simulate human cognitive processes with the aim of building intelligent systems. Amoung them are *Case-Based Reasoners* which have shown success in modelling Human Cognition. This section seeks to introduce Case-Based Reasoning (CBR) and to elucidate a dialect of CBR called *Memory Based Reasoning* (MBR). Initially, the concepts associated with MBR are discussed in detail as well as it's advantages and disadvantages over traditional (rule-based) expert and connectionist systems. Thereafter, three MBR systems developed to date are briefly discussed and, finally, the differences between MBR, Deductive Databases (DDB's) and Information Retrieval (IR) systems are illustrated.

6.1 Case Based Reasoning

CBR may be defined as " A psychological model of human cognition leading to an Analogical Reasoning Methodology [Slade, 1991]. It is based upon the examination of the natural reasoning that people perform and developing a cognitive model thereof.

CBR is consistent with much of what phsychologists have observed in human problem-solving. [Kolodner, 1991] recounts numerous experiments and observations performed by phsychologists and herself in her laboratory at the Georgia Institute of Technology, irrefutably establishing that analogical reasoning is performed by humans when they perform problem solving and decision making. Furthermore, it was observed that people use previous experiences to suggest plausable solutions which are then adapted and evaluated accordingly. It was also observed that in the natural situations that the subjects were placed in, the use of analogues were more significant than the application of abstract principles (what science generally attempts to do), rules (as traditional expert systems do) or conscious deliberation with alternatives. Also, the fundamental advantage that the recollection of cases provides is that they allow the decision maker to deal with unknown and uncertain information, as well as to explain anomalous situations.

The methodology used by humans to solve problems is essentially CBR : Lawyers use cases as precedents to construct and justify arguments in new cases; mediators and arbitrators are taught to do likewise, as are doctors. CBR is also evident in our everyday actions, whether preparing a meal, deciding what to do first or even cleaning a pool.

Expert systems often utilise rules to produce a solution. They are successful for problem solving in well circumscribed domains. However, they are not proficient at solving problems that require creativity, broad common sense knowledge or escetic judgement owing to their brittleness. This will be discussed in more detail later.

Consider an example. A doctor has to diagnose a patient that exhibits an unusual combination of symptons. If the doctor has previously encountered patients with similar symptoms, then they are likely to be recalled. and a similar diagnosis to those of the recalled cases would be proposed. Since medicine often allows for the possibility of more than one diagnosis corresponding to a collection of symptoms, a proposed diagosis based upon a similar past case may not be assumed to be irrefutably correct. Hence the doctor needs to validate the proposed diagnosis vis-a-vis the symptons of the current patient to determine whether there is a more apt diagnosis. The recollection of similar past cases allows the doctor to generate a plausable answer as a starting point easily.

Doctors evaluating a proposed diagnosis or judging which of several are appropriate, make their judgements based upon their previous experiences. Problem instances - instances where a diagnosis was incorrect - are also helpful since they indicate what could go wrong.

While a doctor knows the etiology (progression) of certain disorders, he cannot be trained to identify all possible combinations of disorders. Furthermore, for doctor to use their knowledge of disorder etiologies to repeatedly generate plausable diagnoses is too time consuming to use every time a patient is to be diagnosed. If the knowledge of the disease process was used once to solve a difficult problem, it is sensible to store the solution such that it may be re-used! That is, the doctor will thus be able to recognise a subsequent combination of disorders that are similar to the one that was stored without having to repeat the inference process necessary to arrive at the conclusion again (unlike traditional expert systems).

Therefore CBR is useful to those who have sufficient knowledge about a task or a domain as it allows them to re-use the results of previous hard reasoning. It is also useful to those who know a little about a domain - what others did may be recalled and their actions adapted accordingly. CBR is also useful when information is incomplete or evidence is sparse. Logical systems have trouble dealing with either of these situations since they need to base their answers upon what is well known and sound. Some AI systems use certainty factors and probabilities to counter these problems of rigidity, however all of these require considerable effort on behalf of the computer (computation) and the designer (hand-tailoring and derivation), none of which seem intuitive. A CBR system makes assumptions to augment incomplete or missing knowledge based upon what experience dictates, and progresses from there. Answers generated in such a way will not always be correct, not even optimal, but if the reasoner carefully evaluates its solutions, CBR provides a way to easily generate answers.

There are two styles of Case-Based Reasoning - the *Problem-Solving* and *Interpretative styles*[Kolodner, 1991]. In the Problem-Solving Style, solutions to current problems are derived using old solutions as a guide. As already mentioned, old solutions may provide almost correct solutions to or warnings of failure concerning certain solutions to a new problem. Case-based Reasoning of this style supports a variety of problem-solving tasks including planning, diagnosis and design.

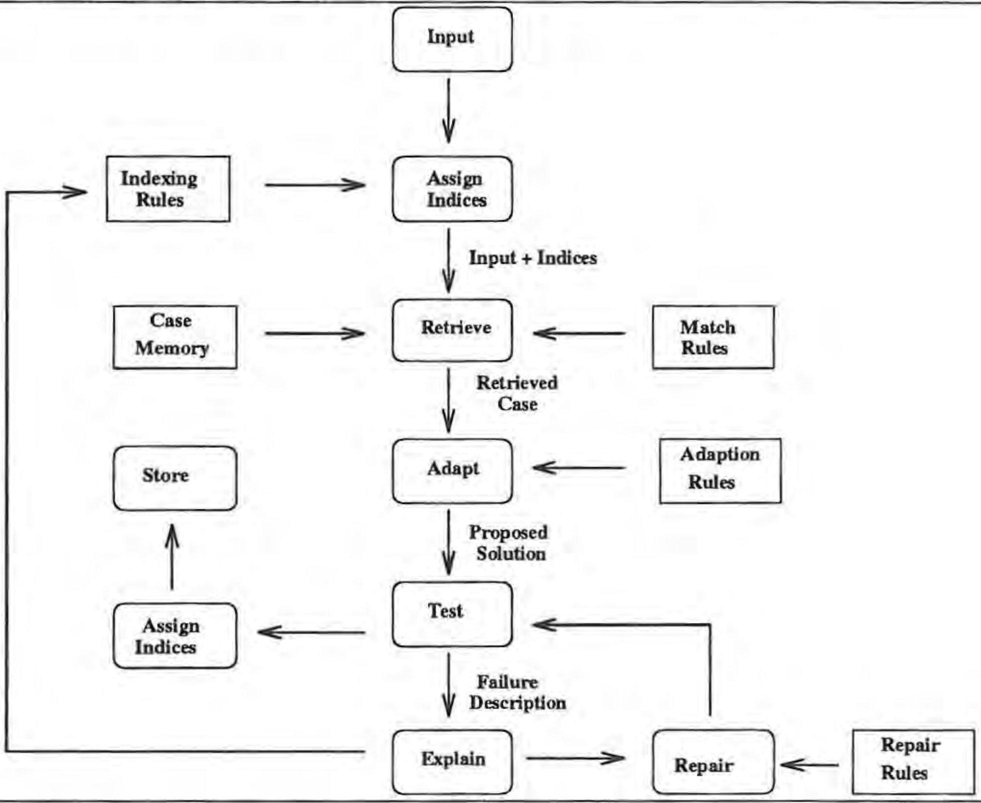
For the Interpretative Style, new situations are *evaluated* in terms of old ones: a lawyer uses this style of CBR to substantiate his arguments for or againsed a particular legal argument. Interpretative CBR is often used to *weigh up the odds* of a problem solution. People making strategic decisions tend to use the interpretative style. Generally, the interpretative style is appropriate for situation classification, the evaluation and justification of a solution, argumentation and the projection of the effects of a decision or solution.

The two different styles of reasoning require that different reasoning be done once cases have been retrieved. In Problem-solving CBR, a solution is proposed by retrieving appropriate cases. Thereafter, the modification of old solutions to fit new situations is performed - called *adaption.* Finally, the new solution is evaluated - called *criticism* - before proposing it. For interpretative CBR, an interpretation or result is proposed either based upon retrieved cases or imposed from the outside (eg: when a lawyer's client stipulates a desired result). Thereafter, the process of *justification* is performed, whereby an argument for the proposed solution is created by comparing the proposed case to prior cases. Finally, the process of criticism whereby an argument is analysed by generating hypothetical situations and applying the argument to them is utilised.

6.1.1 Design of a CBR System

The general design and the functioning of a CBR system is depicted in figure 6.1. The

Figure 6.1 The General Design of a Case-Based Reasoner
(Source: [Riesbeck & Schank, 1989])



cardinal aspect of CBR is the retrieval of appropriate cases. This is termed the *Indexing Problem* [Kolodner, 1991]. Essentially, the problem is one of assigning appropriate labels to cases when they are placed into the case memory such that they may be retrieved appropriately. These labels indicate under what circumstances a case might be retrieved and provides a means of deciding the appropriateness of a case *vis-a-vis* the query case. This requires an understanding of the notion of similarity which is discussed later.

People tend to be proficient at using cases to solve problems for which there is incomplete or uncertain information. However, they are not proficient at recalling the correct cases when there is a myriad of information to remember [Kolodner, 1991]. [Kolodner, 1991] refers to phsychologists who observed that people sometimes recall an experience without validating it *vis-a-vis* the new situation and they are often not reminded of the most appropriate cases during reasoning. Also, novices have the additional problems associated with missing experiences that would constitute cases to be recalled for apt analogical decisions. In effect, their ability for judging the similarity of cases is deficient.

Computers form the complement of humans in this sense: they are proficient at remembering cases but not at using the cases to make decisions, whereas people are proficient at organising information, but not at recalling all the appropriate cases.

Consequentially, [Kolodner, 1991] proposes the development of CBR systems to aid the decision-making capability of people, *Case-Based Aiders* augment a person's memory by providing analogues to a query case which a person may then use to solve problems.

Now that the main aspects of CBR have been introduced, we now turn to Memory-Based Reasoning, a dialect of CBR and a main topic of this thesis.

6.2 Memory Based Reasoning

Memory Based Reasoning (MBR) is an empirical data-driven reasoning scheme that makes *direct* references to a large database of specific episodes (an episodic memory), rather than using a knowledge base of rules and facts to reason. It is as if the MBR system seeks to make decisions by "remembering" similar past circumstances and directly recalling those that are the most similar to the current query case.

The rationale behind this methodology is that it is difficult to conceive of intelligence without cognition and common sense, which are in turn, difficult to conceive of without a memory of past experiences. To illustrate, consider the simple task that we do every day: reading. When reading a book, the story progresses by describing certain events and then others. Often, in order to understand later events properly, recollection of past events is essential. Similarly in the sciences, definitions, theorems and certain introductory concepts are initially given and are subsequently used to develop other or more advanced aspects of the subject. The whole notion of theorems that started with the ancient Greeks is based upon this. It is not to say that memory is the sole important aspect of intelligence, but it is a very important one.

Before progressing with MBR, the notion of similarity, a quintessential aspect of MBR needs to be considered.

6.3 Similarity

As already stated, the real world is a dynamic environment immanent with uncertainty. Whereas entities are different and are evolving, there still appears to be a strand of similarity between them. This strand allows us to generalise, learn from past experience and to develop contingency plans.

Generalisation may be considered a form of abstraction. Of cardinal importance to intelligent systems, it aids in the recognition of similar objects and noise compensation, two attributes that humans are particularly adept at. This notion of similar inputs yielding similar outputs is characteristic of many real-world systems. Neural networks seek to replicate this behaviour by utilising a simplistic representation of the brain.

Is this *generalisation via similarity* also characteristic of humans? Consider the important notion of *categorisation.* In order to categorise entities, computer scientists have formed lists of attributes and matched them againsed other lists to identify entities in the same categories. However, this approach is flawed. Most cats meow, but not all. (eg. Lions and Tigers which *growl*). Some birds fly, others do not (eg. Ostrich and Emu). Ornaments come in many shapes and sizes and are made from plastic, steel, wood or some other exotic material.

A possible solution to the *categorisation problem* is to develop *prototypes*, a collection of examples characteristic of a certain category. This approach has significant phsychological support and is gaining acceptance in the form of Case-Based Reasoners and already in Neural Networks. In line with this, an MBR system computes the similarity between a query and a collection of prototypical cases, producing similar outputs when posed with similar inputs.

This *Generalisation by Similarity* has some interesting phychological observations encountered in everyday life. People consider good or prototypical birds as doves, sparrows or pidgeons. Ostriches and Emus are termed *bad* examples of birds. Almost all mammal classes comprise males and females, however the Black-Backed Jackal of Australia is epicene. Batrachian amphibians discard their gills and fins when adults (eg. tadpoles - frogs). The mudhopper is a swamp-dwelling fish that occasionally leaves the water, often climbing the stems of swamp plants. Oddly, people tend to remember these exceptions very well.

One may state that these exceptions are enumerative and may be classified accordingly.

However, similarity as a *general* descriptive model is offered. According to similarity, then, there exists a continuum between all classes, and examples of these continua exist only if nature allows them, for instance a human with no legs nor arms would not survive very long on earth, so no such specimens exist. The significant inter-class overlap arising from certain entities having qualities of many classes serves to illustrate the concept of *continuity*. The characteristics of land-amphibian-sea mammals change gradually with respect to their main functions. This continuity precludes strict classifications leading to the appropriatness of fuzziness or fuzzy similarities, a cardinal aspect of the system proposed in this thesis.

Nevertheless, similarity is not a flawless approach. While it describes HIP very well [Kosko, 1992], an example will serve to illustrate where it fails. The classical *parity* function which sets a bit according to the number of 1's in a binary vector exhibits total oppositeness for the simple reversing of a single bit. This limitation and the XOR function caused a decline in neural network research in the 1960's.

Neural networks simulate both pre-attentive and attentive processing. Pre-attentive processing includes our unconscious operations of image segmentation and contrast enhancement in our visual systems, low-pass filtering performed by our cochlea as well as word and syllable recognition in our hearing system. Attentive processing encompasses the analysis of the image segments and syllables. We look, see (pre-attentive), pay attention (attentive) and then recognise quickly and efficiently. However, the higher level cognitive functions, those of decision-making, planning and reasoning with noisy, scant or uncertain evidence, our "*cognitive calculus*" [Kosko, 1992], is a result of natural selection and subsequent *experential refinement*. These are the functions that fuzzy mathematics seeks to simulate and MBR seeks to replicate simply.

Both neural networks and fuzzy systems are *model-free estimators*, in that no model of the system to be analysed is needed. They break with the traditional Western view that entities must be observed, defined, quantified exactly and then manipulated by formal methods. Very complex real world systems have been shown to be un-characterisable by these strict methodologies: quantum mechanics uses probability theory in its formalisations and attempts to introduce fuzzy logic are currently underway. In [Kosko, 1992], Richard Anderson recounts the words of Huang Po, a 9-th century Buddist monk

... from discrimination between this and that, a host of demons blazes forth

To summarise, then, similarity is a pervasive aspect of nature and human reasoning and is therefore regarded as a quintessential aspect of MBR and *Cogitator*, the paradigm proposed in this thesis.

Before proceeding to describe the operation of a typical MBR system, the format of database cases needs to be described.

6.4 Case Representation

Cases are represented by records comprising fields. The fields are partitioned into into collections of *Goal* and *Predictor* fields. To illustrate, an example of a case is given in figure 6.2 below.

Figure 6.2 Example of a case record containing information pertaining to a scuba diver patient who has a burst lung (pulmonary barotrauma) and has any of mediastinal emphysema, air embolism or pheneumothorax.

Goal Fields	*Predictor Fields*	
Pulmonary Barotrauma	Shortness of breath	Pain in chest

The predictor and goal fields may be thought of as containing, respectively, the symptoms and diagnoses describing a case. Using the collection of cases in the database, the MBR system utilises both types of fields to determine possible solutions for a query case.

6.5 Operation of an MBR System

The operation of an MBR system comprises a number of steps :

(a) The system is first presented with a query, a case record comprising fields, not all of which need be instantiated,

(b) For each attribute or instantiated field of the query case, the database records are scanned to determine the similarity between the query case and all the database cases,

(c) A list of best matches is returned, and

(d) the list is used to guide the system through inferencing.

Any of four instances may occur after these operations have been performed :

(a) No database record/case is sufficiently similar to the query case.

(b) Some or a small number of the returned cases are similar to the query case,

(c) All or almost all the returned cases are sufficiently similar or

(d) Some of the returned cases are, while some are not, similar to the query case.

For instance (a), the system may state that it cannot make an accurate diagnosis and state some reasons as to why; it may ask whether the entered information is correct or state that has never encountered such a case and therefore regards the query case as apocryphal. Instance (b) may cause the system to state that the combination of attributes comprising the query case constitute a rare condition or it may present a possible diagnosis but state that it is inconclusive. For instance (c), the system may state that the condition illustrated by the combination of features comprising the query case is familiar and present a fairly conclusive diagnosis . For the final instance,(d), the system may state that there is no definite diagnosis and state the alternatives. It may also ask for more information, proceed with an inconclusive case or propose a precaution rather than a diagnosis. All these proposals are made with respect to the database which assumes the form of the MBR systems' memory or "knowledge".

After a list of cases most similar to the query case have been determined, the system utilises an inference procedure with the aim of creating either an amalgamated case representing the systems' diagnosis, or a collection of possible diagnoses (whether conclusive or not).

Finally, the database is "restricted", either *goal* or predictor wise and the MBR paradigm is re-applied to this restricted database (a subset of the database). In predictor restriction, the predictor fields of the elements of the database are scanned and a similarity computation based thereon is performed for each case. Similarly, in goal restriction, the database is scanned for records containing similar goal field values as those of the current query case. Thereafter, the predictor fields of the records in the restricted database are scanned and similarity coefficients computed. This is done in order to discover plausible (or evocative) new values for the goal field. This may be thought of as a system of refinement.

As may be deduced, an MBR system and a database-driven expert system are essentially synonymous. Inasmuch, the FGP machine [Fertig & Galernter, 1991], a virtual machine describing a database-driven expert system, is also equivalent to an MBR system. The final section compares MBR to the FGP machine as well as Information Retrieval (IR) systems and Deductive Databases.

6.6 Computer Operation

The database of case records is segmented according to the number of processors available in the concurrent computer system and each segment is broadcast to a processor which in turn stores the portion in its memory or disk. The operation of the MBR system then proceeds as follows

(a) The values of the query records' predictors are broadcast to each processor.

(b) Each processor computes a similarity metric value for each predictor,

(c) Using these metrics, a total similarity/dissimilarity metric between each predictor value and each record in each processors' database is computed and

(d) The n best matches (those with the greatest similarity metrics) to the query are selected.

(e) These n best matches are then used for inferencing.

The size of the case database (episodic memory) is significant and it is that which is now considered.

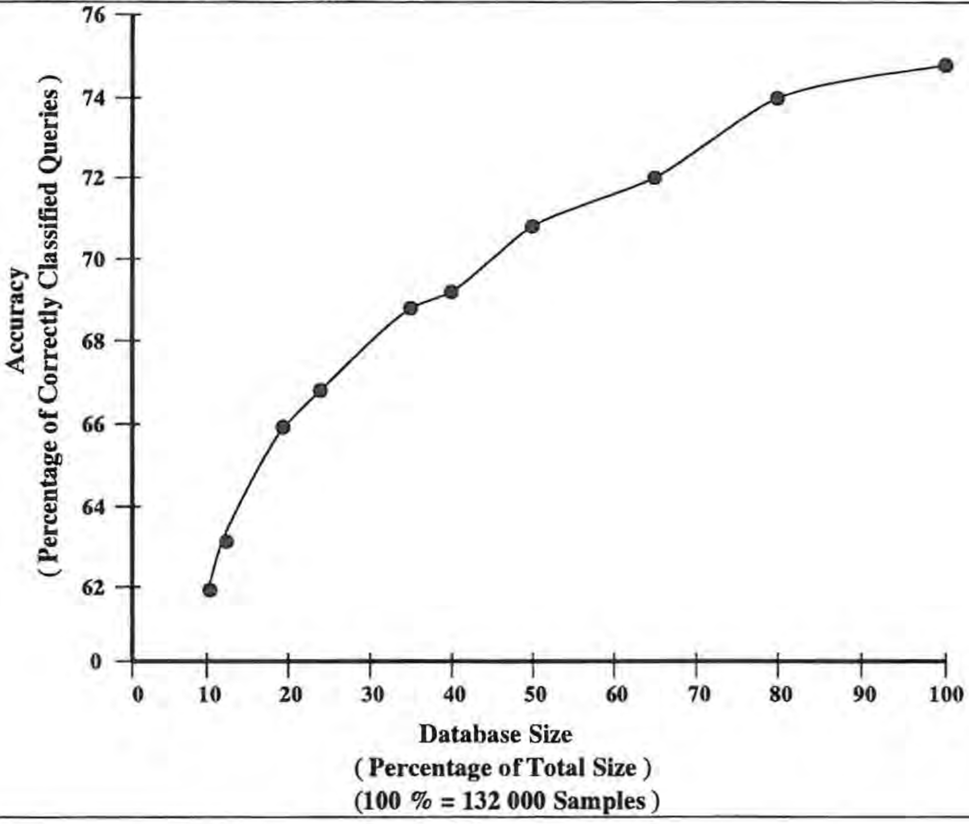
6.7 Database Size

The size of a database used has a significant influence on the accuracy of the systems' conclusions. [Creecy & Waltz, 1991] showed that, generally, the larger the database, the more accurate the deductions are likely to be. [Creecy & Waltz, 1991] also describes how the size of the database was varied and the respective accuracies of the MBR system for each size. Their results are depicted in figure 6.3.

Consequentially, the recognition accuracy of an MBR system is sensitive to the initial database size. Therefore, to improve the systems' accuracy, a simple form of learning (simply adding to the database correct conclusions as irrefutable cases) may be utilised.

However, as databases increase in size, certain problems become apparent including :

Figure 6.3 Accuracy verses Database Size
(Source : [Creecy & Waltz, 1991])



(a) The need for more storage. Not serious, but adds cost.

(b) More processing power. Also, not serious, but adds to cost. Just add more processors (as databases are linearly seperable more processors do not increase the complexity of the MBR system - see MBRs'advantages in the next section).

(c) Greater difficulty in finding and manipulating relevant information (information is more detailed and dense). Serious since it leads to deteriorating search quality (either miss important cases or retrieve irrelevant ones). A solution is to use relevance feedback [Waltz & Thau, 1987] [Waltz, 1986].

Now that MBR has been illustrated, the reader may wonder about its viability as an alternative expert system paradigm. To illustrate this, the next section lists and comments on the advantages and disadvantages of MBR over both symbolic and non-symbolic (connectionist) AI (expert) systems.

6.8 Advantages of MBR

MBR posesses numerous attractive advantages over symbolic and non-symbolic (connectionist) AI systems. Concerning symbolic AI systems (expert systems) these include [1]

(a) *No expert required.* Conferring with a domain expert and having him articulate his knowledge and then transforming this knowledge into a knowledge base takes time. According to [Creecy & Waltz, 1991] the time is often a year or so, sometimes even longer. The reader may recall a quote from [Schank, 1991] in the chapter on expert systems that claimed that building the third version of a system took 30 man years of doctoral students' labour.

(b) *Easier to construct a database than a knowledge base.* A knowledge base is a structured collection of rules and facts. Translating knowledge into this format is non-trivial especially if the domain is fairly large resulting in many rules and facts. Databases are simply collections of previous documented cases and are thus obviously much easier to construct.

(c) *Establishing the consistency of knowledge bases.* The main problem of knowledge bases is in establishing the consistency of the rules and facts therein. This is acknowledged as a major problem, especially when sufficiently large domains are to be characterised. This problem has elicited many researchers to enquire and suggest alternative expert systems to be designed [Stanfill & Kahle, 1986]. Databases do not have a need for consistency establishment at all.

(d) *Databases are numerous and more readily available.* Owing to the effort that is expended in constructing a knowledge base, the compiler(s) thereof are more reluctant to give them to other researchers or other people without some form of reward (financial or referential). Databases are available from many sources, for instance medical and biological databases, and records are also kept by many private and public institutions.

(e) *MBR systems may form a hypothesis based upon a single precedent.* Expert systems or, rather, systems that use rules cannot since rules are generalised summaries of the regularities of phenomena.

[1]Some of the advantages were highlighted in [Creecy & Waltz, 1991].

(f) *MBR systems exhibit consciousness they "know when they don't know".* The use of a similarity metric and additional criteria such as thresholds allow for the embedding of a simple form of consciousness into the system. The system may then use its "consciousness" to determine whether its decisions are conclusive or not and relay this to the user. It may ask for more information, question the veracity of the input or state that its decision is inconclusive and why.

(g) *MBR systems substantiate their conclusions and offer meaningful commentary during operation.* Expert systems cannot generate commentary during execution nor can they substantiate their conclusions. This is a serious limitation since the veracity of an argument or conclusion is based upon how the conclusion was reached (intermediate commentary) and what substantiations are used.

(h) *The implementation time is much shorter than that of expert systems.* As they circumvent the knowledge acquisition bottleneck, they are easier to implement.

(i) *MBR systems are inherently tolerant of noise and deviation.* MBR systems have been shown to be noise tolerant, and to be able to handle great deviations in input [Creecy & Waltz, 1991] [Stanfill & Kahle, 1986]. The deviations in input are also handled by the MBR systems' "*consciousness*" - it may state that the input data is incorrect, ask for more information or state that its decisions are inconclusive. Also, MBR systems degrade gracefully in the presence of noise - as noise increases they degrade sub-linearly until noise reaches about 90% [Creecy & Waltz, 1991]. Traditional expert systems are "brittle" - they tend to fail miserably if presented with a query whose attributes do not match very closely or exactly to those that are contained in the knowledge base. They are therefore intolerant of noise and degrade in the presence of it.

(j) *MBR systems lend themselves to easy implementation on parallel computer systems.* Since databases are completely decomposable, database searches are naturally parallel - segment the database into equal segments and assign each segment to a processor allowing for linear speedups in searches. Expert systems contain many enumerative paths and executing them in parallel implies the propagation of intermediate results/conclusions to all the processors thus leading to plenty of communication overhead. [Blelloch, 1986] each describe a parallel expert system and mention the following problems associated with them : Chapter three described various attempts to parallelise expert systems and illustrates the trickiness associated therewith.

(k) *Since expert systems are generalisations over domains,* they average over many domain cases omitting the details specific to a number of the cases. In certain situations, in fact most, more details about a phenomenon are needed before a decision may be made. Often, when asked for a decision concerning a problem, you might say "Give me more details"? Thus the specific details about a collection (small or large) of cases may be needed to make a decision.

Consequently, expert systems are only really useful for interpreting non-large domains (this is also forced by the need for consistency of the rules : as the no. of rules increase, the determination of consistency becomes increasingly difficult). However, MBR determines those cases most relevant to the current query and in so doing the details specific to all those related cases are considered.

(l) *It is easier to add correctly classified examples to a database than it is to add rules to a knowledge base.* The former operation is simply an append, whereas the latter involves testing for consistency of the rules and deriving them before adding them.

MBR's learning is experiential - when a case is deemed to be correct, it (and its characteristics) is added to the "memory" of the system for possible future reference.

(m) *The inferencing procedure required to reach a conclusion is not repeated when a similar query is posed.* Rather, the closest matches to be found in the database are retrieved instead. This is a more efficient procedure and closely simulates human cognition. It also presents a possible solution to the classic *Categorisation Problem* (see the earlier section on Similarity in this chapter).

(n) *The pervasiveness of Similarity in the natural world lends many real world applications amenable to the MBR paradigm.* MBR has shown encouraging results in modelling real-world systems, much greater than that of traditional expert systems (see the section on applications below for more details).

Of these advantages, the ability to circumvent the "knowledge acquisition bottleneck" (instances (a), (b) and (c)), to generate meaningful commentary and to tolerate noise and inexact input may be considered as being the most attractive.

Concerning non-symbolic or connectionist AI systems, the advantages include

(a) *No training required.* Connectionist systems must be trained with an example set and much time is often consumed for this purpose. The larger the training set, the longer the learning process takes.

(b) *Overtraining.* Connectionist systems also suffer from the possibility of overtraining - when the system learns the example set so well that when presented with inputs not in the example set, recall accuracy is very low. [Weber, 1991] mentions recall percentages around 5% in his time-delay neural network-based voice recognition system.

(c) *Connectionist systems cannot explain their decisions.* MBR systems can and do.

(d) *Connectionist systems are non-interactive.* MBR systems are.

(e) *MBR systems are conscious.* Connectionist systems are not.

(f) *The values representing features are averaged by connectionist systems.* This leads to the loss of detail of collections of phenomena that occur or have similar characteristics. This averaging occurs as the training examples are superimposed upon one another to obtain a connectionist representation of them.

6.9 Disadvantages of MBR

The disadvantages of MBR *vis-a-vis* symbolic AI systems include:

(a) *They require more processing power than traditional expert systems.* The similarity computation must be computed using every element in the database and must be executed for every attribute of the predictor field. Expert systems simply follow pre-determined inference paths.

(b) *Expert systems can generate a proof tree of how their conclusions were reached.* MBR systems cannot - rather, they substantiate their intermediate and final conclusions and produce commentary on their operation.

(c) *MBR is still in the nascent stages of research.* Only two applications [Stanfill & Waltz, 1986] and [Creecy & Waltz, 1991] have been implemented and tested.

Of these, (a) and (c) should diminish over time as, respectively, hardware becomes more powerful and cheaper, and as research in MBR (and related paradigms) progresses. Concerning (c), whether it constitutes a disadvantage may be deemed to be debatable.

Connectionist systems also exhibit some advantages over MBR including:

(a) *Connectionist systems are order-1 search systems.* O(1) search systems are systems that take the same time to search for an element irrespective of the number of elements contained in the search space. MBR systems are O(N) search systems at best - the time to search a database increases linearly with the size of the database.

(b) *Connectionist systems are at a fairly advanced stage of development.* There is a myriad of books on connectionist systems and applications. MBR is still in its incipient stages.

(c) *Connectionist systems have an inherent learning capability.* There is a myriad of books on connectionist systems and applications. MBR is still in its incipient stages.

Instance (a) is a major advantage, although the "forgetting" property of neural networks could be problematical for large databases. (b) is currently an advantage that would diminish with time as more work in MBR and related paradigms progresses.

As may be deduced, whilst connectionist systems exhibit an attractive property (O(1) searching), the advantages of MBR over both symbolic (traditional expert systems) and connectionist AI systems far outweigh its respective disadvantages. MBR is therefore a viable alternative paradigm for the creation of intelligent systems.

6.10 Evaluation Criteria

Research for this thesis has led to the identification of five criteria deemed to be important for the evaluation of the performance of an MBR system, and are :

(a) *Precision* : The proportion of retrieved cases that are relevant to the query case is termed to be precision.

(b) *Recall*: The proportion of relevant documents or cases in the entire database that are deemed to be relevant is termed recall.

(c) *Ease of use*: How quickly a novice / neophyte can learn to use the system and the ease with which an experienced user can use the system.

(d) *Response time*: To facilitate interaction and fast processing of large databases, respectively, good response times and an efficient implementation are needed.

(e) *Ease of Extensibility*: The ease with which the system is ported to process other databases efficiently, and the ease with which records are added to the database.

All the aforementioned five criteria may be used to evaluate the performance of the MBR system. For the proposed system, *Cogitator*, these five criteria are of vital importance to establish its viability.

MBR systems exhibit certain characteristics that may be found in certain other systems, and may sometimes be confused with them. The following section seeks to delineate between MBR and two similar systems, Information Retrieval systems and Deductive Databases, and to illustrate the similarities between MBR and the FGP machine.

6.11 Similarities to Other Systems

The operation of a MBR system is similar to that of the FGP machine [Fertig & Galernter, 1991]. The retrieval of cases from the database is represented in the FGP machine by a *Fetch()* command. The inferencing performed by a MBR system is equivalent to the *Generalise()* operation of the FGP machine, although the methodologies employed may differ. The restriction of a database is equivalent to the FGP machines' *Project()* operation. Thus, MBR and the FGP machine are, in essence, equivalent. However, the difference is that MBR utilises concurrency whereas the FGP Machine does not, but it may easily be extended by defining the following primitive functions:

P_Fetch() - Parallel Fetch
P_Generalise() - Parallel Generalise
P_Project() - Parallel Project

where *P_Fetch()* broadcasts the query case to all the processors/computers being utilised which in turn perform the searching on different portions of the segmented database and return ordered lists of the most similar cases, *P_Generalise()* performs as does *Generalise()*, and *P_Project()* broadcasts the amalgamated or new case to all the processors/computers that are being utilised.

It is important to delineate between *Database-Driven Expert Systems* (DDES's), *Deductive Database Systems* (DDBS's) and Information Retrieval (IR) Systems. Although similar, there are several important differences that need to be illustrated.

A DDBS is a database system that applies logic programming techniques to databases with the aim of extracting inferred or implied information. They may be considered to be an extension of relational ones, in that anything that can be done with the latter may be performed with a DDBS. In addition, a deductive database can of course utilise *rules*, which are usually a subset of (pure) Prolog rules. Queries are similar to those of Prolog. A database-driven expert system is very similar, and possibly the same, depending on aspects such as how rules are expressed [Harland, 1992]. An example of such a system is *ADITI* [Harland, 1992], which is being used to identify and find airline flights between arbitrary cities.

Both *Database-Driven Expert Systems* and *Deductive Databases* have an *inference mechanism.* However, the databases are represented differently. Another difference is that a deductive database is usually conceived of as having a vast number of *facts*, and a comparatively small (though not tiny) number of *rules*(an expert system may have as many *rules* as *data* in some cases), but the database of the DDES does not contain any rules at all.

There are a couple of standard examples of a DDBS. One is family trees. Suppose you have a parent relation, full of tuples such as

```
parent(john, susan).
parent(john, roger).
parent(susan, george).
⋮
```

This is no more than what a relational database can do. However, in a DDB you can also define rules such as the one below, which defines the ancestor relation in terms of the parent one :

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

Queries can then be posed as follows :

```
?- ancestor(X,john)    % "Who are the ancestors of John?"
?- ancestor(john,X)    % "Who are the descendants of John?
?- ancestor(X,Y)    % "Find all tuples in the family tree."
?- ancestor(john,susan). % "Is John an ancestor of Susan?"
```

Another example is a flight information database. Assuming that there is a relation *flight* which has information on direct flights only, we can then write rules such as those below for a trip relation, which finds feasible flight routes between two destinations :

```
trip(X,Y,F) :- flight(X,Y,F).
trip(X,Y,F1.F2) :- feasible(X,S,Y), trip(X,S,F1), flight(S,Y,F2).
```

[Harland, 1992] has a sample application of such a database, implemented in his system - *Aditi.*

Now that the delineation, although fine, between database-driven expert systems and deductive databases has been illustrated, the difference between DDES's and information retrieval systems needs to be illustrsted.

Allied with databases, Information Retrieval (IR) may be deemed to encompass the notions of many modern database management systems such as Oracle. Essentially, IR systems attempt to reproduce the decisions of an expert librarian by studying the information seeking and needs of, respectively, librarians and library users. IR therefore is concerned with the retrieval of appropriate or relevant *documents.*

The classical problem in IR concerns the automatic extraction of relevant textual documents. Attempts to solve this problem have led researchers to propose different models of document representation and their subsequent retrieval. Document models may be classified as one of three types : *exact match*, *probabilistic* or *vector space. Relevance Feedback* is a powerful method of retrieval [Walz, 1990].

The difference between IR and MBR is twofold. Firstly, IR is concerned with the retrieval of relevant *documents* whereas MBR is concerned with the retrieval of relevant *cases.* Secondly, MBR posseses an inference mechanism whereas IR systems do not, the main delineating factor. Nevertheless, cases and documents may be considered to be synonomous so that many retrieval techniques of IR may be utilised in MBR.

IR has established links with AI (cognitive Science and NLP) and database management systems (DBMS) and according to [van Rijsbergen & Agosti, 1992], these links are to strengthen in the future; certainly true if Case-Based Reasoners (CBRs) and their dialects are considered.

6.12 Applications of MBR

Since the FGP machine and MBR systems may be regarded as two different implementations of the same paradigm, their respective applications and the results obtained needs to be mentioned.

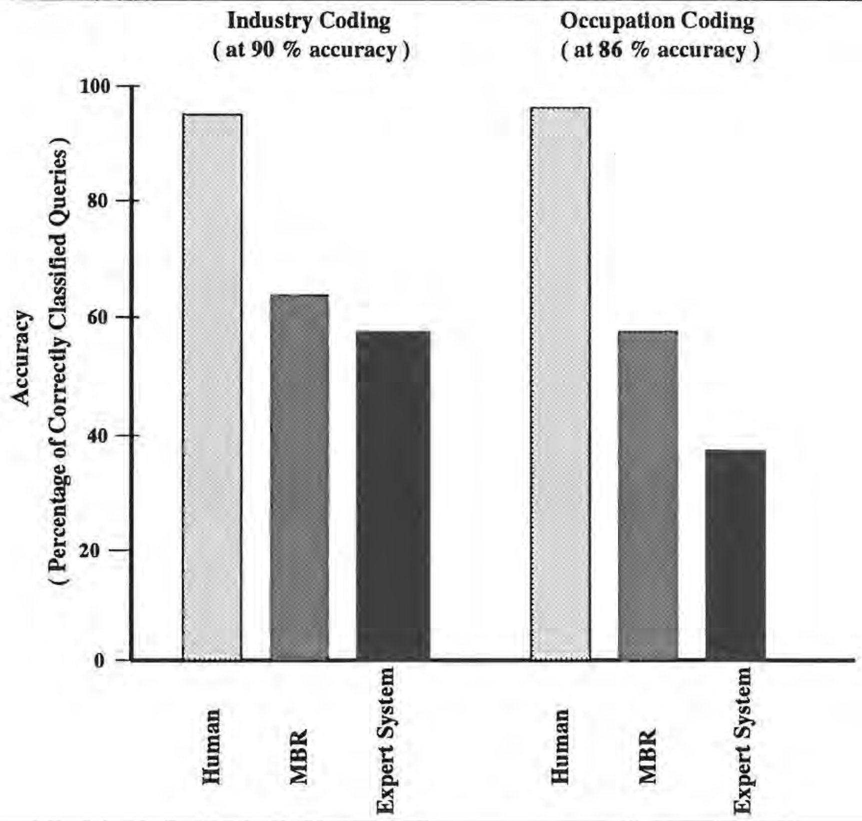
To date, only three applications of the MBR paradigm have been implemented [Stanfill & Waltz, 1986], [Creecy & Waltz, 1991] and [Fertig & Galernter, 1991]. The first, described in [Stanfill & Waltz, 1986], is MBR applied to the pronunciation of English words. The aim was to identify the phonemes and the phoneme accents for each English word (ie. to try to pronounce the word correctly).

Utilising a dictionary of 20,199 words, 4,438 words with a total of 32,768 letters were selected. MBRTalk was then applied to 100 words, totalling 772 letters, and 86 % of the phenomes (43 % of the words) matched the dictionary pronunciation exactly. Humans listening to the output played through a speech synthesizer judged the pronunciation at least as good as that of the dictionary 47 % of the time, slightly and badly mispronounced, respectively, 21 % and 32 % of the time. As English pronunciation is very difficult, [Stanfill & Waltz, 1986] state that these results are encouraging.

The second application, described in [Creecy & Waltz, 1991], concerns using MBR to interpret natural language responses to census forms of the US department of Census. The Censes are conducted annually in order to determine the number of people working in various US industries, i.e. to profile the US workforce. By interpreting a person's responses to the census, the system attempts to classify that person in an industry and occupation category. Those applicants that cannot be processed by the MBR system were referred to clerks for classification.

The performance of the system was evaluated and compared to an expert system, called AIOCS, which was developed to perform the same task. The results are depicted in table 6.4 below, from which it may be easily deduced that the MBR system performed significantly better.

Figure 6.4 Performance comparison of the AIOCS and MBR systems
(Source : [Creecy & Waltz, 1991])



Both these systems were implemented on the Connection Machine (r), a SIMD parallel machine. To date there have been no implementations on other (MIMD) parallel computer systems, as proposed by this thesis.

Three applications of the FGP machine [Fertig & Galernter, 1991] have been implemented including the classification of Irises, attributing folk dances to countries and mammography (breast Cancer). Respectively, the percentages of correct classifications are depicted in figure 6.5 below.

Figure 6.5 Test results of the FGP machine in three domains
(Source: [Fertig & Galernter, 1991])

Class	System	Human
Irises	77%	n/a
Folkdances	57%	53%
Mammography	70%	60%

The reader is referred to [Fertig & Galernter, 1991] for more information.

Other applications of MBR include DNA and protien sequencing [Zhang, 1993] [Core, 1988], document retrieval, Natural Language Processing [Stanfill & Kahle, 1986], and many others. There has even been an application in computer vision: the recognition of objects by using prototypes [basri, 1992]. Continued use and application of MBR to real world problems should indicate its wide applicability and efficacy.

6.13 Conclusion

This chapter sought to illustrate the utility of a CBR dialect, MBR. Initially, CBR was introduced and its close correlation with human cognition, in particular problem-solving, was illustrated. Thereafter, the pervasiveness of similarity in nature was discussed with an aim of further substantiating CBR, and the following discussion on MBR. Certain aspects of MBR were then discussed including database size and operation, whereafter, finally, the advantages and disadvantages of MBR with respect to symbolic and non-symbolic AI systems, was illustrated. The greater number of advantages *vis-a-vis* both systems should convince the reader of the utility and viability of MBR as an alternative expert system paradigm.

The next section illustrates *Cogitator*, the expert system paradigm proposed in this thesis. As *Cogitator* is an extension of MBR, almost all the concepts developed in the current chapter are either utilised or extended upon.

Chapter 7

Cogitator

It has been conclusively shown that traditional expert systems suffer from a variety of problems, some of which are serious. The previous chapter illustrated that MBR circumvents many of them, in particular those allied to the *Knowledge Acquisition Bottleneck* (KAB) indicating that MBR is a viable alternative expert system paradigm. Furthermore, MBR is a naturally concurrent paradigm and the trickiness associated with concurrent traditional expert systems further enhances MBR's appeal.

Towards the end of the previous chapter, the FGP Machine was mentioned and a parallel version thereof was shown to be equivalent to an MBR system. Initially, the use of a parallel version of the FGP machine to define *Cogitator* by the following eight primitive functions from which a concurrent virtual machine defining a concurrent database-driven expert system could be constructed.

- *P_Fetch()* - Parallel Fetch
- *P_Generalise()* - Parallel Generalise
- *P_Project()* - Parallel Project
- *P_Add()* - Add a case to the database
- *P_Delete()* - Remove a particular database case
- *P_SetSize()* - Reset the size of all the databases
- *P_Getsize()* - Determine the sizes of the database segments
- *P_Ossify()* - Determine the irrefutable cases (facts) implicit in the database

However, a parallel version of the FGP Machine is under development at Yale University. Also, *Cogitator* utilises *fuzzy* instead of crisp logic and therefore has a totally different inferencing and similarity computational mechanism. Furthermore, the FGP machine could serve as a system to compare *Cogitator* to. Consequentially, building *Cogitator* directly from the FGP Machine seems inappropriate. Nevertheless, the parallel virtual machine concept is an attractive aspect.

What follows is a description of a particular MBR system, called *Cogitator*, designed to be commercially viable and to offer ease of adaptability and extensibility. *Cogitator* may be considered to be an extension of fuzzy relational databases (FRDBs), in the same way as the FGP machine and MBR are extensions of traditional (non-fuzzy) relational databases (RDBs). For this reason, fuzzy RDBs are initially discussed. Thereafter, a discussion of the types of membership functions that are suitable for *Cogitator* is given, followed by *Cogitator's* proposed analogical inferencing methodology, the suggested representational format of the database, and how a simple form of learning may be incorporated. Finally, the proposed operation of *Cogitator* followed by a sample implementation utilising a popular concurrent programming paradigm, *Linda*, is given.

7.1 Fuzzy Relational Databases

Relational databases, such as Ingres and Oracle, play an important role in the functioning of businesses and corporations today: client records, employee records, etc. However, all these databases contain and deal with information and data *per se*, in that *exact* matching between queries and data must occur. This exact matching places certain limitations on queries and the interpretation of data. For instance, consider the following query

> Which members are in *Considerable* agreement with the president on the effects of fossil fuels?

To characterise the concepts *considerable agreement* and *fossil fuels* (ie. coal, oil and natural gas) in terms of boolean matching queries is difficult. Furthermore, when subjective or imprecise data is being considered, repesenting the data satisfactorily and recalling the correct information is difficult [Buckles, 1982].

For these reasons, researchers have explored the possibility of natural language interfaces as well as fuzzy databases. The author considers a natural language interface to a standard (crisp) database to be artificial. Furthermore, natural language processing is known to be very difficult owing to the panopoly of interpretations of many common statements.

Judging from the literature, most work on fuzzy RDBs was performed during the late 70's and early 80's and is still applicable today. *Fuzzy Databases*, specifically *Fuzzy Relational Databases*, are similar to their crisp counterparts, except that incomplete and imprecise data and queries are handled more efficiently. In effect, they allow [Buckles, 1982]

- Easier specification of incomplete and imprecise queries
- Representation of fuzzy data in a linguistic format
- The similarity relations used to interpret data, and thus the database, may be modified modified to reflect the values of a different individual.

[Buckles, 1982] describes the basic structure of - and the utility of - fuzzy databases when dealing with imprecise or incomplete data. They are useful in areas where subjective knowledge and judgemental evaluation pervade, such as in the economic and medical fields.

Whereas many concepts of fuzzy databases may be extended to Database-Driven Expert Systems (DDESs), they are not equivalent, the two main differences being that

- RDBs, both fuzzy and crisp, seek to eliminate redundancy - something DDESs rely on for noise tolerance and accuracy, and
- RDBs have no inferencing mechanism, they may be considered to be lookup systems (albeit sometimes sophisticated)

Fuzzy RDBs represent the impreciseness of data by *fuzzy sets*. As in traditional RDBs, the database comprises a collection of records each composed of fields. The fields are represented by fuzzy sets, for instance for a field "Salary", records may contain "High" or "Low" instead of, respectively, \$96 000 and \$20 000. In RDB's, records are often termed *tuples*, and fields, *keys*.

For retrieved information, an answer to a query, keys are often represented as vectors, for example

$$\text{Occupation} = < \text{economist, envirnomentalist, economist, geologist} >$$

In essence then, a RDB may be specified as

$$\text{RDB} = (\ t_1, t_2, \ldots, t_n\ )$$

where

$$t_i = (\ d_{i1}, d_{i2}, \ldots, d_{im}\ )$$

the tuples comprising the key field values d_{ij}.

In a Fuzzy RDB, keys are termed *domains* and tuples are sometimes referred to as *relations.* Furthermore, instead of characterising entities or tuples as

$$d_s = \{\ \text{i} \mid \text{i} \in \text{I and i "is about" d}\} \qquad \text{I an index set}$$

they are represented as

$$d_s = \{\ (\text{i}, \mu_d(\text{i})\ ) \mid \text{d} \in \text{D and i} \in \text{I}\ \}$$

where μ_d(i) represents the degree of membership of entity i in domain d (ie. permit the set of index terms to be represented fuzzily) and D is a domain set.

Consequentially, the two main differences are as follows

- $d_{ij} \in D_j$ for a standard RDB
  $d_{ij} \subseteq D_j$ for a fuzzy RDB
- $\forall D_j$, a *similarity relation* [1] S_{ij} is defined.

Therefore, for domain sets D_j, a fuzzy relation R_f may be defined as

$$R_f \subseteq 2^{D_1} \times 2^{D_2} \times \ldots \times 2^{D_n}$$

where 2^{D_i} is the power set of domain D_i.

Membership in R_f is determined by the underlying semantics of the relation.

Definition 7.1.1 A *Fuzzy Tuple* t_i is defined as

$$t_i = (D_{i1}, D_{i2}, \ldots, D_{in})$$

where $D_{ij} \in D_j$ represent *fuzzy labels* of fuzzy set D_j

Definition 7.1.2 An *Interpretation* $(a_1, a_2, \ldots, a_m)$ of tuple $t_i = (D_{i1}, D_{i2}, \ldots, D_{in})$ is an assignment of values a_j such that $a_j \in D_{ij}, \forall j$.

Definition 7.1.3 A *Similarity Threshold* $T(D_j)$, is defined as

[1] A reflexive, symmetric and transitive relation.

$$T(D_j) = \min_i\left(\max_{x,y \in d_{ij}} [S(x,y)]\right)$$

Relational databases are interrogated by some *Relational Algebra.* Fuzzy RDBs are queried by a fuzzy relational algebra which takes the following general form:

(< Operation Name > < one or more Relation names> <one or more Domain names > < Optional Condition > < Minimum Similarity Clause >)

When applied, all relational algebras return a relation that may be used to pose the next query.

For instance,

(Project People over <Height + Age> with < T(age) $\geq$ 0.8 and T(Height) $\geq$ 0.75> Giving Tall-and-Old)

[Buckles, 1982] proves the uniqueness of the union and intersection operations for Fuzzy RDBs. He also mentions two concepts used to measure accuracy: *precision* and *recall* (mentioned in the previous chapter), both fundamental to determining the accuracy of RDB's as well as DDES's.

Before proceeding with a discussion of the operation and functioning of *Cogitator*, a perspective on the types of membership functions, of cardinal importance to any MBR system, needs to be considered.

7.2 Cogitator's Membership Functions

As previously mentioned, the choice of an appropriate membership function for the fuzzy sets and the linguistic variables is of cardinal importance. Consequentially, the question arises: which types of linguistic variable membership functions are suitable for retrieval and/or inferencing?

[Kim & Lee, 1993] provide a possible solution. Arguing that the traditional functions used for information retrieval, AND and OR, based upon T-Norms and Averaging, are inadequate for effective information retrieval, they propose a class of operators called *Positively Compensatory Operators.* A brief outline of their ideas and arguments now follows. For more detailed information, see [Kim & Lee, 1993].

In information retrieval, the logical operators AND, OR and NOT are used as part of the user queries and are evaluated according to certain fuzzy operator functions. The AND and OR functions are of particular importance and are often evaluated, respectively, according to the MIN and MAX operators. [Kim & Lee, 1993] argue that this does not correspond with how people would rank documents. For instance, consider two documents indexed as follows

$$D_1\text{: (kitty 0.35) (ant 0.3)}$$
$$D_2\text{: (kitty 0.95) (ant 0.25)}$$

and the query

$$\text{Q = (kitty AND ant)}$$

As is apparent, applying the MIN operator for AND yields a rating of 0.3 and 0.25 for ,respectively, documents D_1 and D_2, thus rating D_1 higher. However, that is not how a human would rate them.

Before proceeding, a few definitions are in order.

Definition 7.2.1 An operator θ is termed *Single Operator Dependent* if

$$\theta(x, y) = x \text{ or } y$$

θ is termed *Partially Single Operator Dependent* if

- $\theta(x, 0) = \theta(0, x) = 0$
- $\theta(x, 1) = \theta(1, x) = 1 \text{ or } x$

Definition 7.2.2 An operator θ is termed *Negatively Compensatory* if $\forall$ x,y either

- $\theta(x, y) \leq MIN(x, y)$, or
- $\theta(x, y) \geq MAX(x, y)$

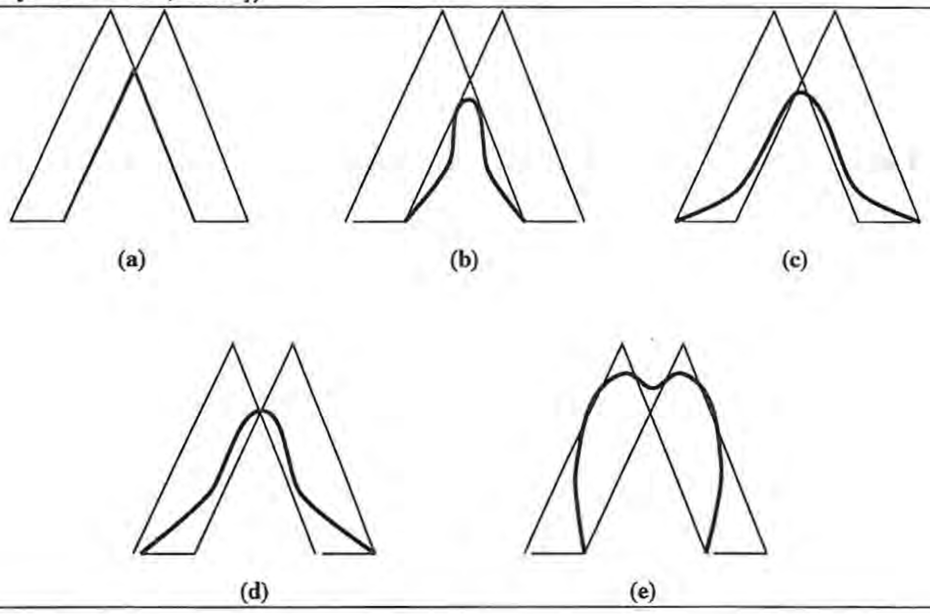
An operator θ is termed *Partially Negatively Compensatory* if $\theta(x, y)$ is *Negatively Compensatory* for some values x,y.

Definition 7.2.3 An operator θ is termed *Positively Compensatory* if

$$MIN(x, y) \leq \theta(x, y) \leq MAX(x, y)$$

and $\theta(x, x) = x$

Figure 7.1 The Various Types of Operator Functions
(Source: [Kim & Lee, 1993])



To illustrate these concepts, consider figure 7.1.

[Kim & Lee, 1993] show that the MAX and MIN operators are *Single Operator Dependent* whereas the T-Norms and T-Conorms are *Negatively Compensatory* and in some cases *partially negatively Compensatory.* [Kim & Lee, 1993] illustrate that negatively compensatory functions depend upon the number of documents and are therefore undesirable as they do not model a human's intuition concerning document ranking.

T-Norms and T-Conorms have also been used to model human, rather fuzzy, decision making by the union and intersection of fuzzy sets. Deemed to be inadequate for this purpose, *Averaging Operators* were suggested in place of T-Norms and T-Conorms.

Positively Compensatory operators avoid all the problems associated with the other four types. Whilst stating that they have not tested these functions yet, they compare positive compensatory operators with those of the *Extended Boolean Model*, an effective information retrieval model, (i.e. E-And and E-Or) concluding that the E-And and E-Or are positively compensatory.

Consider another perspective on the aforementioned operator classes. *Single Operator Dependent* operators appear artificial, rather *extremal*, in that they choose "one or the other", not "weighing up the odds" like people readily do to compensate for ambiguity and uncertainty. *Negatively Compensatory* and *Partially Negative Compensatory* operators behave similarly by over-compensating or under-compensating, in a sense, extremal too.

Positively compensatory operators appear to simulate human cognition more closely by staying within the "extremal bounds" provided by the problem at hand by "weighing up the odds".

For the reasons and discussions mentioned above, the functions to be used for case retrieval and inferencing in *Cogitator* are to be *Positively Compensatory.*

7.3 Cogitator's Inference Mechanism

As has been mentioned, *Cogitator* is an alternative expert systems *paradigm.* Any fuzzy inferencing scheme may be utilised within *Cogitator*, for instance [Nakamura & Iwai, 1982], [Chunxi & Shiquan, 1988] or [Ram & L., 1991]. To illustrate how a fuzzy inferencing scheme might be incorporated into *Cogitator*, consider an adaption of the fuzzy learning scheme introduced in [Nakamura & Iwai, 1982]. Utilising fuzzy topological spaces and semantic networks, they introduce a similarity-based analogical scheme for learning and fuzzy information retrieval. Below is a brief description of how their system functions, the interested reader is referred to [Nakamura & Iwai, 1982] for more details.

Similarity is modelled utilising a semantic network wherein two types of nodes exist to represent the concepts x_i and their properties p_k. Each concept is linked to its properties as well as to certain other concepts, where the inter-concept links are weighted whereas the links to the properties are not. These weights are considered to represent the *degree* of similarity or dissimilarity between concepts. Furthermore, there is the notion of *direct* and *indirect* properties. Properties immediately adjacent to a concept x_i are termed *direct* properties, those adjacent from any concept that may be reached from concept x_i's node are termed *indirect* properties. Indirect properties are classified according to the number of edges that must be traversed to reach them from the concept x_i, n edges traversed to reach property p_r classifies p_r as an *n-reachable* property.

Utilising these notions, [Nakamura & Iwai, 1982] define the following similarity metric

$$S(x_i, x_j) = \frac{\sum_n \lambda^{n-1} N_n(p_i \cap p_j)}{\sum_n \lambda^{n-1} N_n(p_i \cup p_j)} \quad 0 \leq \lambda \leq 1$$

where $N_n(p)$ represents the cardinality of the set of n-reachable properties from p.

Together with this similarity metric, the semantic network forms a topological metric space: if two concepts are very similar, they are placed close together in the topological space. Thus the related concepts form a sub-region of the topological space.

Inference is analogical, is characterised by the use of fuzzy sets and is based upon a question-answering scheme wherein the computer poses questions for the human user to answer. The system then utilises these answers to reason, possibly posing further questions in its quest for a conclusion. The operation of such a system is as follows:

- [1] Assume the Fuzzification functions, $f_{x_i}(x)$ and $\overline{f_{x_i}(x)}$, respectively, defining the fuzzy sets $F_{x_i}(x)$ and $\overline{F_{x_i}(x)}$. Additionally, assume a measure of fuzziness $(D(\tilde{C}_k, \tilde{\tilde{C}}_k)$, where $\tilde{C}_k$ and $\tilde{\tilde{C}}_k$ are, respectively, the k'th concept set and some reference set.
- [2] Prompt the user for initial information - this becomes the initial concept set $\tilde{C}_0$.
- [3] Based upon the information received and inferred to date, the next question concept is chosen according to a *selection criterion* $E(I(C_k))$ which is defined as follows:

  $$Min[E(I(C_k))] = Min_q[\mathcal{F}(\tilde{C_{k-1}}, F_{x_q})]$$

  where

  - $E(I(C_k))$ is the *Expectation of Fuzziness*
  - $I(C_k) = \mathcal{F}(D(\tilde{C}_k, \tilde{\tilde{C}}_k)$ is the *Index of Fuzziness*
  - $\tilde{C_{k-1}}$ is the (k-1)'th concept set
  - x_q is a possible k'th question concept
- [4] The system poses a question concerning the chosen x_q to the user
- [5] If user answers "Yes", form the fuzzy set $\tilde{C}_1 = C_0 \cup F_{x_q}$
  If user answers "No", form the fuzzy set $\tilde{C}_1 = C_0 \cap \overline{F_{x_q}}$
  Otherwise, do nothing.
- [6] Return to step 3. (See [7] below for the termination condition)
  Eventually

  $$\tilde{C}_k = (\bigcup_{x_y} F_{x_y}) \cap (\bigcap_{x_n} \overline{F_{x_n}})^2$$

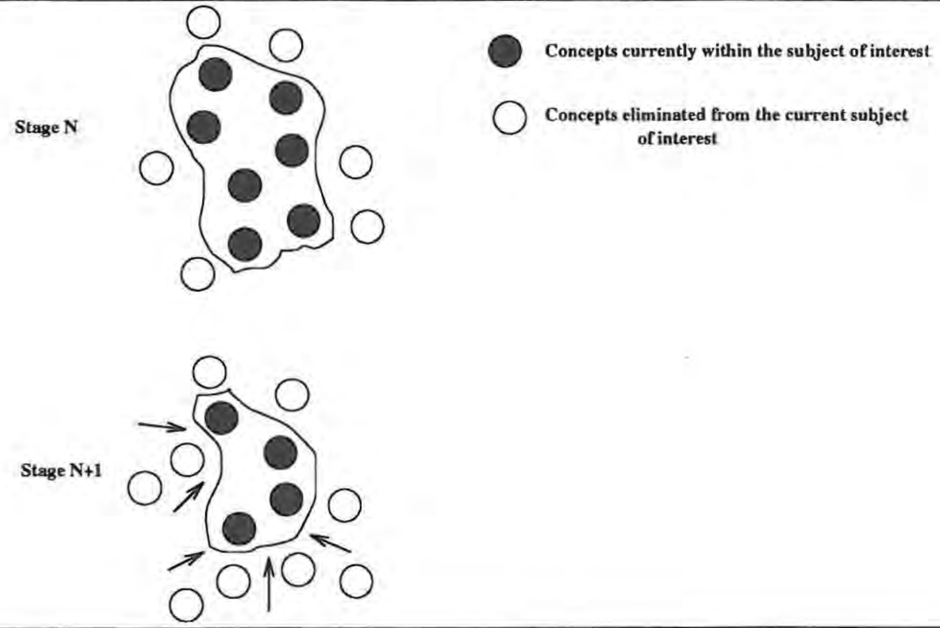
²Owing to the associativity and commutativity of the fuzzy union and intersection operators, the order of question-asking and answering is immaterial.

where x_y and x_n are, respectively, all the concepts that when posed received the answers "yes" and "no".

- [7] Inferencing terminates when the index of fuzziness $I(C_k))$ drops below a certain threshold value τ indicating that the inter- conceptual ambiguity is minimal. This is deemed to be synonomous to people understanding something when their cognitive ambiguity has diminished sufficiently.

The main idea behind such an inferencing process is that the regions, rather the topologies, defined by the fuzzy sets $F_{x_i}(x)$ and $\overline{F_{x_i}(x)}$ are repeatedly deformed according to the concept combination functions given in stage [5] above to yield a space C representing a *coherent* collection of concepts. Each step of the above process generally improves the coherency of the subject of interest.

Figure 7.2 The Incremental Deformation of Regions in K_c


Concepts currently within the subject of interest
Concepts eliminated from the current subject of interest
Stage N
Stage N+1

This is illustrated in FigureF 7.2 and is codified as follows:

- When a concept x_i is specified to be contained in the subject of interest and each concept x in K_c is assumed not to be contained therein, the fuzzification function $f_{x_i}(x)$ *raises* the degree of membership of each concept x according to the distance they are from x_i (i.e. their similarity).

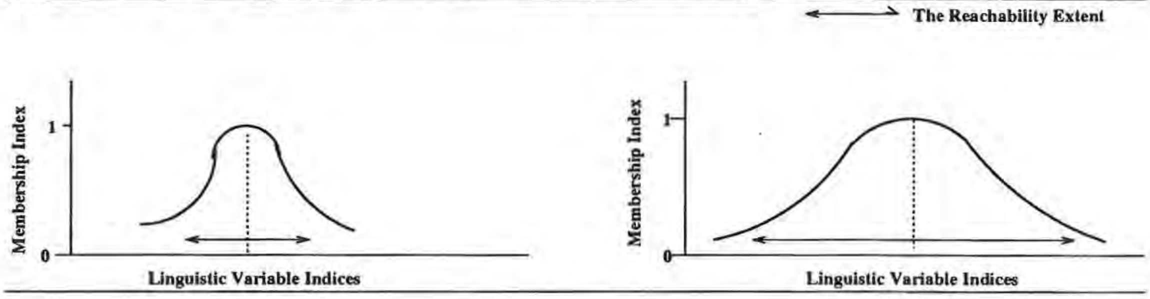
- When a concept x_i is specified *not* to be contained in the subject of interest and each concept x in K_c is assumed to be contained therein, the fuzzification function

$f_{x_i}(x)$ *reduces* the degree of membership of each concept x according to the distance they are from x_i (i.e. their similarity).

The *Convergence* of the inference process to this cogent collection is largely determined by the type of information in the database and the type of query posed : if the database contains a collection of information that is minimally contradictory then convergence is quick and efficient.

The coherency of a region representing a body of knowledge may be measured according to any *measure of fuzziness*, such as the difference between a fuzzy set and its complement, between a fuzzy set and the closest crisp set (used by [Nakamura & Iwai, 1982]) or by a fuzzy entropy. [Nakamura & Iwai, 1982] also considered what they termed the "reachability extent of analogical inference" as a parameter of $f_{x_i}(x)$ and $\overline{f_{x_i}(x)}$ that determines the spread of the function. Figure 7.3 illustrates this notion.

Figure 7.3 The Reachable Extent of a Fuzzy Membership Function



[Nakamura & Iwai, 1982] also analysed these "reachable extents" and concluded that larger extents lead to quick, yet rough learning, whereas smaller extents lead to slower yet more defined learnt regions of knowledge. This finding corresponds closely with human cognition and is utilised in *Cogitator*.

The coherency of regions or inferred bodies of knowledge is significant. The greater the coherency of a subject, the easier it is learnt and within which conclusions are reached (as it is essentially represented by a single region in K_c). Less coherent subjects of interest tend to be represented by disjoint regions of K_c, indicating that there is a need for a form of reconciliation between such unrelated bodies of knowledge. This also corresponds closely with human cognition.

The reader may have noticed the ommission of functions for the measures, indices and expectation of fuzziness and the similarity functions. There are a multitude of such functions, some of which were given in chapter 4 on fuzzy logic. The choice of function rests with the system developer, some types or classes of functions may exhibit certain properties that may be desirable for problems in certain domains.

7.4 Database Representation

Both Wong's Generalised Vector Space Model (GVSM) [Wong & Wong, 1987] and semantic networks were considered as possible representation models for the database. The simplicity of a standard textual record-based database and the complete decomposability thereof prompted its choice above semantic networks and the GVSM: by their nature, semantic networks do not partition easily and utilising the GVSM to represent fuzzy concepts is non-trivial. Nevertheless, the GVSM does partition fairly simply and a fuzzy extension to it could be a possible extension of the current project.

Furthermore, methods for the indexing of databases, such as signature files or hashing, may be implemented to improve the efficiency and storage of and access to the database. If speed is a crucial component (which it usually is for database searches), then indexing the database is natural.

7.5 Learning

Learning is an integral part of an intelligent system. Traditionally, it has operated within the deductive / heuristic paradigm and involves the generation of rules. As in MBR, learning in *Cogitator* does not involve the generation of any rules. This is a desirable aspect since rule generation consumes plenty of time owing to the myriad of rules that are possible for even a moderately sized database and is non-trivial (consistency and redundancy checking, etc).

In *Cogitator*, learning is simple as it entails the mere appending of a case to the database. Some cases may be stored as facts for future reference, facts are deemed to be irrefutably correct cases. These facts are derived from irrefutable conclusions reached previously. Learning is therefore incremental: as the system is used, the database grows and the accumulation of cases and facts should improve the system's accuracy as it is used. A very simple process.

7.6 Design and Operation

Now that the type of membership functions, an inferenceing methodology and the representation of the database have been discussed, how these all fit together to form a DDES is considered.

Cogitator is a *general* scheme describing a fuzzy database-driven expert system. As mentioned previously, the choice of fuzzy measures, indices and expectations of fuzziness as well as similarity equations is the whim of the system developer. In fact the user may even choose a different type of inferencing scheme - the one illustrated above appeals to the author owing to its close resemblance to human cognition. It is easily adapted for execution on any of the major concurrent architectures: SIMD, MIMD and hybrids, and may thus be considered to be architecture independent. Almost all universities and corporations own networked computer systems and the computers therein may be considered to collectively form a concurrent computer, a MIMD-type architecture (ie. distributed computation/concurrency). Consequentially, the development of *Cogitator* to execute on this type of architecture enhances its appeal. This chapter contains a description of such a (possible) implementation, initially illustrating its functionality and then how it could be implemented using a popular concurrent programming paradigm. The following section illustrates a sample implementation of *Cogitator* over a network of workstations.

The operation of *Cogitator* is as follows. Upon startup, the database, which normally resides on a certain machine, is accessed and partitioned according to the number of computers (workers) in the network. Each segment is then sent to each member computer and becomes resident on that machine. After some initialisation procedures, the system is ready. The user then poses a query which is passed on by the host computer to all the other computers (workers) in the network. Upon receipt of the query case, each worker initiates a search procedure to determine a list of cases from its resident database segment that most closely matches the query case. In effect then, the database search (back-end) extracts a collection of elements most similar to the query case from the database's vast collection. These are then passed on to the host machine (front-end) which utilises the inferencing mechanism to hone the collection of returned cases down to a more cogent or consistent collection of concepts. *Cogitator* may also pose questions if it requires more information or is "unsure" and utilise the users' answers for the furtherance of inferencing. The system terminates according to the conditions as laid out in the inferencing procedure described earlier.

Now that *Cogitator* has been completely discussed, an example implementation of *Cogitator* over a computer network using a concurrent programming paradigm is appropriate. It is hoped that the next section illustrates to the user a new concurrent programming paradigm, as well as how easily *Cogitator* may be implemented on a concurrent architecture.

7.7 Proposed Implementation using Linda

As was mentioned in the chapter on concurrency, programming concurrent computers is far from easy, far more difficult than programming their uni-processor cousins thus complicating matters and exacerbating the familiar *Software Crisis*. For these reasons, different concurrent or parallel programming paradigms have been developed, many for specific architectures, others for more general ones. The difficulties associated with programming concurrent computers and that when switching architectures, the need to learn a new parallel programming paradigm is necessary. Therefore, architecture-independent paradigms have become popular. Amoung these architecture-independent paradigms, one, called *Linda* and developed by David Galernter of Yale University during the early 80's, appears to be the most popular. It has been used for many applications such as database searching, computational fluid dynamics (CFD) and molecular modelling to name a few. It's success may be attributed to its simplicity and its associated programming ease. What follows is a brief description of *Linda*, whereafter the design of *Cogitator* is given within *Linda*'s framework. The interested reader is referred to [Carriero & Galernter, 1989], [Carriero & Galernter, 1988] and [Ahuja & Galernter, 1986] for further details.

7.7.1 Linda

Linda is a *Distributed Data* parallel programming system. In fact, *Linda* is a *Coordination Language* comprising a set of five functions that are embedded within a traditional base programming language such as *C* or *Fortran*. Quintessential to its design is the notion of a *Tuple Space*, an associative shared memory containing data elements called *Tuples*. The five functions are

- in(τ) : reads a tuple τ from tuple space and removes it from Tuple Space (TS).
- rd(τ) : reads a tuple τ from tuple space without removing it from TS.

- out(τ) : places a tuple τ into TS.
- rdp(τ) : the predicate form of rd(); checks to see if a tuple matching τ. is present in TS and returns a value indicating whether the tuple is present.
- inp(τ) : the predicate form of in(); the same as rdp() except that if a tuple is found, it is removed from TS and returned.

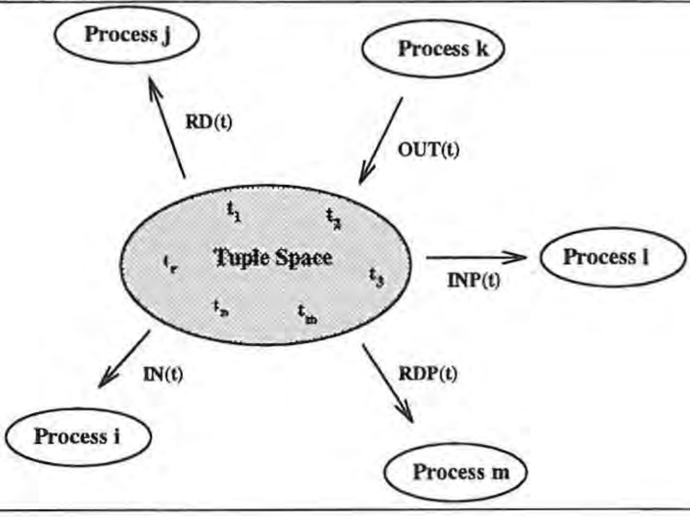
In addition there is another function called *eval()* which evaluates a tuple τ *after* placing it into TS (i.e. by forking another process to do so).

The tuples τ as the arguments to the five functions above (excluding *eval()*) correspond to *templates.* For example the tuples

- out("string", s)
- rd("string", s)
- out("matrix", rows, columns, M)

respectively, place a tuple with template ("string", s) into TS, reads a tuple with the same template as ("string", s) from TS and places a matrix tuple corresponding to the template ("matrix", rows, columns, M) into TS. Figure 7.4 illustrates a general schematic of *Linda.*

Figure 7.4 The Topology of Linda



As may be deduced from figure 7.4, *Linda* is essentially a *master-slave* architecture, although other architectures may be easily accomodated and a master may become a worker.

When a rd(τ) or an in(τ) is initiated, and there is no tuple corresponding to tuple τ in TS, they wait until one is available (i.e they block). However, rdp(τ) and inp(τ) do not block if no matching tuple τ is found, rather they return a code indicating abscence of a tuple or they read the tuple if present.

Two tuples match only if the following conditions are satisfied

- They contain the same number of fields.
- All corresponding fields are of the same type.
- All corresponding fields are of the same size.
- The corresponding fields in the tuple contain the same values as the actuals in the template.

The *blocking* features of IN() and RD() are important and are used for synchronisation of processes; in fact, tuple space is used as an inter-process communication medium. A common practice is for the first field of a tuple to be a string constant, such as "string" and "matrix" for the tuples above. This is termed *tuple naming* and is utilised for easier debugging and writing of *Linda* programs.

There are two types of tuples, *Live* and *Data* tuples. Data tuples are simple collections of fields in a template, such as ("matrix", rows, columns, M). Live tuples represent an executable process that, upon termination, becomes a data tuple. When eval() is called, it creates live tuples that may execute on different processors. For example

eval("Power", 3, 5, power(3,5))

causes the function *power()* to be evaluated after it is placed into TS (i.e. *Linda* spawns another process on an arbitrary host in the network to evaluate the aforementioned tuple). See the example below as well.

To illustrate the concepts outlined above, consider, the following C program.

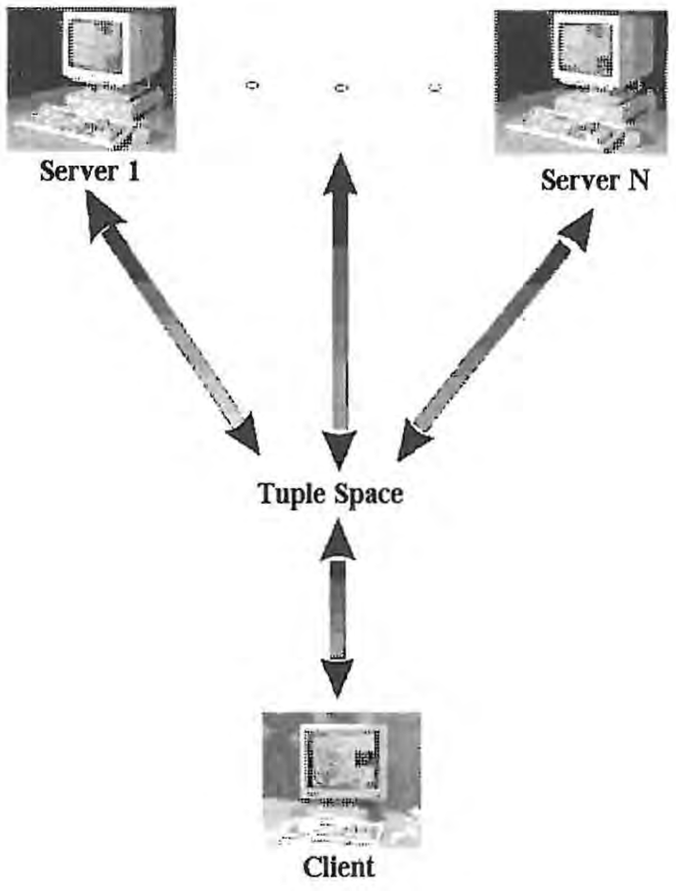
```
real_main(argc, argv)
int argc;
char *argv[];
{
 int nworker=5, j, hello();    /* Declaration of variables and the function */

 for (j=0; j < nworker; j++)
 {
  eval("worker", hello(j));       /* place a "live" tuple (the function */
                                  /* "hello()" ) into tuple space and   */
                                  /* have it evaluated                  */
 }
 for (j=0; j< nworker; j++)
 {
  in("done");                      /* All the processes are finished? */
  in("worker", ?retval);           /* In the values returned from the */
                                   /* terminated worker processes     */
}
 printf("All processes have terminated");
}

int hello(i)                      /* function to print out the result */
                                  /* of executing on a certain host   */
int i;
{
char name[25];

 gethostname(name,25);             /* get the name of the host the */
                                   /* process is executing on      */

 printf("Hello from host number %d host name %s \n", i, name);
                                                  /* print the result */
 out("done");                /* Place a tuple in tuple space to */
                             /* indicate completion of the task */
 return(0);
}
```

The above program initiates worker processes on each of five machines in a network. To accomplish this, it utilises eval() which creates a live tuple out of the function *hello()* that retrieves the name of the host it is executing on, prints the result and places a tuple into tuple space to indicate that it is finished. These values are then read by the master process (the function real_main()), and when all have been sent, it prints a message to indicate that the program has terminated.

7.7.2 Cogitator's Design and Functionality

Being a master-slave architecture, the functionality of *Cogitator* is easily expressible within *Linda*'s framework. Refer to figure 2.

Figure 2 The Topology of *Cogitator* using *Linda*



Upon startup, the segmentation of the database according to the number of hosts that may be utilised, the copying to each host of a database segment (via the remote copy facility) and the accepting of an initial query from the user are codified as follows (assuming that the operating system is UNIX ©) :

```
No_of_Hosts=atoi(*argv[1]);  /* The number of hosts - an argument to the program */

segment(DB, DBSegments);     /* Partition the database into segments according */
                             /* to the number of computers to be utilised      */

for (count=0; count<No_of_Hosts; count++)
{
 command = "rcp ./";
 host=*argv[count+1];                  /* Get the host name */
 strcat(command,DBSegment[count]);   /*  Build up the remote copy command */
 strcat(command, host);
 strcat(command,":/tmp");

 system(str);                /* issue the system command to copy the     */
                             /* database segment to the appropriate host */
}
Initialise_Master();            /* Initialise the master host */
printf("Please enter query :");    /* Prompt for and read */
getstring(query);                  /* the initial query   */
```

The user starts the system by entering a query which the master translates into a tuple such as

("Query", Q)

where Q is an array of strings or a long string; and places the tuple into TS :

out("Query", Q)

Meanwhile, each worker is waiting, rather *blocking*, on a rd():

rd("Query", ?Q)

When the query arrives in TS, the workers immediately read it and perform fuzzy similarity matching [3] between the query tuple and *all* the elements of their respective resident database segments, building a list ordered according to the computed similarity coefficients. After all entries in the resident database have been checked, the worker translates each member of the ordered list that satisfy a threshold value and/or a threshold

[3] Assuming that the fuzzy sets representing the coded database domains have been defined.

number of elements into tuples and places them into TS. After placing all the elements into TS, it places a tuple in TS to indicate that it is finished. This may be codified as follows:

```
in("Query", Q);              /* read in the query tuple */
entry = 1;

while (1)
{
 finished = GetNextEntry(DB, Record[entry]);     /* retrieve the next database record */
 If (!finished)
 {
  FuzValue[entry] = Fuzmatch(Q, Record[entry]);
                                        /* Determine the fuzzy         */
                                        /* similarity between the      */
                                        /* query and the database entry */
  if (FuzValue[entry] > FuzValue[entry-1])
  {
   Swap(FuzValue[entry], FuzValue[entry-1]);  /* Sort the fuzzy similarity    */
                                              /* values into descending order */
   Swap(Record[entry], Record[entry-1]);       /* Sort the records into */
                                               /* into descending order */
  }
 }
 else
  break;               /* end of database - stop searching */
}             /* while loop delimiter */

  entry = 1;
  gethostname(HostName,25);                   /* Determine the name of the */
                                              /* current workers' host     */
  while ( (entry < Threshold) || (FuzValue[entry] < Threshold_Value))

  {
   out("worker", HostName, entry, FuzValue[entry], Record[entry]);
                                        /* Place all returned entries and  */
                                        /* their fuzzy similarity          */
                                        /* coefficients into tuple space   */
  }
  out("Done", HostName);        /* Send a completion signal to the master */
```

While the workers perform their tasks, the master waits for a worker to finish; i.e. to place its tuple indicating completion into tuple space. When this tuple is detected, the master records the host it emanated from and in()'s all the list tuples that that worker placed into TS. This is accomplished as follows:

```
HostCount = 0;
while (HostCount<No_of_Hosts)
{
 in("Done", ?HostName);                    /* Find a completed worker */
 HostCount++;

 counter=1;
 Entry_Number=1;
 while (counter<=Entry_Number)
 {
  in("Result", HostName, counter, ?Record[counter]);
                                  /* Read in the data returned */
                                  /* by the completed worker   */
 }
}
```

After reading in all the tuples returned by all the workers, the master then constructs an ordered amalgamation of the workers' ordered lists, yielding a single ordered list of matching elements. This amalgamated list is constructed according to certain thresholds and almost certainly contains fewer elements than were collectively returned by all the workers. This is a simple merge sort. This resulting amalgamated ordered list is used by *Cogitators* inference mechanism [4] to hone in on a coherent collection. While the searching and matching were being performed concurrently on the database segments in the back-end of *Cogitator*, the front-end performs the inference sequentially.

7.8 Conclusion

It is hoped that a powerful, yet simple general reasoning scheme that utilises the cognitive simulation power of fuzzy logic, along with the utility of databases, which are numerous,

[4]Any of the numerous fuzzy inference methodologies may be used.

as well as the power of concurrency has been illustrated. Concerning the little work performed by others as well as the numerous advantages over and few disadvantages *vis-a-vis* traditional expert systems and connectionist systems, *Cogitator* may be deemed to be viable.

As is common in AI, it is one thing to propose a system, but another to implement and test it. However as was intimated in the chapter on expert systems, AI systems of any repute and size almost always require massive software engineering. While a sample implementation was given, this thesis represents the first phase; the second phase - implementation - is to be undertaken by the author to establish its viability as an alternative expert system paradigm.

Chapter 8

Conclusion

The necessity of concurrency for any realistic speedup was categorically established. Furthermore, as knowledge-based systems are known to be space and time hogs, it seems natural to build concurrent expert systems to speed up their execution and ease memory problems. However, as was illustrated, the concurrentising of expert systems is no simple task: not do they also suffer from the main problems associated with their sequential counterparts, but other problems like convergence and compatibility, the need to write "correct" programs or to assign appropriate priorities to rules so as to preclude conflicts became apparent. Attempts to overcome these problems led to restrictions on the system or programmer or no guarrantee of a correct terminal state. Furthermore, the speedups gained, were not all that impressive - for more than about 8 processors, the speedups tended to level off indicating that the production systems paradigm exhibits limited concurrency. It was also shown that AI systems of any decent nature would be very large indeed. Consequentially, more power than that offered by simply concurrentising the production system paradigm will be needed. To answer this, concurrent non-production systems have been built and tested. They offered the sort of speedups that will be needed for the processing of large knowledge bases. However, they are custom architectures, so that they may rerally only be used for certain AI applications only and their cost would be high. Not viable in today's commercial world.

To answer the problems associated with sequential and concurrent production systems, as well as concurrent non-production expert systems, a new expert system paradigm, called *Cogitator*, was proposed. Based upon a Case-Based Reasoning Dialect, *Memory-Based Reasoning*, utilising fuzzy analogical inferencing schemes as well as databases, the system was shown to bypass almost all the problems associated with connectionist and traditional expert systems. That, in both cases, the advantages of *Cogitator* far outweighed its disadvantages illustrates its viability as an expert system. In particular, its circumvention of the *Knowledge Acquisition Bottleneck* and the resulting shorter construction times illustrates its commercial viability. Its commercial viability is further enhanced by its operation on a network of workstations, which many companies posess.

Hiroaki Kitano is a researcher using *Memory-Based Reasoning* on a supercomputer (a Connection Machine ©) to tackle the language translation problem. In recognition of his results and efforts, he was awarded the most prestigious award in AI for researchers under 35: the *Computers and Thought Award* at the 13'th International Joint Conference on AI (IJCAI) in Chambery, France, on November 12'th 1993. As *Cogitator* is a fuzzy version of the Memory-Based Reasoners that Kitano utilises, and as fuzzy mathematics supersedes classical mathematical techniques for human cognitive modelling (Mentioned in chapters 4 and 5), further strengthens *Cogitator's* viability as an alternative expert systems paradigm [Margolin, 1993].

To date, the system has been proposed and compared to traditional expert systems. Linda is deemed to be an appropriate implementation platform and *Cogitator's* implemention on a network of workstations has been considered and partially specified. To implement *Cogitator* and to run a series of tests to ascertain its viability as a commercial expert system, is needed. As was already mentioned, that is the next phase of this project which the author intends to pursue.

References

AHUJA, S. CARRIERO, N., & GALERNTER, D. 1986. Domestic parallelism : Linda and friends. *Computer (usa)*, **19**(8), 26–34.

ALMASI, G, & GOTTLIEB, A. 1989. Highly parallel computing. *Benjamin cummings.*

AMDAHL, A. 1967. The validity of the single processor approach to achieving large scale computing capabilities. *Afips conference proceedings*, **30**, 483–485.

BAISE, P. 1993. The fifth and sixth generation projects. *Technical document 93/08, dept. of computer science, rhodes university*, April.

BASRI, R. 1992. Recognition by prototypes. *Mit ai memo no. 1391*, December.

BLELLOCH, G. 1986. Cis : A massively concurrent rule-based system. *Technical report rl86-2, thinking machines corp.*

BUCKLES, B. 1982. Fuzzy databases and their applications. *Fuzzy information and decision processes*, 361–371.

BUCKLEY, J. 1992. Equivalences between nets, controllers, expert systems and processes. *Proceedings of the 1992 fuzzy theory and technology conference.*

CARRIERO, N., & GALERNTER, D. 1988. Applications experience with linda. *Acm sigplan notices*, **23**(9), 173–187.

CARRIERO, N., & GALERNTER, D. 1989. How to write parallel programs: A guide to the perplexed. *Acm computing surveys*, **21**(3), 323–356.

CHUNXI, X., & SHIQUAN, C. 1988. Fuzzy analogical reasoning and its application. *Fuzzy logic in knowledge-based systems, decision and control*, 337–348.

CORE, N. EDMISTON, E. SALTZ J. SMITH R. 1988. Parallel processing of biological sequence comparison algorithms. *Yale university technical report yale/dcs/rr-630*, July.

CREECY, R. MASAND, B. SMITH S., & WALTZ, D. 1991. Trading mips for knowledge engineering: Automatic classification of census returns on a massively parallel supercomputerassiv. *Technical report tmc-192, thinking machines corp.*, April.

CUNNINGHAM, H., & ROMAN, G. 1990. A unity-style programming logic for shared dataspace programs. *Ieee trans. parallel and distributed systems*, July.

DESBIENS, J. GAGNE, D. JANTA-POLCZYNSKI M., & NAULT, G. 1992. Simulation of crew interaction in a military aircraft via a multi-agent system. *Internal paper - available from nault@cmr.ca.*

DESROGERS, G. 1987. Principles of parallel and multiprocessing. *Mcgraw-hill.*

DOUGLAS, R, & MAHOWALD, M. 1992. Silicon neurons. *Byte magazine*, October.

ENSLOW, P. 1977. Multiporcessor organisation. *Acm computing surveys.*

ERMAN, L. HAYES-ROTH, F. LESSER V., & RAJ REDDY, D. 1980. The hearsay-ii speech recognition system: Integrating knowledge to resolve uncertainty. *Acm computing surveys*, **12**(2).

FERTIG, S., & GALERNTER, D. 1991. The design, implementation and performance of a database-driven expert system. *Technical report yaleu/dcs/tr-851, department of computer science, yale university*, February.

FLYNN, M. 1966. Some computer organisations and their effectiveness. *Ieee trans. on computers*, **21**(9), 948–960.

FORGY, C. 1982. Rete: A fast algorithm for many pattern/many object pattern match problem. *Artificial intelligence*, **19**(1), 17–37.

GALLANT, S. 1988. Connectionist expert systems. *Communications of the acm*, **31**(2).

GROSSMANN, M. 1992. Modelling reality. *Ieee spectrum*, September, 56–60.

HARLAND, J. 1992. Personal correspondence with author. September 10.

HERATH, J. 1992. Computer architectures for intelligent systems. *Ieee computer*, May, 6–9.

HIGHLAND, F., & IWASKIW, C. 1989. Knowledge base compilation. *Proc. 11 th joint conference on ai, ijcai-89*, 78–83.

HIGUCHI, T. KITANO, H. FURUYA T. HANDA K. TAKAHASHI N., & KOKUBU, A. 1991. Ixm2: A parallel associative processor for knowledge processing. *Proc. 9 th national conference on ai, aaai-91*, 296–303.

HOCKNEY, C., & JESSHOPE, P. 1981. Parallel computers. *Adam hilger ltd*, 5–7.

HOFSTADTER, D. 1980. Gödel, escher, bach : An eternal golden braid. *Penguin books.*

IEEE. 1992. How dec developed alpha. *Ieee spectrum*, July, 26–31.

ISHIDA, T. 1991. Parallel rule firing in production systems. *Ieee trans. on knowledge and data eng.*, **3**(1), 11–17.

KIM, M. LEE, J, & LEE, Y. 1993. Analysis of fuzzy operators for high quality information retrieval. *Information processing letters*, **46**(5), 251–256.

KLIR, G, & FOLGER, T. 1988. Fuzzy sets, uncertainty and information. *Wiley.*

KOLODNER, J. 1991. Improving human decision making through case-based decision aiding. *Ai magazine*, Spring, 52–68.

KOSKO, B. 1992. Neural networks and fuzzy systems. *Prentice hall inc.*

KUO, S., & MOLDOVAN, D. 1992. The state of the art in parallel production systems. *Journal of parallel and distributed computing*, **15**, 1–26.

LENAT, D. GUHA, R. PITTMAN K. PRATT D., & SHEPERD, M. 1990. Cyc: Toward programs with common sense. *Communications of the acm*, **33**(8).

MARGOLIN, B. 1993. Psc researcher wins ai award for machine translation program. *Posting to the internet*, November.

MICROELECTRONICS, & CORPORATION, COMPUTER. 1991. The cyc information sheet. *Internal report, microelectronics and computer corporation.*

MIRANKER, D. KUO, C., & BROWNE, J. 1990. Parallelising compilation of rule-based systems. proc. 1990. *International conference on parallel processing*, II, 247–251.

MIRANKER, D. 1987. Treat: A new and efficient algorithm for ai production systems. *Phd thesis, dept. of computer science, columbia university.*

MOLDOVAN, D. 1989. Rubic: A multiprocessor for rule-based systems. *Ieee trans. on systems man and cybernetics*, July.

MOLDOVAN, D. LEE, W. LIN C., & CHUNG, M. 1992. Snap: Parallel processing applied to ai. *Ieee computer*, May, 39–49.

NAKAMURA, K., & IWAI, S. 1982. A representation of analogical inference by fuzzy sets and it's application to information retrieval systems. *Fuzzy information and decision processes*, 373–386.

PASIK, A. 1989. A methodology for programming production systems and it's implications on parallelism. *Phd thesis, dept. of computer science, columbia university.*

PATTERSON, D. 1989. Introduction to artificial intelligence and expert systems. *Wiley.*

PRATT, V. 1987a. Thinking machines: The evolution of artificial intelligence. *Basil blackwell inc.*, 72.

PRATT, V. 1987b. Thinking machines: The evolution of artificial intelligence. *Basil blackwell inc.*, 70–71.

PRATT, V. 1987c. Thinking machines: The evolution of artificial intelligence. *Basil blackwell inc.*

RAM, A., & L., HUNTER. 1991. A goal-based approach to intelligent information retrieval. *Machine learning: Proceedings of the eighth international conference*, 265–269.

RIESBECK, C., & SCHANK, R. 1989. Inside case-based reasoning. *Lawrence erlbaum. hillsdale, nj.*

SCHANK, R. 1991. Where's the ai. *Ai magazine*, 38–49.

SCHMOLZE, J., & GOEL, S. 1990. A parallel asynchronous distributed production system. *Proc. 8 th national conference on ai, aaai-90*, 65–71.

SCHMUCKER, K. 1984. Fuzzy sets, natural language computations and risk analysis. *Computer science press inc.*, 35–41.

SEJNOWSKI, T. 1992. Bee smart (article). *In [sejnowski, 92a]*, 142.

SHORE. 1973. Second thoughts on parallel processing. *Comput. elec. eng.*, **1**, 95–99.

SLADE, S. 1991. Case-based reasoning : A research paradigm. *Ai magazine*, Spring, 42–55.

STANFILL, C., & KAHLE, B. 1986. Parallel free-text search on the connection machine system. *Communications of the acm*, **29**(12), 1213–1228.

STANFILL, C., & WALTZ, D. 1986. Toward memory-based reasoning. *Communications of the acm*, **29**(12), 1213–1228.

STOLFO, S. WOLFSON, O. CHAN P. DEWAN H. WOODBURY L. GLAZIER J., & OHSIE, D. 1991. Parulel : Parallel processing using meta-rules for redaction. *Journal of parallel and distributed computing*, **13**(4), 366–382.

VAN RIJSBERGEN, K., & AGOSTI, M. 1992. The context of information retrieval. *The computer journal*, **35**(3), 193.

WALTZ, D. STANFILL, C. SMITH S., & THAU, R. 1987. Very large database applications of the connection machine system. *Technical report tmc-70, thinking machines corp.*, March.

WALTZ, D. 1986. Applications of the connection machine. *Technical report 86-12, thinking machines corporation.*

WALTZ, D, & STANFILL, C. 1988. Artificial intelligence related research on the connection machine. *Proceedings of the international conference on fifth generation, computer systems, tokyo*, December, 1010–1027.

WALZ, D. 1990. Parallel ai. *Proceedings: Eighth nataional conference on ai, aaai-90.*

WEBER, D. 1991. Parallel implementation of time delay neural networks for speech recognition. *Proceedings of the 1991 south african council for automation and computation conference on neural networks.*

WONG, S. ZIARKO, W. RAGHAVAN V., & WONG, P. 1987. On modelling information retrieval concepts in vector spaces. *Acm transactions on database systems*, **12**(2), 299–321.

ZADEH, L. 1965. Fuzzy sets. *Information and control*, **8**, 338–353.

ZHANG, X. WALZ, D. MESIROV J. 1993. Protien structure prediction by a data parallel algorithm. *Thinking machines corp. report tmc-155.*

ZIMA, H., & CHAPMAN, B. 1991. Supercompilers for parallel and vector processors. *Acm press, addison weseley.*

ZIMMERMANN, H. 1986. Fuzzy set theory and its' applications. *Kluwer-nijhoff publishing.*

ZORPETTE, G. 1992a. The power of parallelism. *Ieee spectrum*, September, 28–33.

ZORPETTE, G. 1992b. Teraflops galore. *Ieee spectrum*, September, 26–27.

CORRIGENDA SHEET - MSc THESIS - PAUL BAISE

Please note the following minor corrections:

Page 52	middle Diagram + to $\cdot$.	
Page 53	line 8	from top "continuous" to multi-valued.
Page 54	line 6	from bottom I^A to I^X.
Page 54	line 6	from bottom 2^A to 2^X.
Page 54	line 8	from bottom "A of X".
Page 60	line 6	from bottom "values".