ZONING IN VANCOUVER: AN EXPERT SYSTEM TO ASSESS DEVELOPMENT PROPOSALS

By

JULIAN FRANCIS ATKINS
B.A. Hons., Simon Fraser University, 1986

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ARTS (PLANNING)

in

THE FACULTY OF GRADUATE STUDIES
(School of Community and Regional Planning)

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
September 1990
© Julian Francis Atkins, 1990
In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

School of Community and Regional Planning

The University of British Columbia

Vancouver, Canada

Date September 28, 1990
ABSTRACT

A sample of Vancouver's zoning has been expressed as an expert system, microcomputer program, - Euclid - in order to demonstrate the feasibility of applying this tool in municipal planning, and to explore the desirability of such computer applications. Review of the literature on Expert Systems from a planning practice perspective showed that this is considered one of the more exciting and potentially useful developments emerging from Computer Science. Expert Systems are thought to be applicable to some planning tasks which are difficult to handle using the computer tools currently available yet suited to automation, however, there is in the literature very little empirical work on developing systems and testing the potential utility of expert systems in land use planning. Euclid is programmed in Turbo Prolog 2.0, a language accessible without extensive or specialized computer training. The first conclusions of the thesis are that simple but useful expert systems can be built rather quickly by planners, and that development control is a good application domain. The thesis also uncovered several weaknesses and inconsistencies which appear related to the way Zoning By-laws have been written, and suggests that the discipline of programming in logic may avoid or resolve such problems. Finally the process of developing a system is shown to be just important as the system itself: Expert systems are a way of thinking about a problem just as they are a way of solving it.
Table of Contents

CHAPTER ONE: INTRODUCTION

- Thesis page 1
- Context page 1
- Contribution page 1
- Methodology page 4
- Scope & Assumptions page 5
- Organization of Chapters page 7

CHAPTER TWO: DEFINING EXPERT SYSTEMS

- Artificial Intelligence page 10
- Expert Systems page 11
- An Illustrative Definition page 13
- Expert Systems Structure page 14
- The Knowledge Base page 16
- The Inference Engine page 17
- Representational Structures page 20
- LISP page 30
- Prolog page 33
- Turbo Prolog 2.0 page 36
- Expert Systems & Shells page 37
- A Sample of Expert Systems in Planning page 42
- Other Systems of Interest page 45
- Application Areas in Urban Planning page 48
- Characteristics of Tasks suited to Expert Systems Solutions page 53

CHAPTER THREE: AN EVALUATIVE FRAMEWORK FOR ASSESSING PLANNING TOOLS

- Towards an Evaluative Tool page 61
- The Evaluative Framework page 62

CHAPTER FOUR: A DESCRIPTION OF THE DEMONSTRATION SYSTEM - EUCLID

- Euclid page 70
- The Planner/computer Interface page 77
- The Inference Engine page 78
Table of Contents
(continued)

The Knowledge Base page 79
The Development Process page 85
A Sample Run Session page 88
Conclusions page 94

CHAPTER FIVE: AN EVALUATION OF THE DEMONSTRATION SYSTEM - EUCLID page 96
  In Relation to the Evaluative Framework page 96
  Conclusions of Applying the Evaluative Framework page 106
  Turbo Prolog As A Development Tool page 107
  Lessons Learned from the Zoning By-laws page 108
  Conclusions page 110

CHAPTER SIX: A GENERAL CRITIQUE OF EXPERT SYSTEMS page 112
  Theoretical Limitations page 113
  Practical Limitations page 114
  Professional Planning Limitations page 121
  Conclusions page 125

CHAPTER SEVEN: CONCLUSIONS page 128

THESIS BIBLIOGRAPHY page 135

APPENDIX 1 ZONING DATA BASE SPECIFICATIONS page 147

APPENDIX 2 INDEX OF PREDICATES CONTAINED WITHIN EUCLID page 150

APPENDIX 3 EUCLID PROGRAMMING CODE
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Expert Systems Structure</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>Backward &amp; Forward Chaining</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Representational Structures</td>
<td>21</td>
</tr>
<tr>
<td>2.4</td>
<td>The Frame Structure</td>
<td>22</td>
</tr>
<tr>
<td>2.5</td>
<td>The Blackboard Structure</td>
<td>23</td>
</tr>
<tr>
<td>2.6</td>
<td>Examples of Rules &amp; Facts</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Euclid Study Area</td>
<td>71</td>
</tr>
<tr>
<td>4.2</td>
<td>Zoning By-law Structure</td>
<td>72</td>
</tr>
<tr>
<td>4.3</td>
<td>Euclid Programme Modules</td>
<td>73</td>
</tr>
<tr>
<td>4.4</td>
<td>Planner/Computer Interface</td>
<td>75</td>
</tr>
<tr>
<td>4.5</td>
<td>Euclid System Flow Chart</td>
<td>89</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENT

I would like to acknowledge Canada Mortgage and Housing Corporation for supporting part of my M.A. studies through their Scholarship programme. I would also like to thank Mr. John French for very generously offering to help me produce the thesis on his computer. Without his help, the task of finishing the thesis would have been much more difficult. A final note of thanks must also go to my parents for supporting me throughout my studies and to my fellow students from whom I have learned so much.
CHAPTER ONE: INTRODUCTION

Thesis

The nature of planning practice is such that planners are routinely faced with numerous different tasks which require, for their completion, qualitative information just as much as quantitative information, if not more so. Computer tools which are available at present have difficulty in handling the former and so are almost exclusively confined to the latter. Thus, there are some planning tasks which have the potential to be automated and yet lack the requisite computer tools to do so. New research in both computer science and planning suggests that Expert Systems may be suited to some of these planning tasks since they can handle both types of information and provide a means by which professional judgement and expertise can be expressed in a microcomputer program, in a way which is transparent to the user and does not appear as some kind of magical black box. The purpose of this thesis is to evaluate the claims made in the Expert Systems literature by developing a demonstration system and to assess the feasibility and desirability of developing such tools in a municipal planning context.

Context

Society has, in recent years, witnessed an increasingly rapid pace of change as we move from an industrial to a post industrial society and beyond. More sophisticated technology, improved communications globally, and higher levels of education have resulted in much faster cycles of innovation and a more rapid diffusion of ideas. The pace with which urban growth is taking place has also increased. At the same time, society is experiencing a fiscal crisis, especially in its cities, as the cost of providing municipal services and upgrading civic infrastructure increases dramatically (see Gottdiener, 1986, for a discussion of fiscal stress).
Planners and other municipal officials are also faced with more immediate pressures as the growth ethic demands greater development which in turn results, particularly when the development industry is buoyant, in increased numbers of rezoning and development applications. Yet despite these increasing pressures, technology and the pace of innovation have, to some extent, also provided planners with a means of dealing with them. The development of the modern digital computer over the last 40 years or so has revolutionized the ways in which people work and solve problems. Many professions, including planning, have adapted to, and incorporated the power of the computer into their day-to-day activities. The advent of the personal, stand-alone, computer (PC’s or microcomputers to use the accepted industry jargon) has accelerated the recognition by planners of the computer as a piece of equipment necessary for successful planning practice. Indeed, Planners and Municipalities have not been slow in adopting both mainframe and Personal Computers (Micro Computers) in order to handle more effectively repetitive or routine tasks such as word processing and basic number crunching. Thus many departments employ database, spreadsheet, graphics, statistical, and/or integrated packages (Bardon, 1988, p.347) for storing information, analyzing data, forecasting & projection, as well as producing reports. At the same time, Planners also appear to be becoming more computer literate and are beginning to undertake more and more ‘in-house’ development and to produce more and more tailor-made software.

The majority of these packages, models and programs, however, require that real world situations be reduced to quantities, formulas and algorithms. These requirements have led some planners to become disenfranchised with such positivist computing approaches because they fail to incorporate the richness, or conversely the scarcity, of the real world and cannot represent all the elements of most 'planning problems'. They have also been criticized because they fail
to handle changes that occur in the context of a 'problem' and to clarify here, a 'problem' in the context of planning can mean the thing or area being evaluated. Such a definition encompasses a wide range of planning activity from developing goals, formulating plans, evaluating projects, and controlling development, to economic impact assessment, forecasting, projection, environmental impact assessment, and others.

Despite the development of such techniques and tools though, the primary functions of computers in planning departments today are the more menial tasks of information storage and word processing. Pencil, paper, and files have simply been replaced by keyboards, monitors, and disks and planners have not fundamentally restructured their working methods in order to take advantage of the capabilities of the computer tools available. Furthermore, most of the software being used: spreadsheets; database systems; and word processing packages, for example; is generic in nature. That is, each package is designed to be of use to more than one group of professionals. Unfortunately, in designing something for everyone's needs the software engineers have developed tools which meet no one's needs completely. Consequently, users have had to adapt their techniques to the software at hand. Planners are no exception to this adaptation but, in translation, refined and complicated planning techniques become simplified, or omitted altogether, in order to make the software work. Such a situation of making 'reality' fit the 'model' does little to address the apprehension users have of computers or of their faith in the possibilities of computers. What planners require is software which can support and enhance the planning process as well as meet the requirements of the planner/user, not vice versa. Such software though would have to address the criticisms levelled at quantitative and procedural computing methods and provide the planner with greater flexibility.
To this end, some planners are exploring the possibilities in a number of key areas, and research is progressing on three broad fronts: Computer Aided Design (CAD); Geographic Information Systems (GIS); and, to a lesser extent, Artificial Intelligence (AI)/Expert Systems. In the long term, combining all three areas is an obvious and desirable objective since it may allow the development of comprehensive and flexible tools which are geographically based yet make use of sophisticated graphics and which function in ways far more acceptable to professionals and staff than at present. However, while there are obvious overlaps between these three areas, current research is progressing along more or less separate lines. A perusal of recent conferences focussing on information technology and its application to the urban domain reveals that the most progress has occurred in CAD and GIS software. Indeed, numerous systems are already commercially available for these tasks (URISA, Conference Programs, 1989 & 1990). Much less work has been done in the area of AI and Expert Systems, and it is only in the last five years or so that Planners have become interested in the field and only in the last two or three that the first demonstration systems have started to appear. The application of AI techniques and software to Urban and Regional Planning though has the potential to make the greatest impact, of the three, upon the profession.

Contribution

The contribution that this research makes to the field of planning is to introduce expert systems to a planning audience by providing a comprehensive, balanced discussion of their structure and capabilities. Current planning research into Expert Systems has revealed substantial potential but as yet very little empirical research has been done. Therefore, another important contribution of this thesis is to take research to the next level by developing a simple
demonstration system, based upon current development control legislation, and by evaluating
the system itself, the process of development, and the information on which the system is based
(i.e. zoning schedules). Finally, another contribution is the identification of weaknesses and
inconsistencies in the content of the zoning schedules; these making the zoning regulations
themselves hard to interpret.

Methodology

The demonstration system presented - Euclid - has been built around a sample of the
City of Vancouver’s zoning schedules, specifically those schedules governing two-family
residential districts in Kitsilano, Grandview Woodlands, and Mount Pleasant. It is written in
Turbo Prolog 2.0 as a self-contained executable program and took approximately two person-
months to develop. In essence, the system is a translation of the six zoning schedules into a
series of logical rules, each corresponding to a specific zoning regulation or set of regulations,
with a set of rules corresponding to each of the six schedules. The system operates interactively,
asking the user a series of questions which determine whether or not a particular land/use and
site combination is permitted under the current zoning regulations. It then informs the user
whether the use/site proposed meets the current regulations or not. If the use fails, the system
will say why and in some cases will indicate relaxations which may be permitted.

Scope & Assumptions

The scope of the thesis is focused squarely upon the development and evaluation of the
demonstration system above and so is fairly technical in nature. Since the majority of readers
will be unfamiliar with the field of expert systems it has been necessary to preface the chapters
describing and evaluating the system with a fairly lengthy and detailed introduction to the concept of expert systems.

Consequently, the thesis is not concerned with the philosophical or ethical discussions which surround expert systems and artificial intelligence research. It assumes that the theoretical literature behind expert systems is valid in order to proceed with the development of the demonstration system and that Euclid, although it is a very simple tool, is an expert system because it represents a fundamental shift in computer programming and problem solving techniques. Thus, questions regarding the definition of what makes an expert system an expert system are not treated in any detail and are assumed to be of secondary importance to the main focus of the thesis.

In addition, efforts were made to collaborate on the development of Euclid with one of the local planning departments in Greater Vancouver. It was felt that such collaboration could help to provide Euclid with some form of rudimentary discretionary judgement in its analysis of development proposals. However, these efforts were met with a lack of interest and no offers of assistance were forthcoming. The scope of Euclid has therefore had to be restricted to the textual material contained in the Zoning Schedules. Consequently, Euclid represents only a portion of what would be necessary in a fully developed system.

The reader should also be aware that terminology plays a key role in expert systems literature and that such terms usually have very specific meanings. Thus terms which may appear to be familiar should not be interpreted with the meanings that are more generally associated with them. The term 'expert' in 'expert system', for example, does not necessarily mean that the system exhibits human or intelligent judgement and expertise, but more that it employs
techniques, in the way it functions and represents information, which are associated with the AI/Expert System field. Some of these terms are also found in the planning literature - 'top down' and 'bottom up' for example - and planners should be very careful not to read into the text meanings which are more commonly associated with their use in planning since this could lead to erroneous interpretations concerning the capabilities of expert systems.

Organization of Chapters

The thesis is divided into seven chapters each of which addresses a particular theme within the context of Expert Systems:

Chapter One consists of the introduction and places the thesis within the context of technological change, the increasing pace of innovation, fiscal stress, and increasing development pressures. It also outlines current practice in terms of computer applications in planning and identifies the three main areas of 'new' computer research within planning.

Chapter Two provides a fairly detailed definition and description of expert systems, drawing as it does on the existing planning literature vis-à-vis expert systems, as well as the literature which has emerged from the fields of computer science, psychology, and philosophy. At the same time Chapter Two explains how expert systems work and reviews the types of software and programming tools which are available. The chapter also reviews several existing demonstration systems which are relevant to an urban planning sphere and concludes with a discussion of potential urban planning applications.

Chapter Three presents an evaluative framework for use in assessing the performance of various software tools, including expert systems, in terms of their suitability for contemporary
planning practice. This framework draws on critiques of the positivist tradition as well as contemporary planning theory and artificial intelligence/expert systems literature outlining the supposed capabilities of 'intelligent' software. This framework then forms the foundation for evaluating the demonstration system described in Chapter four.

Chapter Four begins with the rationale for selecting development control and in particular zoning schedules as the domain around which the demonstration system - Euclid - is built. The chapter then describes how Euclid works, as well as the development process used to develop the system. It concludes with a description of how a sample interactive session would work.

Chapter Five assesses Euclid using the evaluative framework developed in Chapter Three and discusses problems which arose during the development process as well as the limitations of the system developed. An analysis of Turbo Prolog as a development tool is also included.

Chapter Six analyzes expert systems, and the claims which some supporters of the field make, from a critical perspective. This critique is based upon social sciences literature from such sources as Philosophy and Psychology and upon an evaluation of Artificial Intelligence/Expert Systems concepts from the planner's perspective.

Chapter Seven concludes the thesis with a short discussion of the principal findings. It reveals that Expert systems can indeed be developed quite easily around the domain of development control and that expert systems may also be applicable in other areas of planning. Furthermore, several weaknesses and problems with the written zoning schedules were un-
covered which seem to stem from the lack of a consistent writing style and the legal constraints which govern the drafting of By-laws. These can, in some cases, serve to hinder the clarity of the document and create an atmosphere of uncertainty and confusion concerning the correct interpretation. Finally, Turbo Prolog’s suitability as a programming and system development tool is discussed and it is suggested that initial efforts may be likelier to succeed if an expert system shell is used instead.
CHAPTER TWO: DEFINING EXPERT SYSTEMS

The development of Expert Systems has its roots in a branch of computer science known as artificial intelligence (AI). Research into AI and its computer applications has been ongoing since the early 1950s (Shipley, 1989, vol.2, no.3, pp.64-74) although the roots of the field arguably go back to the Greek's development of deductive logic (Aleliunas, 1989). Until the early 1980s, this activity was confined primarily to academic and R&D institutions, and only recently have a number of practical or commercial successes materialized; these mainly in the area of "expert systems" (Shipley, 1989, vol.2, no.3, p.67).

Artificial Intelligence

The essential thrust of AI is to develop computer hardware and software tools which mimic and exhibit the intelligent characteristics of humans. Tello provides one of the clearest definitions of Artificial Intelligence:

A basic definition might be that artificial intelligence is concerned with making computers do things that would be intelligent if humans did them. But this is too vague from either a practical or philosophical point of view. Some clarification might result from identifying some of the directions that are being pursued within the field. Currently, the following focuses can be identified within AI: (1) AI as a body of techniques and approaches for solving problems; (2) AI as the leading edge of computer science research in general; (3) AI as a class of currently available development and application products; (4) AI as a research tool for understanding how the human mind works...; and (5) AI as a venture to develop the equivalent of human intelligence by artificial means. (Tello, 1988, p.3)

(N.B. - Items 4 and 5 above are beyond the scope of this thesis.)

Within AI there are a number of sub-fields and knowing what these are is beneficial in grasping the full nature and scope of AI research. Again, Tello (1988, p.17) proves the clearest
and most complete source in outlining current applications and research areas, and has identified the following distinct sub-fields:

- Natural-language processing.
- Expert systems and knowledge based systems.
- Automatic programming.
- Intelligent robotics and autonomous systems.
- Machine learning and discovery.
- Vision and sensory recognition.
- Intelligent training and education.
- Intelligent user interfaces.
- Automated software and engineering design.
- Planning and general problem solving.

This thesis is primarily interested in item two above but it will be seen that items seven, eight, and ten also have potential value within the sphere of Planning. These four categories are mirrored in the application areas which have already been identified by planners and which will be discussed later in the chapter.

**Expert Systems**

Many of the successes hailed in AI to date, most notably MYCIN, DENDRAL, PROSPECTOR, and R1, are a product of expert systems research. Some have gone so far as to say that "... the one application area most synonymous with AI is flourishing. Expert systems are no doubt the most mature and resilient products to come out of the AI research community." (Shipley, 1989, vol.2, no.3, p.73). Indeed "... expert systems are widely thought to represent AI's first major achievement ..." (Pfaffenberger, 1988, p.64). Others have been more cautious in their praise: "While the development of Expert Systems clearly marks a watershed in computer
science, however, it could also represent one of the greatest potential dangers yet posed by the Mechanist Thesis.”¹ (Shanker, 1987, p. 114).

While it is proper to be cautious about the advent of any new technology, it is also clear that expert systems represent a new and different approach to those software packages and programming languages currently available in the market place. Conventional programming languages such as FORTRAN, BASIC, or Pascal, as well as general application software popular in planning departments (such as Lotus 1-2-3 or SPSSX), have to approach a problem or issue by collecting numerical data, assigning numerical values to, or ignoring completely, qualitative information, and then defining procedures, functions, and algorithms for reaching a solution. Data and procedures are intertwined and interdependent; a change or addition to one also requires modifications to the other. These tools are also predicated on a brute force approach to problem solving. That is to say these programs test every bit of the database they are given in searching for solutions. In the same way, spreadsheets like Lotus recalculate the entire spreadsheet every time a change is made or data entered. Expert systems on the other hand approach problems in a way which separates data from procedures and they are thus far more flexible.

¹The Mechanist Thesis implies that computers can communicate and can therefore understand whatever is being processed. The corollary of this is that human beings are machines and can therefore be ‘duplicated’ using computers. Shanker argues that supporters of the Mechanist Thesis commit the homunuclus fallacy in that they view the computer in anthropomorphic terms and ascribe human traits to something which is merely an electro-mechanical machine. Just as a pencil cannot write a computer cannot communicate or understand (Shanker, 1987, p.82).
An Illustrative Definition

While no definitive description of expert systems exists (Leary, 1988b, p.8), and while numerous variations on the theme have been developed, there are several distinctive features common to nearly all expert systems. These features help to explain how expert systems are applied to problem solving and how they differ from conventional techniques. Expert systems approach problems and issues by describing and defining the problem itself. They can work with information that is known, partly known, intuitively felt, or guessed about a problem or range of issues. Consequently, they can and have to work with uncertain or missing pieces of information (Leary, 1988b, p.8; and others). This may be one of their biggest advantages over algorithmic and linear programming techniques, especially in applications for the social sciences. Another important feature is that expert systems use rules of thumb (or "heuristics") as part of their information processing and reasoning structure. These heuristics are "... problem solving strategies that point in the direction of a solution but unlike algorithms, do not guarantee that the right answer will be found." (Leary, 1988b, p.8). An additional and important feature of expert systems is that they have the facilities to "explain", on the basis of the information, heuristics, and assumptions they employ, how results were achieved (Leary, 1988b, p.8; and others). This ability means that expert systems have strong potential in the areas of education and job training but at the moment these explanations amount to little more than a listing or tracing of the information and rules triggered during a session with a user. It is hoped that research in the area of natural language interfaces (another of AI's sub-fields) will improve the quality of such explanatory facilities and allow the user to employ a representational formalism closer to their native language than the somewhat cryptic and symbolic formalisms currently used for "communicating" with expert systems software.
Townsend gives an indication of the fields which have the potential to benefit from expert systems applications in his definition of expert systems themselves:

Expert systems are a class of computer hardware and software that can help advise, diagnose, analyze, consult, and categorize. They are tools for problems that normally require the use of a human expert, professional or specialist. Unlike traditional computer hardware and software that attempts to solve problems using defined procedures, expert systems attempt to solve problems in specific disciplines using deductive reasoning. As a result, expert systems are capable of solving problems that are unstructured and poorly defined. (Townsend, 1987, p.4)

Another facet of expert systems is revealed by the following description:

Expert systems actually have built into them the knowledge of human experts, and they draw on it to solve problems. ... this differs markedly from conventional computer programs, which manipulate numbers and quantities, pluses and minuses, in precisely specified ways. (Van Horn, 1986, p.5)

This description illustrates how expert systems are linked to the field of AI and are imbued with AI philosophy. Implicit in Van Horn's definition is the fact that expert systems can capture all human knowledge perfectly and thus perform a specific task as well or better than their human counterparts. They should therefore be able to replace human experts in virtually any limited area. It is sufficient at this stage to simply point out that there is considerable debate concerning the development and use of expert systems for this purpose; this is something which will be discussed in more detail later on. For now we will limit our discussion to the positive, and hence biased, literature on expert systems.

**Expert Systems Structure**

Expert systems consist of two basic elements - a knowledge base (analogous to a database) and an inference engine - although additional components may also be incorporated
Figure 2.1

Expert Systems Structure

into any given system. A central element of expert systems (also called knowledge-based systems) is the fact that the inference engine and knowledge base are separate and independent entities not only in terms of content but also structure (Cullen, 1986, p. 243). Figure 2.1 shows the basic structure of a generic expert system as developed currently.

The theory behind expert systems is that professionals and experts use facts, heuristics, and logical, easily identifiable, reasoning processes to evaluate and solve problems. "Heuristic methods include rules of thumb, intuitions, simplifications, judgements, and other problem-solving approaches that may not find the best solution but often find useful solutions quickly." (Ortolano & Perman, 1987, p.98) All one has to do to encapsulate this knowledge, and the reasoning processes used, into an expert system is to extract them from the expert - a process known as Knowledge Elicitation - and to use a suitable representational structure, or formalism, for 'storing' the information in the knowledge base - known as Knowledge Engineering. Obviously, all of an expert's knowledge could not possibly be encoded into a knowledge base in its entirety since a human's range or domain of experiences is so vast. The way expert systems designers have addressed this problem is to identify specific tasks (from the range of tasks a human expert handles) which are amenable to expert systems solutions. Thus, expert systems are designed to favour domains (i.e. problem areas) which are well structured and narrow in scope and which represent a portion of a human expert's domain.

The Knowledge Base

The Knowledge base acts as a kind of library or repository of knowledge or information. It contains both quantitative and qualitative information such as facts, definitions, data, heuristics, and other 'expert knowledge', which describe the character of a particular domain. "Sources
of that knowledge include published materials, quantitative analysis programs, and the intuitions and problem-solving strategies of experts in the subject area." (Ortolano & Perman, 1987, p.98). This information is then stored using a symbolic shorthand notation which can be processed by the inference engine and which can be modified or added to relatively easily.

The Inference Engine

To use this stored knowledge one has to create a mechanism - the inference engine - for searching the knowledge base and manipulating the information contained within. The engine describes (using the same representational structure as the knowledge base) how the 'symbols' (or information) contained in the knowledge base are to be manipulated and the order in which inferences are to be made in any given situation. "The main requirements for an inference engine are (1) a way of determining when and where to start, (2) a procedure that defines how to search toward a result, and (3) a way of determining either that the result has been found or that no result can be found." (Tello, 1988, p.106). To achieve these objectives the engine must be able to carry out two functions: A) Control the way in which the knowledge base is searched, and B) Make inferences about and between elements of the knowledge base.

The two most common control strategies employed for searching a knowledge base are forward chaining and backward chaining (Townsend, 1987, p. 139). Backward chaining is goal directed, or to use the computer jargon a "top down" search method. The system takes a goal as given and tries to substantiate it from information contained within the knowledge base. Forward chaining operates on the reverse principle and is a 'data-driven' or "bottom up" search strategy. Forward chaining takes facts and tries to deduce conclusions (i.e. goals) from them (Leary,
Figure 2.2

Backward Chaining:

Conclusions:
(Goals)

Forward Chaining:

Conclusions:
(Goals)

The difference between these two strategies is illustrated in Figure 2.2. For a more detailed and comprehensive discussion of search and control strategies see Tate (1985).

The inference engine also contains the mechanisms by which inferencing, that is the system’s ‘reasoning process’ is carried out. These mechanisms are dependent on the representational structure employed (these are discussed below) and therefore on the type of software tool used, but regardless of these the inference engine performs several functions which make the reasoning process work. The basic functions that an inference engine requires, as expressed by Townsend (1987, pp. 132-3), concern missing information, uncertainty, and the facility for producing explanations. First, in the event that the system cannot prove a goal or reach a conclusion due to missing information (or for any other reason) the engine must be able to backtrack and try different avenues of enquiry. In this way the engine directs the search only to those areas of the knowledge base which are likely to lead to success. Second, much information, especially in fields like planning, has a level of uncertainty associated with it. The most common way of dealing with this problem is to attach uncertainty values (usually between 0 and 1 and hence similar to probabilities) to elements of the knowledge base and to include a function for propagating these values through the knowledge base as goals or sub-goals are substantiated or conclusions reached. It should, however, be noted that serious theoretical flaws have been discovered with this approach and that nearly all the other techniques which have been developed to deal with uncertainty are ad hoc at best. Finally, as has already been discussed, the inference engine needs to contain a component which explains how and why a result was achieved.

At this stage it should also be noted that some expert system software tools have a built in inference engine, leaving the user to supply only the information for the knowledge base.
Some of these tools allow one to manipulate and modify the engine although quite a few, especially the cheaper ones, do not.

**Representational Structures**

Several representational structures have been developed for encoding information into the knowledge base and for programming the components of the inference engine. Among these are frame representations, blackboard environments, and production-rule systems, and some software tools allow combinations of these structures to be used (Townsend, 1987, p. 136). Neural networks, another representational structure, represent knowledge in terms of interconnections between nodes. Neural nets are thus themselves information whereas conventional schemas are merely representations and store information as discrete packages. "... neural nets emulate the webs of neurons that make up the brain itself. Thus, neural nets are composed of thousands of interconnected, simulated neurons. Unlike conventional rule-based computer systems, in which instructions are executed serially, neural nets can process multiple instructions synchronously." (Jerome, 1989, vol.2, no.3, pp.72-3). While fascinating in and of themselves, neural nets are really beyond the scope of this thesis since we are interested more in tools for urban planning rather than in systems which model the brain (for a wider ranging discussion of neural nets see the December 1989 issue of *AI Expert*).

While the form of the knowledge base depends upon the representational structure used it is convenient for present purposes to think of the knowledge in terms of an hierarchical tree structure similar to a semantic network (Figure 2.3). Each node in the tree can be thought of as a piece of information - a fact for example - while the links between the nodes indicate the
Figure 2.3

A Semantic Network to Diagnose a Car's Failure to Start:

\[
\text{CAR} \quad \text{NODE} \quad \text{is}_a \quad \text{LINK} \quad \text{has}_a \quad \text{PONTIAC} \quad \text{STARTER MOTOR} \quad \text{is} \quad \text{NOT DEFECTIVE} \\
\text{ALTERNATOR} \quad \text{has}_a \quad \text{BATTERY} \quad \text{has}_a \quad \text{BRUSHES} \quad \text{WATER} \quad \text{CONNECTIONS} \\
\text{is} \quad \text{LOW} \quad \text{is} \quad \text{NOT DEFECTIVE} 
\]

Source: Townsend, Carl; Mastering Expert Systems with Turbo Prolog; Howard W. Sams & Co., Indianapolis, Ind.; 1987, p. 121

A Hierarchical Tree Structure Describing Design Guidelines for Ensuring Internal Privacy

\[
\text{HOUSE ENVELOPE - INTERNAL PRIVACY} \\
\text{PUBLIC ZONE} \quad \text{PRIVATE ZONE} \\
\text{SET FORWARD} \quad \text{SMALL WINDOWS} \quad \text{SET BACK} \quad \text{SCREENING} \quad \text{DESIGN} \quad \text{REMTENESS}
\]

Source: Leary, M.E.; "Knowledge and reasoning in development control and urban design: an expert systems approach; Environment and Planning B: Planning and Design; 1988; Vol. 15, p. 391.
Figure 2.4

The Frame Structure:

Frame Concept

Slot #1 Value #1

Procedure #1

Slot #2 Value #2

Procedure #2

Slot #3 Value #3

Procedure #3

Slot #4 Value #4

Procedure #4

A Frame Representation for Classifying Computers:

Format:

Example:

Microcomputer

John's PC

CPU

80286

Monitor

Colour

Keyboard

Enhanced

Ports

2

Printer

Yes

Source: Floyd, Michael; "Suitable for framing: An object-oriented approach using frames can make an unruly knowledge base more manageable."; Turbo Technix; March/April, 1988, pp. 80-4.
Figure 2.5

A Blackboard System for Job Identification

Control Blackboard: Controls how the occupation and personality blackboards function.

<table>
<thead>
<tr>
<th>Level of Abstraction</th>
<th>Control Strategy:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>General</td>
</tr>
<tr>
<td>Strategy</td>
<td>Specific</td>
</tr>
<tr>
<td>Focus</td>
<td></td>
</tr>
<tr>
<td>Policy</td>
<td></td>
</tr>
<tr>
<td>To-do-set</td>
<td></td>
</tr>
<tr>
<td>Chosen Action</td>
<td></td>
</tr>
</tbody>
</table>

Occupation Blackboard: Identifies suitable jobs

Direction of Inferencing

Specific Jobs
Job Types
Abilities
Interests
School Subjects

Occupation

Personality Blackboard: Identifies personality traits and passes them on to the Occupation Blackboard

Direction of Inferencing

Relationship
Character
Features

Personality

explicit relationships - e.g. rules of thumb - between nodes which the inference engine uses to search the knowledge base (Leary, 1988a, p.389; Townsend, 1987, p. 121).

Frame representations are similar in structure to databases, or records in a spreadsheet program and "... are useful in handling complex situations involving objects and classes of objects." (Ham, 1984, vol.2, no.1, p. 37). The frame is used as a tool for organizing disparate pieces of information, such as objects or concepts, into groupings and each frame has 'slots' for storing all the information associated with the object or concept it represents (see Figure 2.4). Each slot can have a value or attribute attached to it as well as a procedure describing how the information in that slot is to be handled (Floyd, 1988, p.80) and the frames are linked to each other by a hierarchical structure. An important feature of hierarchical frame structures is that frames can inherit the properties of frames above them in the hierarchy (Townsend, 1987, p. 124). For a more detailed explanation of frame representations see the rest of Floyd’s article (1988, pp. 80-8) which shows an example of a frame representation using Turbo Prolog, as well as Townsend (1987, pp. 124-7, 135-6), and Tello (1988, pp.414-6).

Blackboard systems consist of a common workspace - the 'blackboard' - and multiple processing components, each of which works on a particular aspect of the overall task (see Figure 2.5). "Blackboard systems are used when external events arrive at the system sporadically and many levels of knowledge are required." (Ham, 1984, vol.2, no.1, p. 37). The blackboard itself controls how the various components are used and contains procedures which handle each component as if it were a database. Information and results retrieved into the blackboard from a particular 'database' become accessible by all of the other 'databases' for use in further computations (Ham, 1984, vol.2, no.1, p. 37). It should be clear that the individual 'databases'
Figure 2.6

An Example of Rules and Facts in a Rule Based Expert System: The URBYS System

URBYS contains rules in the following semi-English format:

1. public land(43);
2. IF public land (X) THEN any new construction should be limited in (X).
3. IF public land (X) AND close to center (X) THEN any new construction should be limited in (X) AND reserved for green spaces (X).
4. IF (X) is the town center AND (Y) is a direct neighbour of (X) THEN (Y) is close to the town center.

These can be translated as meaning the following:

1. area 43 is public land;
2. if a zone is public land, then any new construction must be limited in this area.
3. if a zone is public land and it is close to the center then any new construction should be limited in this area and the zone should be reserved for green space.
4. if an area is the town center and another area is next to the first, then the latter area can be defined as being close to the town center.

in such a blackboard system could be expert systems themselves and these could be rule- or frame-based, thus creating quite a powerful and flexible structure. For an example of an expert system which uses this blackboard architecture see Boyle's description of the JOBBES system, in Merry (1985, pp. 273-86).

Production-rule systems are by far the most common representational structure employed in expert systems (Townsend, 1987, p. 129) and express information in the knowledge-base in terms of statements (i.e. facts) and IF–THEN rules. Logical operators - AND, OR, NOT (see Schnieder et al, 1982, p. 101) - can also be incorporated into the rules to create conjunctive and disjunctive clauses. Figure 2.6 illustrates what a rule-based system looks like in its English language equivalent, using information from an urban planning domain, and is taken from Tanie’s discussion of the URBYS system (Tanic, 1986, p. 139).

Thinking back to the semantic network discussed earlier it should be clear that the factual statements in a rule-based system are analogous to the nodes of a semantic network and the IF–THEN rules are the linkages which explain the relationships between facts. These rules and facts can represent qualitative information since they are based on predicate and not propositional logic (Van Horn, 1986, p. 137). Under predicate logic (or predicate calculus as it is also called) the IF–THEN rules are treated as IF-proposition(s)-THEN-conclusion statements. To verify if the conclusion is true, a backward chaining system would search the knowledge-base to determine if the proposition(s) is true (i.e. if the information in the knowledge base supports the proposition). If it is true then the conclusion would also be treated as true whereas if the information in the knowledge base is insufficient or refutes the proposition(s) then the conclusion is considered to be false (Van Horn, 1986, pp. 137-41). Such true/false determinations
can be made because predicate logic operates using a principle known as MODUS PONENS. This "... states that if a rule exists such that 'IF A, THEN B' and if A if known to be true, then B is true." (Townsend, 1987, p. 132). Note that these true/false judgments are only true/false in a logical sense. All a return of "true" means is that the system can 'prove' the conclusion, while "false" simply means that the conclusion cannot be proved using the information currently contained in the knowledge base. It does not mean that a conclusion has any inherent truth or falsity built into it. Note also that the use of a true/false test does not allow for uncertainty to be incorporated into the rules and statements since predicate logic in its strictest sense can only deal with absolute truths and absolute falsities. This problem can be solved to some extent by incorporating relational operators (such as <, ≤, =, ≠, ≥, >, see Schnieder et al, 1982, p. 101) into the rules and statements of the knowledge base although this would also require the inclusion of numeric values for testing the relations and would still not provide a method of handling uncertainty.

In order for the system as a whole to work, the conclusion section of a rule has to be tied to the proposition section of other rules in the knowledge base. Rules and facts are thus triggered by others which are linked to them or as a result of asking the user the answer to a specific question through what is known as a 'query-the-user' facility. Notice that in this regard a fact can be thought of as a rule which is always true (i.e. it is not conditional since it does not have a proposition attached to it).

Production-rule systems allow information to be stored in a well structured manner, the rules and facts in the knowledge base creating a very clear description of the task which makes the system easy to both build and read. As the knowledge base becomes larger and more
complex, however, it becomes unwieldy, difficult to control, and less easy to interpret since more and more rules are required to express the problem clearly (Aleliunas, 1989; Floyd, 1988, p. 80). Nevertheless, for the purposes of this thesis and the demonstration system, this rule-based approach is sufficient for presenting the concept of expert systems applications in urban planning.

**Software Tools for Developing Expert Systems**

There are two principal types of software available commercially for developing expert systems applications: 1) Programming languages; and 2) Purpose built expert systems and expert system shells. Within these two categories there are a wide variety of software tools. Indeed, a quick survey of the marketplace (based upon lists of vendors published in the magazine *AI Expert*) reveals that some 39 expert system tools are available (Anon, 1988, vol.3, no.9, pp. 71-2, 74, 76-7) as well as 22 versions of the AI language LISP (Hildebrand, 1988, vol.3, no.3, pp. 67-72) and 24 versions of Prolog (Shafer, 1988, vol.3, no.6, pp. 71-5). Other languages, such as Smalltalk and OPS5, have also been developed for AI and Expert Systems research, and procedural languages such as Pascal, FORTRAN, or C can also be used although developing workable applications with the latter can prove to be difficult (Tello, 1988, p. 323). Obviously, the choice of which software tool to use is dependent on installation and training costs, one’s existing hardware, software, and human resources, software performance, and the types of applications to be developed. Some languages and tools require mainframes, dedicated workstations, or even supercomputers to operate and the software alone can cost anywhere from about US$ 1,000 to US$ 70,000 (Burlinson, 1989). Hardware designed specifically to run particular types of AI software is also available but it too is expensive (Sobkowski & Tischler, 1988, vol.3,
LISP machines for example are designed solely to run the LISP language and the hardware/software comes as one integrated package which can be difficult to learn; see Graham, (1988, vol.3, no.12, pp. 26-32) for a more detailed discussion of such machines.

By contrast, there are numerous software tools available for the current range of personal computers: IBM PC’s, XT’s, AT’s, and compatibles; 80386 machines; PS/2’s; Apple Macintosh’s; and there are even tools for 'home' computers such as the Atari ST and Apple II series. These software packages can cost anywhere from US$ 50 for relatively simple tools to a few thousand dollars for very powerful development environments, though there are quite a few powerful and easy to use tools, such as Level 5, at around the US$ 600 mark. In terms of selecting software for any task, including expert systems applications, it should be pointed out that substantial thought needs be given to the availability and cost of support material and services in addition to the cost and performance of the system itself. Support services can include tutorial packages, user guides and manuals, consultation and trouble shooting assistance, and software updates, in addition to the system itself (Hildebrand, 1988, vol.3, no.3, p.68). Another important factor which should be considered and which emerged in the course of selecting the software for the demonstration project developed here is the availability of handbooks and reference material published by third party sources.

Because of the vast and diverse range of tools available, it is impossible to cover all of them here and so the discussion will be limited to a generic description of two of the more popular classes of AI programming languages: LISP and Prolog; and to that class of tools known as Expert Systems and Expert System Shells. For a more in-depth analysis of these and a wide range of other tools see Tello (1988).
The LISP, or LIS$\text{t}$ Processing, language was developed by John McCarthy in the mid 1950s and is thus second only to FORTRAN in terms of its age (Tello, 1988, p. 323). It is also the standard language of AI and Expert Systems research in the United States although it has been challenged more recently by Prolog. As its name implies LISP is designed to manipulate lists of information but the feature which differentiates it from other languages is that this information is symbolic not numeric. In fact LISP is notoriously poor at handling numbers and formulas because it has to treat them as symbols (Hasemer, 1984, p. 2). The building blocks of LISP are not numbers but atoms; these can be words, numbers, phrases, etc.. One or more atom(s) can be used to make up a list and variables can be 'bound' to these atoms for the purposes of evaluation, using several functions available for manipulating the lists. LISP has a few other distinguishing characteristics worth noting. Firstly, its interactive nature means that the computer responds each time the user inputs something (although this is only true of interpreted LISPs and not compiled LISPs). Secondly, like Pascal, LISP allows recursion, that is for functions to invoke themselves (Schnieder et al, 1982, p. 299; Hasemer, 1984, pp. 23-6) and thus allows lists to be manipulated by a function repeatedly until the desired result is achieved. And finally, functions and data are described using the same language structure and are thus indistinguishable from each other.

An example of LISP code for defining a function called "do-something-with" is shown below to illustrate this point as well as the LISP language as a whole and is taken from Coffee (1988, vol.3, no.3, p. 39). In the example, anything to the right of a semicolon is a comment to clarify what the function does, and is ignored by the computer during execution. Note also that
"list-of-values" is a variable which represents a list of items and that the function names are user-defined surrogates of standard LISP functions (see Coffee, 1988):

```lisp
(define-a-function
  ;; called
do-something-with
  ;; taking as input
(list-of-values) ;; a list called list-of-values
  ;; and working as follows:
(if (null list-of-values) ;; if list-of-values is empty
  nil ;; then return the result "nothing"
  ;; otherwise, return the result of
(make-list-beginning-with
  (some-function-of (first-of list-of-values))
  ;; and ending with the result of
  (do-something-with (rest-of list-of-values))))
```

This code defines a function called "do-something-with" which has as its input a list of information called "list-of-values". If "list-of-values" has nothing in it (i.e. it is empty) the function returns the result "nil" which means that "do-something-with" can't do anything since there is no input for it - you can't do something with nothing. If, on the other hand, the list contains something then the function is told to construct a new list called "make-list-beginning-with" by applying the function "some-function-of" to the first element of "list-of-values" and by applying "do-something-with" to the remaining elements in "list-of-values". This latter call to "do-something-with" occurs within the original definition of "do-something-with" and so invokes the whole "do-something-with" function again, this time to the remaining elements of the "list-of-values". Consequently, "do-something-with" applies the function "some-function-of" to every element of "list-of-values" until the list is empty and then stops; this is the principle of recursion. The example below illustrates recursion more clearly and is taken from Hasemer's introductory text on LISP (Hasemer, 1984, pp. 24-5). The pseudo-LISP code:
(defun shorten (L)
  (print L)
  (stop if the list L is empty) ;; i.e. return 'NIL'
  (shorten (cdr L)))

defines a recursive function called 'shorten' which takes a list, prints it out and then deletes the first entry (the function of the (cdr L) term). The last line in the above code contains a call to 'shorten' (using the truncated list) which is within the limits of the original 'shorten' call (i.e. the function is recursive) and so 'shorten' starts all over again and keeps repeating the same steps until the list is empty. Thus given a list called animals and containing three atoms - (CAT MOUSE DOG) - the call "(shorten animals)" would result in the following output:

(CAT MOUSE DOG) - The original list
(MOUSE DOG) - The list minus the 1st element
(DOG) - The list minus the 1st & 2nd elements
NIL - Stop since the list is now empty

These features mean that LISP is tremendously flexible and can modify and write itself as well as build complex functions from a standard set of simple functions. It is, therefore, useful for developing systems which can "learn" or need to add to their store of knowledge. However, LISP also has a reputation for being slow and for requiring large amounts of memory space in order to operate (hence the need for dedicated LISP machines). The author also found it difficult to interpret code listings. For a more in-depth introduction to the LISP language see Halsemer (1984).
Prolog stands for **PROgramming in LOGic** and was developed by Alain Colmerauer's team at the Universite du Marseilles in 1972 (Tello, 1988, p. 364). It is much more popular in Europe than in the U.S. and has been selected by Japan as the basis for their "Fifth Generation Project" computer initiative. The standard syntax for Prolog is Edinburgh Prolog developed by Clocksin & Mellish and their text: **Programming in Prolog** (1987, 3rd edition) is the standard reference work. Prolog, like LISP, is a symbolic, declarative language but instead of being built around list manipulation it is based upon first order predicate logic (though both LISP and Prolog can handle IF-THEN rule structures) and this has built into it a basic inferencing mechanism which can be augmented by the user. The similarity between first order predicate logic and natural language constructions makes the concept of Prolog deceptively simple to understand at first but provides a powerful language which can become very complex. The easiest way to understand Prolog is to compare English sentences with their Prolog syntax equivalents. Suppose, for example, that we had the following land use information which we wanted to express using Prolog syntax:

1). Council's policy is that land in D.L. 273 which is zoned for public use should not be developed.

2). Lot 12, in District Lot 273, is zoned public use.

These would look like this in Prolog:

1). do_not_develop(X) :- is_in(X, d_l_273), is_zoned(X, public_use).

2). is_in(lot_12, d_l_273).
   is_zoned(lot_12, public_use).
Notice that the Prolog statement in 1) represents an IF -- THEN rule and that the two statements in 2) represent facts. To clarify the more confusing elements of Prolog syntax: ":-" translates as "if"; a comma after a fact as the logical operator "AND". Like LISP, the same Prolog structures are used to describe both facts and rules and thus pieces of "data" are indistinguishable from the "guts" of the program even though they remain functionally distinct. It should be clear that we could add more facts to the above two, as in 3) below:

3). is_in(lot_11, d_1273).
   is_zoned(lot_11, commercial).

without having to alter rule 1) and that this rule would still be able to evaluate the additions. By the same token, we could add additional rules to operate on the facts available and it is in this manner that rule-based expert systems are built using the Prolog language. Notice that the facts and rules in such an expert system can only be true or false since Prolog's inferencing mechanism - predicate logic - and pattern matching capabilities can only determine absolute truth and absolute falsity. The way of solving this problem, as mentioned earlier, is to incorporate one of several uncertainty techniques for handling the inferencing process when information is missing or has some form of uncertainty associated with it. Among these techniques are: Zadeh's fuzzy logic or fuzzy sets (which allows truth values to range between 0 and 1 and includes rules for relating these to one another); Shortliffe's certainty factors (already mentioned); and non-monotonic logics (non-linear reasoning heuristics - "leaps of thinking" (Burlinson, 1989)). For a more in-depth discussion of what constitutes uncertainty and a description of the techniques which have been developed to handle it see the comprehensive paper by Mamdani et al in Merry (1985, pp. 181-194).
One of the strongest features of Prolog is the relative ease with which the programming code can be read and this should be readily apparent from the above example. In this respect, it is easier to read than LISP although trying to follow the way that predicate logic works in even relatively small knowledge bases can prove tricky, especially for those with knowledge of procedural languages.

Prolog works by responding to questions (or goals) posed by the user. Prolog then uses the backward chaining technique, already mentioned, and searches the knowledge base trying to find information which verifies the goal. Forward chaining can also be programmed in Prolog but the majority of Prolog programs backward chain since this control strategy is an inherent component of the language. Thus if we asked, based on the knowledge expressed above:

4). ?- do_not_develop(What).

Prolog would respond:

5). What=lot_12

N.B. "?-" is a Prolog prompt and "What" is a variable since it starts with a capital "W". Thus constants start with lower case letters.

The chief advantage of Prolog over LISP is that programs usually take up less memory space. During processing, however, it is sometimes difficult to prevent the backward chaining process from getting out of hand (Tello, 1988, p. 384). Consequently, the efficiency of Prolog depends upon the order of the rules and facts within the knowledge base and the way in which rules narrow the search space early on. Even so, as a language in which to develop demonstration systems, Prolog can yield results in less time than with other languages since it can be built up
in modules (Cooper, 1989, pp. 5-9) and because it is primarily a declarative language and describes a problem not how to solve it.

**Turbo Prolog 2.0**

While most versions of Prolog follow the Clocksin & Mellish standard, a popular alternative dialect, is Borland International's Turbo Prolog, introduced in 1986 and now available as Version 2.0 (Horak, 1988, vol.3, no.6, p.77). There is great debate over whether or not Turbo Prolog is a true Prolog language since it does not contain some of the standard Clocksin & Mellish predicates and incorporates "strong typing" in order to run faster. The use of "strong typing" means that every predicate and argument type must be specified at the beginning of the program and in this respect Turbo Prolog is very similar to the Pascal language which also incorporates type checking (Rich & Robinson, 1988, pp. 62-6). Furthermore, Turbo Prolog is compiled rather than interpreted and again this is in the interest of speed. Compiled languages translate programs into machine code and evaluate them in one go whereas interpreted languages translate and evaluate programs line by line and are thus more flexible. These limitations mean that Turbo Prolog does not have the flexibility or sophistication of standard Prologs and thus cannot handle truly dynamic, flexible knowledge bases (Shammas, Sept. 1986, p. 295).

Nevertheless, while Turbo Prolog is not really a fully functional Prolog language, it does have several advantages and has been recommended as a suitable language for beginners because of its simplicity (Horak, 1988, vol.3, no.6, p. 77). Chief among these is that Turbo Prolog is quite a powerful and yet inexpensive (less than US$ 150) tool and has excellent documentation and user support. There are also a wide variety of third party texts describing the language
available. Turbo Prolog is able to handle external databases and quite complicated graphics and also has an editor and development environment which is very easy to use. These features mean that it is "... a robust tool for rule-based decision support, statistical analysis, and general graphics." (Horak, 1988, vol.3, no.6, p. 78) and as a result it has been chosen as the tool with which the demonstration system will be developed. For a more detailed discussion of Turbo Prolog's features see Rich & Robinson (1988) and Rodgers (1987).

Expert Systems & Shells

Another variety of software for developing expert systems are the purpose built tools that either provide a ready-made "off-the-shelf" expert system or an environment in which expert systems can be built. Ready-made systems contain a developed knowledge base and inference engine which can be put to immediate use in solving problems for a specific domain. More interesting, though, are those tools which allow users to develop their own applications. Many of the simpler development tools contain a pre-programmed inference engine and provide the user with tools for building their own knowledge bases. That is, they are expert systems with empty knowledge bases - these types of tools are known as expert system shells. To produce a functional expert system, all the user has to do is to supply the appropriate information for building a knowledge base in the desired domain. Some shells, such as VP-Expert, PC-Expert, and others (see the magazine AI Expert), provide the user with a spreadsheet-like workspace for organizing and structuring the knowledge to be used into record-type entries. VP Expert even provides routines for linking its expert systems with spreadsheet programs such as VP Planner or Lotus 1-2-3 (Tello, 1988, p. 132). These types of shells then automatically create the rule base using pre-defined routines. Thus, a knowledge base can be built from case-data or a
series of examples. The user then provides a series of questions which are linked to the rules in the knowledge base. These questions are used to "query-the-user" for answers to each question. The input provided is checked against the rule base to determine a result or to identify the next question which needs to be asked. This process is repeated until the system exhausts all the possibilities and/or arrives at a conclusion.

This query-the-user process is used in most expert systems and it is important to note that the user is only asked questions and provided with responses which are of direct relevance to the user's needs. This is unlike the process used by most database and interactive systems which have to search their entire databases and ask irrelevant questions in order to provide results, some of which may themselves be irrelevant. In this sense then, Expert Systems search their knowledge bases in an almost intelligent manner.

Advanced shells, such as Level 5, provide more flexible development tools but also they provide menu options with explanation facilities. For example, Level 5 allows the user to ask at any point in a consultation session what the system is trying to prove and why it is trying to prove it. These facilities mean that the user can assess the reasoning processes behind a result and thus the system becomes more transparent to the user. At present though, such explanations amount to no more than a display of the current rule being evaluated and the goal being pursued, or a listing of the rules triggered and facts deduced up to the point of query. They are an improvement on AI language tools which require that the user program explanation facilities into the expert system in addition to having to program the system itself. As natural language interfaces improve, these facilities will no doubt improve and provide more dynamic interaction with the user.
MYCIN

MYCIN, one of the first expert systems, was the result of a research project at Stanford Medical Center to produce a computer tool which could assist doctors in the diagnosis and treatment of a range of infectious blood diseases (Van Horn, 1986, p.9). It operates in a very narrow and well structured domain and contains a knowledge base which relates symptoms to associated diseases. Users go through an interactive session by responding to questions about symptoms possibly exhibited by the patient and by supplying information in the form of test results. Throughout a session MYCIN interacts with the doctors in English.

MYCIN led to the development of EMYCIN, one of the first expert system shells to be developed. EMYCIN, or "Empty MYCIN", contained MYCIN’s inference engine without its medical knowledge base and left the user to supply information for their domain of interest. While MYCIN is hailed as a landmark system, some have noted that its success rate is only about 70%. Consequently, it will yield a mis-diagnosis in three out of every ten cases and while this may be better than the performance of a beginner it appears to be no match for a true expert with many years experience (Dreyfus & Dreyfus, 1984, p. 231). This raises some serious ethical questions about expert systems and it has been shown that in order to improve the success rate of any expert system the number of rules in the knowledge base has to be expanded substantially. Indeed, the law of diminishing returns means that each unit increase in the success rate requires an ever increasing number of additional rules. For more information on MYCIN see Tello (1988, pp. 101-5) and Van Horn (1986, pp. 5-10).
PROSPECTOR

PROSPECTOR was developed at the Stanford Research Institute (now SRI International) during the mid to late 1970s with the help of the U.S. Geological Survey (Tello, 1988, p. 101). PROSPECTOR consists of several ore deposit models which take advantage of expert systems technology. These models are designed to assist geologists in evaluating field exploration data for a specific site and in determining where mineral deposits are likely to exist. PROSPECTOR is a rule-based mainframe expert system, written in INTERLISP (a dialect of the LISP language), and contains geological information which is represented in the knowledge base using a combination of taxonomies and semantic networks. Fuzzy set theory is used to handle uncertainty (Benson, 1986, p. 19-23). The ore models were built up from case data and were refined through a series of workshops and discussions with geological experts as well as through sensitivity analyses and performance tests using field data where the outcome was already known (Benson, 1986, pp. 23-5).

During a typical run session, PROSPECTOR requires that the user enter field observations pertaining to the site in question and this information is then compared to the system's models concerning soils, rock structures, minerals, etc. If the information input is insufficient for it to reach a firm conclusion, PROSPECTOR will ask for additional information and can provide a rationale as to why that information is needed. Once it has enough relevant information to reach a conclusion it summarizes the results (Benson, 1986, p.18). PROSPECTOR can also explain how it reached its conclusion or what it is trying to do at any point in the interactive session.
PROSPECTOR is an important system because it is hailed as one of the greatest successes of the AI and expert systems industry to date. It is claimed by many to have been 'responsible' for the discovery of a molybdenum deposit valued in excess of US$ 100 million (Tello, 1988, p. 101; and Van Horn, 1986 p. 10). But Dreyfus and Dreyfus (1984, p. 230) paint a different picture:

In reality, the PROSPECTOR program was given information concerning prior drilling on Mount Tolman where a field of Molybdenum had already been found. (authors' emphasis)

Indeed Dreyfus and Dreyfus go on to point out that all PROSPECTOR did was to map out the undrilled portions of the site using the drilling data already gathered and to suggest the extent of the ore deposit. Furthermore, while the ore found was very valuable, the geologists decided that it was too deep to be retrieved economically (Dreyfus & Dreyfus, 1984, p. 230).

The example of PROSPECTOR raises some very challenging questions and points the way towards a critical evaluation of the role of expert systems in real world environments, especially Planning. This is something which will be discussed in more detail in Chapter Six but it should be pointed out now that such unrealistic claims of expert system performance do not necessarily mean that planners should ignore the potential for expert systems applications in their sphere, merely that they should steel themselves against the kind of "AI Euphoria" which has been so prevalent in other fields: most notably the U.S. defense industry which has invested heavily in AI research (Dreyfus, 1987, p. 41; Dreyfus & Dreyfus, 1984, p. 221; and Rosenberg, 1986, pp. 168-172).

41
A Sample of Expert Systems in Planning

Since the mid 1980s there have been several Planners and Planning related professionals, as well as academics, involved in developing Expert Systems for Urban Planning domains and other related fields. By far the majority of these have been developed as ongoing research projects in order to test their performance and to see how planners react and adapt to them.

URBYS

URBYS, developed by Emile Tanic for the Municipality of Fort-de-France, Martinique, represents one of the first attempts in applying AI and Expert Systems research to a planning sphere (see Tanic, 1986). URBYS is a hybrid 'intelligent' urban information system comprising an expert system front end and a relational data base containing information and statistics on population, housing, land, infrastructure, etc. The expert system's knowledge base contains production rules which are linked to the data base via an interface. These rules determine how the data base is searched and how conclusions are reached. However, whether a rule is triggered or not is dependent upon variables contained in the IF portion of the IF--THEN statements and the values of these variables are contained in the relational data base (Tanic, 1986, pp. 140-3).

It is unclear from Tanic's article exactly how URBYS would be used in a Planning Department, for example. Tanic does argue, however, that Planners spend an excessive amount of time (he estimates approximately 60-80%) researching and organizing information concerning various planning issues and development proposals. Given this situation, it appears that the purpose of URBYS is to speed this handling and organization of information (Tanic, 1986, p.
135) and to make it easier to analyse and draw conclusions from the large amounts of, sometimes conflicting, information collected from censuses, surveys and the like (Tanic, 1986, pp. 139-40).

ADAPT

ADAPT (A Decision Aid Planning Tool) is a knowledge-based Decision Support System (DSS) which has been developed as a demonstration project for assisting Australian land-use planners in drawing up zoning schedules (Davis & Grant, 1987, pp. 60). Decision Support Systems have developed out of the Management Informations Systems (MIS) developed in the 1970s and:

... represent a point of view on the role of the computer in the management decision making process. Decision support implies the use of the computer to:

1). Assist managers in their decision processes in semi-structured tasks.
2). Support rather than replace, managerial judgement.

As Davis and Grant point out "The major task faced by planners when drawing up any land-use plan is to assemble information from diverse sources (for example, special interest groups, political parties, scientific predictions, personal experience) and make decisions that best satisfy the multiple goals advocated by these sources." (Davis & Grant, 1987, p. 55). Algorithmic models of such decision-making processes have been built in the past, witness the whole field of Operations Research, as well as Systems Theory, but by and large these have met with little success because of their grounding in the positivist tradition. ADAPT is an attempt to apply the flexibility, uncertainty-handling, and explanatory facilities of expert systems to the
decision making processes that planners frequently employ. The system is centered on three premises about expert systems which I feel are essential to the long-term development of successful applications in the planning sphere:

1) While the simpler parts of most decision making processes can be 'automated' using expert systems, planners will still have to resolve the more complex problems which arise. It is highly unlikely that expert systems will ever be able to replace human judgement on planning issues completely.

2) Despite 1. above, the expert system should act to support the planner in evaluating these complex issues.

3) The reasoning processes used by the expert system and the planner in making decisions should be transparent, i.e. visible, to other planners, professionals, community groups, and the public at large. (after Davis & Grant, 1987, p. 60)

ADAPT is written in FORTRAN-77 and so its production rule structure is rather restricted at present. The rule base contains land use policies, in an IF--THEN format, which determine how land-use activities in specific planning areas are assigned to land-use control categories such as "permit as of right" or "permit with Council consent". In general, the rules take the following format:

Assign or exclude (activity) to (control category) on all (planning units) where (criterion) is true. (Davis & Grant, 1987, p. 60)

Like URBYS, ADAPT is linked to a database containing statistical, geographic, and land-use information which forms the criteria to be met in the IF portion of the production rules. Davis and Grant illustrate some typical rules in their English equivalents:

Exclude pig keeping from 'as of right' category if distance to town is less than 5 km and effluent disposal capability is low.
Assign pig keeping to ‘as of right’ category if current zoning is rural 1(a).

Assign rural residential allotments to ‘permit with council consent’ if agricultural capability is class 3 or 4. (Davis & Grant, 1987, p. 60)

The knowledge base structure of ADAPT allows it to handle cases where information is missing or where conflicting alternatives arise and to inform the planner/user of these cases. ADAPT also keeps track of the decisions which are made at each stage of the zoning allocation process and the reasoning which underlies each decision. At present, however, the system cannot assess the reliability or uncertainty of the information which it uses, or of the results which it achieves. Furthermore, ADAPT cannot determine which of the various interest groups’ needs are met, or to what degree they are met, by the zoning scheme produced (Davis & Grant, 1987, p. 64). Finally, ADAPT is in the process of being developed and refined, and will be re-written in LISP or Prolog so as to rectify some of these deficiencies and to improve the overall performance of the system as a DSS tool for Planners. For a more detailed discussion of ADAPT and the stages of the decision making process which it goes through see the rest of Davis & Grant’s paper (1987, pp. 53-66).

Other Systems of Interest

There are a variety of other expert systems tools in both planning and planning-related fields. To discuss these in any kind of detail would require much more space than is available here and so only a brief, and thus incomplete, listing of other noteworthy systems (and the associated references) will be made.
ESMAN

ESMAN stands for Expert System for Manufacturing Site Selection and has been developed as a simple demonstration system by Suh et al (1988) at the Department of Urban and Regional Planning, University of Illinois. ESMAN has been designed to assist state and local planners in assessing and selecting suitable sites when approached by foreign investors wishing to establish manufacturing plant(s) in the U.S.. ESMAN is written using Texas Instruments’ LISP-based shell Personal Consultant Plus (see Tello, 1988, pp. 279-93 for a description of this software) and is a rule-based system similar in operation to URBYS and ADAPT. For more information on ESMAN see Suh et al (1988, pp. 239-52).

GRANT ADVISOR

GRANT ADVISOR is a very simple, rule-based, expert system designed to assist junior planners at the Tayside Regional Planning Department in giving out initial information to the public concerning the range of government grants available to local businesses. The system operates interactively and asks the user a series of questions to determine the specific needs of the person making the enquiry. The answers to these questions are then applied to the various rules contained in the system and the rules that are triggered determine which portions of the knowledge base are searched for grants relevant to the user’s needs. In this way, only grants appropriate to the user’s needs are selected, unlike a traditional database solution which would also present grants irrelevant to the user. For a more detailed discussion of GRANT ADVISOR see File & Oldfield (1988, pp. 29-30).
OTHER SYSTEMS

Other expert systems of interest are JESSE, which models aspects of the decision making and planning processes involved in Japanese national energy policy, in relation to national and international threats to Japan’s energy security (see Goel et al, 1987, pp. 178-87); HERMES (Heuristic Emergency Response Management Expert System), which is an experimental rule-based system for assessing emergency situations (caused by hazardous materials such as liquid chlorine), estimating hazard levels, and suggesting plans of action for dealing with the emergency such as containing the material and ensuring the safety of those nearby (see Clark et al, 1988, pp. 18-21); an expert system (built using the LEONARDO expert system shell) designed to assist planners in evaluating various development applications (see Leary & Rodriguez-Bachiller, 1988, pp. 26-8); and, a rule-based system developed by Hammond (1986, pp. 107-16) for representing the regulations of the U.K. Department of Health and Social Services and used for determining benefit entitlements. This last system is similar in concept to, but is more advanced than GRANT ADVISOR and uses APES (Augmented Prolog for Expert Systems) as the development environment (for a discussion of APES see Tello, 1988, p. 115-7).

Research into other expert system applications for planning and planning-related fields has also been carried out or is ongoing in other areas (see Barath & Futo, 1984, pp. 135-54; Davis & Clark, 1989, pp. 1-18; Leary & Rodriguez-Bachiller, 1987, pp. 198-210; Diamond & Wright, 1988, pp. 205-14; Dickey et al, 1986, pp. 251-71; and Leary, 1988a, pp. 383-98). While research into expert systems for urban planning seems to be most advanced in the United Kingdom (especially at Oxford Polytechnic) and in Australia (primarily at CSIRO), there has been some interest in Canada with some research being carried out in the Department of Urban
& Regional Planning at the University of Waterloo. Unfortunately, it has not been possible to obtain any information on this research. By far the majority of Canadian expert systems research to date has been led by Transport Canada which has organized several major seminars and workshops on the subject (see Transport Canada, 1985 & 1986). Under their direction, and the direction of the Ontario Ministry of Transportation and Communications, several expert systems have been built although these are aimed primarily at engineering problems and infrastructure maintenance. For a review of these see Gomi (1987, pp. 24-42).

**Application Areas in Urban Planning**

There are a number of areas within the planning process where expert systems applications can be fruitfully developed. Davis et al base their discussion of application areas on a definition of planning as a "... rationality-seeking activity ..." but one in which "... problem solving and decisionmaking (sic) ... is ... a continuous process which takes into account our uncertainty about knowledge of the environment, of community values, and of decisions in related fields. Finally, we view the process of ... planning as one of accommodating learning from consultation and public participation." (Davis et al, 1987, p. 241)

Despite the provisos included above, the authors remain squarely within the traditional planning paradigm and break the planning process down into six stages, each of which presents opportunities for expert systems applications: 1) Goal Identification; 2) Design of Strategies; 3) Evaluation and Selection; 4) Devising Tactics; 5) Implementation; and 6) Monitoring. I do not propose to defend the rational paradigm here since it merely provides an organizational tool around which a discussion of expert systems applications can take place. For discussions of traditional and alternative planning theories see Hudson (1979) and Friedmann (1987). Most of
the potential applications in these six categories require that much more refined expert systems techniques be developed in order to be able to deal more effectively with the open ended nature of planning in general (planning does not take place in a closed system). Nevertheless, rudimentary tools have been developed to assist experts in representing and organizing knowledge in the areas of natural language and story understanding, automated knowledge elicitation, as well as problem solving in simple environments (see Davis et al, 1986, pp. 244-7). As the authors point out, these techniques, if suitably modified, could be of assistance to planners and community groups in eliciting their goals.

We have already seen an expert systems implementation (JESSE) which assists energy policy planners in developing, refining, and selecting plans of action. Systems like JESSE, as well as Decision Support Expert Systems (see Dickey et al, 1986) and the others described above, illustrate that expert systems applications could be of significant assistance over a large portion of the planning process - namely plan formulation, evaluation, and selection. However, the existence of these systems means merely that rudimentary systems have been developed in domains which mirror elements of the planning process and thus in theory could be extended to planning. It does not, of course, mean that they should be so developed, and any attempts in this regard would require that planners address more critically what the role of the computer should be in planning, especially vis-á-vis contemporary planning theories and alternatives to the traditional planning paradigm.

Expert systems aimed at goal elicitation, plan formulation, plan evaluation, and plan selection in an open-ended and complex field like planning (i.e. truly comprehensive decision support tools) are many years from being realized. Certainly, using current expert systems and
AI techniques, it appears unlikely that such systems will ever be achieved barring a fundamental shift in hardware performance and/or expert system development philosophy. Nevertheless, there are elements of the planning process where current expert systems techniques are sufficient for providing useful workable tools within the next five to ten years, possibly sooner depending on the complexity and subtlety of both the systems developed and the domain in which they are developed.

While it appears unlikely that expert systems have a role to play at the implementation stage of the planning process, since implementation requires actions and not decisions (Davis et al, 1986, p. 249), the example of ADAPT as well as research undertaken at Oxford Polytechnic (see Leary, 1988a & 1988b; Leary & Rodriguez-Bachiller, 1987) illustrates that development control (what Davis et al call "Devising Tactics") is an area where useful expert systems tools can be developed quickly and effectively although, as yet there are no operational systems in place (nevertheless work is progressing on a number of fronts).

The monitoring of plans and development control policies is another area where the rule-based/pattern matching methodology of current expert systems can be fruitfully applied in evaluating the expected consequences of policies or plans against those actually realized. Monitoring expert systems which monitor in real time (i.e. they collect information immediately from sensors, computer links, etc.) have already been developed in industry and elsewhere (see Davis et al, 1986, p. 250). Similar systems in planning would not require a real time component and thus would enjoy a considerable savings in terms of complexity, size, and expense, although the process for selecting the indices and criteria for comparing normative and real results would need to be thought out clearly.
An alternative way of evaluating potential application areas is to look at the types of tasks which expert systems themselves can perform and see where these overlap the planning process. Ortolano and Perman (1987) assess expert systems from just such a perspective and have identified the following tasks: Interpretation; Diagnosis and Prescription; Design and Planning; Monitoring and Control; and Instruction. It should be clear immediately that some, if not all, of these categories mirror Davis et al’s classification of the planning process. Diagnosis and Prescription - "inferring malfunctions from observable data and prescribing remedies" (Ortolano and Perman, 1987, p. 100) - are both involved in issue identification, goal formulation, and plan preparation. Design and Planning - designing "... the form or arrangement of objects and actions under given constraints." (Ortolano and Perman, 1987, p. 100) - is similar to site planning, urban design, and site selection. Monitoring and Control - "... comparing observations to expected outcomes and governing the behavior of the monitored system." (Ortolano and Perman, 1987, p. 100) - are both elements of plan reviews, planning analysis, forecasting, and development control. And finally, both Interpretation - "... inferring situation descriptions from data." (Ortolano and Perman, 1987, p. 99) - and Instruction - "... expert systems that help novices understand concepts or learn to perform tasks." (Ortolano and Perman, 1987, p. 101) - are involved at every level of the planning process and planning activity.

Under each of these five headings, Ortolano & Perman identify specific tasks and areas of application where expert systems have been or could be developed to assist planners. These will be summarized briefly below:
Diagnosis & Prescription

Expert systems have been developed in project management to identify problems and constraints.

Expert systems have been built to assist in the operation and maintenance of infrastructure (roads, sewers, utilities, etc.) (see Hadden & Hadden, 1987, for a discussion of an advisory expert system designed to help the Austin/Travis County Health Department assess sewage facility permits.)

Design & Planning

Expert systems have been designed for site analysis, site selection, and habitat management. (recall ESMAN)

Expert systems have been used to help select, calibrate, and interpret the output from complex mathematical models. (i.e. they could be applied to the urban and regional models currently used by planners.)

Monitoring & Control

Expert systems have been proposed for climate control within buildings.

Several research projects are currently under development for the purposes of zoning administration and development control.

Interpretation

Expert systems are being linked with existing data base management systems (DBMS - see Rosenberg, 1986, p. 83-6) to provide ‘intelligent data base tools’ for assisting professionals in interpreting data. (recall URBYS).

Expert systems have been developed to assist in evaluating legal information and could be applied to land-use law in the same way. (see Berkowitz & Strzepek, 1987, for a description of a legal expert system designed to help administer provisions of the 1982 U.S. Reclamation Reform Act.)

Expert systems are being built to assess if development proposals meet the necessary zoning and land-use regulations.

Expert systems could be developed to determine the amount of property damage caused by natural disasters.
Instruction

Expert systems have been built to tutor and train junior staff and novices for specific tasks such as equipment troubleshooting and maintenance.

(N.B. for a more detailed discussion of these application areas and for information on the kind of research which has been carried out, see Ortolano & Perman, 1987, pp. 99-101).

Cullen (1986, pp. 249-50) has also addressed the issue of expert systems applications within planning and identifies three domains in which such systems could be developed, namely: Spatial and locational analysis; Development control and planning law; and environmental impact assessment. Concerning the latter there has already been some work completed by a local consulting firm in Vancouver (Environmental & Social Systems Analysts Ltd., Office tour, June, 1990) but for a more detailed review of research in this area see Page (1989). Although not present in Cullen’s review, I would also include economic impact assessment on this list since current techniques fail to handle the qualitative values which invariably emerge and which are a central element in trying to determine the impact of a project upon quality of life.

Characteristics of Tasks suited to Expert Systems Solutions

Obviously, expert systems, like any other tool a planner uses, are useful for solving some problems or tasks and not so useful in addressing others. An analysis of the literature reveals several common characteristics of tasks which are likely to benefit from and be suited to an expert systems solution. The key characteristics, in no particular order of importance, are:

* The task has many ways of being solved.

* The task cannot be solved using conventional algorithmic methods.
* Experts cannot always agree on how the task should be solved.

* An incorrect or late decision has significant consequences.

* Knowledge about the task is incomplete, uncertain, and subjective.

* Conclusions and recommendations about the task may be uncertain.

(see Townsend, 1987, p. 112; Van Horn, 1986, p. 48; and Sandra Cook, referenced in Dreyfus & Dreyfus, 1986, p. 120-1)

These characteristics describe features of the planning process in general as well as tasks within the planning process itself and so expert systems have significant possibilities with a range of planning domains. Within this range however, there are specific tasks which are more suitable than others as platforms from which initial demonstration expert systems can be developed. Choosing the correct domain for building a demonstration system depends upon achieving a balance between a complex 'meaty' problem and one which is manageable and relatively easily solved. Furthermore, knowledge about the chosen domain must be readily available, preferably in a structured short-hand form such as a procedures or regulations manual. The assistance of one or more knowledgeable experts can also be critical in developing the heuristics and rules of thumb necessary for the system to function effectively - this process of extracting the knowledge with the assistance of expert(s) - knowledge elicitation - was mentioned earlier in the chapter. In general the demonstration system should be built around a domain which is compact and manageable. "Of overriding importance is the need to identify a small and well bounded field of expertise with as few open-ended questions as possible." (Leary, 1988b, p. 11). From these characteristics and the requirements of domains in which a demonstration system may be built, Leary (1988a) has identified development control as the application
area in which initial demonstration systems are most likely to succeed since it is based upon clearly defined, structured, and codified, procedures and regulations (i.e. the Zoning By-law) as well as less structured information (i.e. design guidelines, etc.) and broader community and regional issues and policies.

Furthermore, development control is something which most municipal planning departments spend significant resources on. Irwin & Farrow's (1989) recent survey of sixteen planning departments in Southern Ontario revealed that on average anywhere from 51% to 68% of a department’s resources are spent on development and project-related activities while only 13% are spent on planning studies, reviews, and other 'long-range' matters. Perhaps more interestingly, 81% felt that their department was understaffed. Given these findings, albeit from a small sample size, development control expert systems could be invaluable in enabling planning departments to employ their resources more effectively and in helping planning staff deal more effectively with heavy workloads and more creatively with the control process itself.

It is not surprising then that most of the initial planning interest in expert systems has been directed towards development control (recall ADAPT, the work of Leary, Rodriguez-Bachiller, Cullen, and to some extent Tanic). And for the purposes of demonstrating expert systems here, development control will also be the domain used. This is something which will be discussed more in Chapter Four but first an evaluative framework needs to be developed from which expert systems applications in planning can be objectively and rationally evaluated.
We have seen in the last chapter that expert systems are being successfully applied to limited spheres within urban planning - primarily in the area of development control - although much work has also been done in the ecological planning and natural resources management spheres (see Robertson et al, 1985; Davis & Clark, 1989; Page, 1989). As was hinted in Chapter Two, however, the advent of expert systems will require that the very nature of planning and the planning process itself be reevaluated in order to see how expert systems can help improve the way planning is conducted. Any new technology has a powerful impact upon those who use it and on the way they approach various problems or tasks:

An implemented technology carries with it a powerful vision of the society in which it is to be used .... and, sometimes an obligatory plan for the way people will have to arrange themselves to use it. (Pfaffenberger, 1988, p.16)

On the other hand, new tools or technologies allow users to modify their work and behaviour. The motor car revolutionized personal transport but also altered the morphology of urban and rural areas in the same way that the computer has revolutionized work practices in nearly every sphere of human endeavour. Expert systems thus have revolutionary potential in the planning profession:

The successful application of expert systems to social sciences problems may bring about fundamental changes in the ways in which the social sciences are conducted. It may lead to a reappraisal of the extent and scope of problems that can be tackled in the social sciences. (Brittain, 1986, p.117)

The advent of any new technology, though, requires that some thought be given to what we are trying to achieve as planners and to what kinds of tools are needed to support these tasks. Developing such tools without a clear normative framework upon which to base evaluations
could lead to an empirical "hit and miss" approach to research and to the development of tools which hinder rather than help the planning process.

Contained within technology are powerful pre-understandings, social assumptions, cultural biases, value statements, and other prejudices. Technology is neither independent nor a deterministic manipulator of society. That is to say "technology is socially constructed" (Pfaffenberger, 1988, p.16) and therefore to decide how and what aspects of technology are to be used requires an evaluation of social behaviour and social relationships. In the context of planning this equates to an evaluation of the role of the planner and the nature and form of the planning process itself.

Robin Thompson, the 1990 President of the Royal Town Planning Institute has, in his Presidential statement, developed a normative framework for the planning profession as a whole and it is worth repeating briefly so that the overall goals of planning in general are not lost in the dazzle of expert systems technology. Thompson has argued that the role of planning should be to:

1). Enable and regulate economic growth.

2). Improve both the quality and equality of life.

3). Mediate conflict between economic growth and desires for a better and fairer quality of public life. (Thompson, 1990, vol.76, no.3, pp. 11-2)

To achieve this, he has put forward six criteria by which the planning process and the planner’s role should be evaluated:
1). STRATEGIC  Planners should develop regional and strategic long range plans.

2). EFFICIENT  Planning should be "cost effective and offer a good standard of customer care". It should also be able to demonstrate the value it adds to the development process (be the value economic, environmental, or social).

3). ENABLING  Planning must be entrepreneurial, flexible, cooperative and imaginative. It should enable the improvement of our society and the environment.

4). INVOLVING  Planning should be "open and consultative" and planners should seek input from all segments of society. Planning and decision-making should be decentralized.

5). NEEDS-BASED  Planning should assist in reducing the rift between the haves and the have-nots as well as the imbalances between regions and neighbourhoods.

6). GREEN  Concern for the overall quality of life and environment should pervade every element of the planning process. Development should be sustainable. (Thompson, 1990, vol.76, no.3, pp. 11-2)

I do not propose to enter into a discussion on the theoretical or epistemological merits of these six goals, and elsewhere I have indicated references on the subject of planning theory and the role of the planner. Thompson's goals for planning are presented merely as an organizational tool indicating the overall objectives toward which support tools should be directed and to give an idea of the overall thrust and direction which the planning profession should take. On a general philosophical level, however, there can be little debate concerning the validity of these six aims since they attempt to balance pragmatic reality and normative idealism. They also point the way towards suitable criteria for evaluating new technologies and for ensuring that such new tools support the goals and aims of the planning profession as well as the role of the planner.
We should not forget the central aims of planning in striving for a fair, just, safe, sustainable, society and environment when developing technical tools for assisting the planning process. If planning is to achieve these fundamental goals then, from an epistemological point of view, problem solving and decision support tools in planning must also be broadly designed with the same fundamental ends in mind. As Keen & Morton (quoted in Rosenberg, 1986, p.86) have noted with regard to decision support systems:

Decision support implies the use of the computer to:

1). Assist managers in their decision processes in semi-structured tasks.
2). Support rather than replace, managerial judgement.
3). Improve the effectiveness of decision making rather than its efficiency.

It thus follows that computers should be used in such a way as to allow planners to become more involved in the planning issues themselves rather than in the computer. In other words, the computer is a means to an end not an end in itself (after Pfaffenberger, 1988, p. 23). On a somewhat simplistic level, if the computer encourages this involvement then it can be seen as a positive influence otherwise, it is a hindrance. Thus, any evaluations of new technological tools must take place from the overriding point of view that tools should support planners and the planning process. This has not always been the case, as Cullen (1986, p.241-2) has noted and it can be argued that up until quite recently planners have been developing tools without any clear reference to a normative standard. As a result, planning theory, in borrowing ideas from operations research, cybernetics, systems theory, and others, has been molded to meet the requirements of these methodologies rather than the other way around (see Leary 1988a, p.383). Consequently, the traditional positivist tools which have dominated the social and regional
sciences to date have also infused planning. These tools include nearly all of the quantitative and procedural software as well as most of the generic software packages which are available today and which are in use in most planning departments (see Gar-On Yeh, 1988 and Bardon, 1988). Such tools and the techniques they use have been harshly criticized in recent years since it has been argued that they suffer from an "underlying technical malaise" on the grounds that:

1). They yield at best partial models of social process and are therefore, when put together with other similar devices, likely to lead a programme of investigations towards global inconsistency;

2). They are mechanical in form and function and so tend to build discretion-inhibiting inflexibility into any such programme;

3). They are data intensive, and are thus often incapable of handling information gaps, varying levels of uncertainty, and the achievement of less than a ratio scale of measurement [i.e. they cannot handle qualitative distinctions];

4). They tend to be functionally myopic, oriented more or less exclusively towards predictive applications;

5). They are operationally opaque, and so are hard not only to use but also to interpret and teach. (Cullen, 1986, p.241)

(For an even lengthier indictment of positivist tools see Macgill, 1986, p.1429-30)

Cullen has also pointed out that such techniques were based upon mechanical models of causation directed towards addressing the question of "Why?" whereas planning should equally be concerned with questions of "How?" since "... planning analysis ... [should] ... explore the impacts of complex structural relationships upon social processes ..." (Cullen, 1986, p. 242). Obviously, positivist methodologies are clearly not 'ideal problem solving or decision support tools'. Nevertheless, Cullen's critique hints at possible solutions to the problems he raises since an ideal tool would be diametrically opposite to the flaws outlined above.
Such epistemological and practical concerns would be irrelevant to planners in a Utopian society since they would be faced either with perfect and plentiful information or, should the information be imperfect (i.e. incomplete, uncertain, unverifiable, etc.), with the ideal support tools necessary for supporting their needs in planning analysis and policy formulation. In the real world, however, such luxuries are impossible due to the complexity and interconnectedness of information itself and to the open nature of the planning process. However the concept of a "Utopian Planning Tool" is useful in guiding the development of real world support tools and in developing a normative framework which outlines an "ideal problem solving and decision support tool".

Towards an Evaluative Tool

Macgill (1986, p.1430) has argued that "... different styles of model (and models of a given style) ... [including expert systems] ... have rather different qualities and should be judged in particular terms. ... Nevertheless, standards can rightly be spelled out as an institutionalized heuristic or normative framework to guide and evaluate analytical practice". This is the intent of the framework presented here.

Ideally, any model should be comprehensive and thus equally successful in a variety of areas. This is patently not the case, however, with the majority of models, including expert systems, since they are usually aimed at specific roles or domains. Macgill has developed seven questions for use in judging the quality of any model and these have been rephrased to form the core of the evaluative framework presented here. This framework has also drawn upon Cullen (1986), as well as expert systems literature in general but specifically Bramer (1984), Dalenoort

The advent of expert systems has brought with it a variety of claims concerning the performance of computers and software tools which operate on artificial intelligence principles. There is much debate amongst AI theorists and researchers alike concerning the validity of these claims (see Chapter Six). While many of these have yet to be proven conclusively, it is nevertheless possible to use them as yardsticks by which the performance of expert systems themselves can be judged, particularly with regard to the representation of knowledge and the handling of uncertain or incomplete information.

The Evaluative Framework

The framework presented here has been designed to be applicable to a wide variety of computer models and tools but will be used specifically to evaluate the demonstration system developed and described in Chapter Four.

Any problem solving and/or decision support tool should exhibit the following characteristics. Most of these are self explanatory but justifications for several of the criterion are given where necessary:

1). The output of the tool should be A) Consistent, B) Suitable, and C) Relevant to a particular policy content or planning problem. (Macgill, 1986; Phillips, 1986)

2). The accuracy of results, as well as the way results are achieved, should be recognizable. The tool must be able to substantiate and explain the results achieved and users must have confidence in both the tool’s process as well as its output (Macgill, 1986). In other words, the tool’s internal structure (i.e. the model’s assumptions, policy statements, rules of thumb,
etc.) and method of functioning should be transparent to the user (Cullen, 1986).

This criteria is designed to prevent the development of tools which appear to operate in some magical "black box" fashion. For a discussion of "black box" techniques, problem solving methods for planning in general, and the similarity between research efforts in both Planning and AI see Leary (1988a, p.383-8).

3) Input must be A) Accurate, B) Valid, and C) Of good quality. (Macgill, 1986). This is another way of phrasing that standard computing adage: "Garbage in = Garbage out."

4) The tool should have a solid epistemological grounding (Macgill, 1986; Brittain, 1986). It should be theoretically and methodologically valid.

5) The tool should be highly interactive and act as a conduit for communication between "expert" and user, as well as between users, regardless of the user's ability (Macgill, 1986; Collins et al, 1985).

6) The tool should be easy to learn, use, understand, and interpret, for a range of users from novice to expert, i.e. it should not be operationally opaque (Macgill, 1986; Cullen, 1986; see Robertson et al, 1985 for an example of this).

7) The tool should be rooted in both the context of the problem as well as the context of our "... cognitive heritage, biologically as well as culturally." (Dalenoort, 1986, p.89; see also Sloman, 1985 and Macgill, 1986)

This criteria is in complete contradiction to the traditional notions of systems analysis, rational planning, and other attempts to create general problem solving and decision support tools which were so prevalent in the 1960s (see Leary, 1988a). It is also a reaction to the attempts of the AI industry to create universal problem solvers, most of which failed.
To these we can also add criteria based upon the antitheses of Cullen's critique of positivist tools; that is to say, ideal problem solving and decision support tools should also exhibit the following:

8) The tool should not be data intensive but must be able to handle a wide variety of information and expand upon what has already been included in the database. It must be easy to update and to modify.

Planning issues and problems consist of a huge range of information types which can be decomposed into two primary categories: quantitative and qualitative. Numerous techniques have been developed for the former but there have been few truly useful methods devised for the latter. The problem of qualitative information is further compounded by problems associated with handling uncertainty as well as incomplete or missing information. Qualitative and uncertain information are frequently elements of planning issues, so for any tool to be valuable and be able to push the bounds of planning analysis beyond the capabilities of current tools (and thus improve planning practice), it must be able to handle the whole spectrum of information types.

9) The overall inferencing system should be independent of the representational schema used to express data, thus providing a consistent means by which several systems could be integrated. (Cullen, 1986, p.242)

As noted earlier, the separation of the inferencing or processing system from the data (information) structures increases flexibility and can allow several databases, each focussing on a specific problem domain, to be
combined to produce an integrated support environment using the same inferencing system.

10). The tool should be highly flexible and allow the user discretionary freedom as well as freedom of purpose (i.e. the freedom to solve a range of problems).

That is, the tool should not be technique-specific but should use a variety of problem solving methods and strategies so that should one approach fail there are alternatives to fall back on. Nor should the tool be functionally-specific such that it can only perform one kind of task (e.g.: prediction or generation - see Macgill, 1986 for a breakdown of modeling tasks)

Finally, given that these criteria have been developed from the expert systems literature as well as from reviews and critiques of positivist techniques, it should be no surprise that by and large the positivist, procedural tools currently in use will "fail" to meet most of the criteria when evaluated using this framework. If the claims of AI and Expert System proponents are borne out, then expert systems and other AI based tools should withstand an evaluation using this framework. This does not mean that all existing positivist tools in operation should be discarded in favour of Expert Systems based solutions since that would be sheer folly. It is highly unlikely that all of the claims about expert systems will come true, and therefore, the prudent planner should look to establishing a range of tools, both traditional and AI-based, as well as to augmenting existing tools with expert systems. The evolution of hybrid systems (i.e. systems which draw on a combination of both traditional as well as AI-based techniques) should also not be ignored since the emphasis should be on developing useful Decision Support Systems and not on the specific technology used, be it AI- or procedurally-based.
We will now turn to developing and describing a very rudimentary demonstration expert system in Chapter Four and in the following chapter (Chapter Five) will apply this framework to an evaluation of the system developed.
CHAPTER FOUR: A DESCRIPTION OF THE DEMONSTRATION SYSTEM - EUCLID

Most urban planning research into the feasibility of Expert Systems within the profession has, to date, been either highly preliminary in nature or, for the few embryonic systems which have been developed (largely in academic institutions), focussed around the domain of Development Control. The same cannot be said of regional and environmental planning where far more research has been carried out both in Canada and abroad, and where operational expert systems have already been developed [see Davis & Clark (1989), Page (1989) for concise summaries of the work that has been done]. Nevertheless, within the urban sphere Leary (1988a) has noted the potential for translating Design Guidelines into an expert system format, and has argued that design guidelines are in essence pre-structured knowledge bases in a written format. Leary & Bachiller's (1988) LEONARDO-based system, which assists in checking development applications, and Davis & Grant's ADAPT system (1987) also demonstrate that feasibility studies are already being undertaken within the domain of urban planning.

Yet curiously, despite all of this activity in the domain of development control, there appears to have been limited research in the use of expert systems for helping to administer Zoning By-laws; such a pervasive part of North American planning practice. This should not be surprising since most of the expert systems work to date has occurred elsewhere or in other fields. Indeed, North American planners seem to be just beginning to assess the overall possibilities although one or two studies dealing with zoning by-law administration do appear to have been conducted at the University of Waterloo. Unfortunately, it has not been possible to obtain any details on this research to date.
It seems curious that there has not been more research on translating Zoning By-laws into Expert Systems since such By-laws are well structured and organized, concise in nature, and, like design guidelines, are essentially written knowledge bases in that they contain a set of legal rules, regulations, and conditions. They thus conform with the types of domains that are recommended in the AI literature as being suitable for developing not too trivial and not too complex demonstration systems. Zoning By-laws also exhibit most of the characteristics of tasks suited to Expert Systems solutions:

- There are many ways a By-law can be interpreted.
- Zoning By-laws cannot be reduced to functions and algorithms since they require a substantial amount of judgement in evaluating the suitability of development proposals.
- Planners, developers, and the public are often in discord over how a proposal should be judged and whether or not a development application should be allowed.
- It is certainly true that an incorrect or late decision can have, or lead to, significant impacts upon both developers, planners, and others, not to mention communities and the viability of the municipality itself.
- Development proposals are presented in varying stages of completeness and may change as the application progresses. As a result, they may require scrutiny at various stages throughout the approval process and so there is a need for consistency between and within applications.

Perhaps a more persuasive argument for developing expert systems to assist in the assessment of development proposals is the fact that most planning departments expend vast resources to fulfill their development control function and provide over-the-counter services. Furthermore, this emphasis on development control often occurs at the expense of the long range planning and community planning functions which provide the policy framework from which development proposals are evaluated, particularly in municipalities without a long range planning division.
From a contemporary point of view, this imbalance in municipal planning functions is exacerbated by the increasing pressures of heavy workloads due in part to staff shortages and, in periods of economic prosperity, to increasing numbers of development proposals from the development industry. The City of Vancouver, for example, estimates that, on average, the time it takes for a development application to be processed is anywhere from 2 weeks to 2 months, or even longer, depending on the complexity of the proposal, issues which arise, and any studies which have to be carried out (City of Vancouver, Development Permits for Major Developments, Brochure #3, Oct. 1987).

Given what has been said in the preceding chapters and above, it should be clear that Development Control Expert Systems could provide planners with a significant weapon with which to combat these pressures. Such systems could assist both junior and senior staff members in administering Zoning By-laws and assessing development proposals more expeditiously and on a more consistent basis. As a consequence, they could ultimately free up staff and departmental resources, thus providing greater freedom and flexibility to do more long range planning and to increase community involvement in the planning process itself. Nevertheless, it should be pointed out that there could be serious negative consequences in introducing Expert Systems to the planning office. These will be discussed in more detail in Chapter Six.

At this stage, however, the suitability of using Expert Systems in the urban planning sphere and of creating systems, based upon AI techniques, which function as development control 'assistants' remains to be proven. Demonstration systems must therefore be developed in order to be able to evaluate more conclusively how the introduction of expert systems will affect municipal planning practice.
It is within the overall context of both this pragmatic, contemporary view of urban planning and the need to produce demonstration systems in the domain of development control that the system developed here - Euclid - is presented.

**Euclid**

Euclid is a simple backward chaining expert system built purely to test the feasibility of translating Zoning By-law schedules into a production rule schema as well as the practicability of providing planners with a tool to assist in the evaluation of development proposals. It has been conceived as a tool for assisting planning staff in their assessments of development applications. However, due to the limited programming experience of the author, and the limited time in which to do the research, Euclid, as an initial exploratory vehicle, represents only a portion of the features which would be necessary in a fully developed system. It also performs as a 'test case' for applying the criteria developed in Chapter Three. It has been developed using the Turbo Prolog 2.0 language and has been built around a portion of the City of Vancouver's current Zoning By-law. It is named, for obvious reasons, after the Village of Euclid, over which the U.S. Supreme Court upheld the concept of zoning in its landmark ruling of 1926 (see Babcock, 1966, p.4).

It should be clear that trying to translate an entire Zoning By-law into a production rule knowledge base is a formidable task indeed. The City of Vancouver's Zoning By-law alone defines over 55 separate zoning districts, not counting the Comprehensive Development Zones (City of Vancouver, Zoning By-law, September, 1989). Even the City of North Vancouver's (a small municipality) Zoning By-law identifies 23 separate zones, again not including their Comprehensive Development schedules which, by 1986, numbered 85! (City of North Van-
## Figure 4.2

### Zoning By-law Structure

<table>
<thead>
<tr>
<th>INTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTRIGHT APPROVAL USES</td>
</tr>
<tr>
<td>- accessory uses</td>
</tr>
<tr>
<td>- dwelling uses</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONDITIONAL APPROVAL USES</th>
</tr>
</thead>
<tbody>
<tr>
<td>- accessory uses</td>
</tr>
<tr>
<td>- dwelling uses</td>
</tr>
<tr>
<td>- cultural &amp; recreational uses</td>
</tr>
<tr>
<td>- institutional uses</td>
</tr>
<tr>
<td>- retail uses</td>
</tr>
<tr>
<td>- service uses</td>
</tr>
<tr>
<td>- utility and communication uses</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PHYSICAL REGULATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>- site area</td>
</tr>
<tr>
<td>- frontage</td>
</tr>
<tr>
<td>- height</td>
</tr>
<tr>
<td>- front yard</td>
</tr>
<tr>
<td>- side yards</td>
</tr>
<tr>
<td>- rear yard</td>
</tr>
<tr>
<td>- floor space ratio</td>
</tr>
<tr>
<td>- site coverage</td>
</tr>
</tbody>
</table>

### RELAXATION OF REGULATIONS

Source: The City of Vancouver; *Zoning and Development By-Law*; September, 1989.
Figure 4.3

Euclid: Program Modules

USER

SYSTEM/USER INTERFACE

EUCLID PROGRAM

TURBO PROLOG INFERENCE ENGINE

KNOWLEDGE BASE
- Outright Use Rules
- Conditional Use Rules
- Zoning Regulation Rules

INTERNAL DATABASES
- Zoneinfo
- Dimensions
- Answer Storage

RT-2 DATABASE
RT-4 DATABASE
RT-4N DATABASE
RT-5 DATABASE
RT5N DATABASE
RT6 DATABASE
In addition, a truly comprehensive development control tool would require the inclusion of the Parking By-law, the Health and Fire By-laws, as well as Development Permit regulations, and would need to be tied into a Geographic Information System or the administrative database system in order to keep track of development applications and their status. Therefore, in order to keep the workload within manageable limits, the scope of Euclid has been restricted to those Zoning schedules which govern Two-Family residential areas in Kitsilano, Grandview Woodlands, and Mount Pleasant - namely the RT-2, RT-4, RT-4N, RT-5, RT-5N, and RT-6 districts. Figure 4.1 delimits the geographical locations of these three local areas.

The six Zoning schedules are all organized in a similar fashion and are split into various sections as outlined in Figure 4.2. The schedules list those uses which can be approved outright and those which may be approved conditionally. In addition, they describe regulations governing heights, setbacks, floor space ratios, etc., which both outright and conditional uses must meet in order to be approved. The schedules also contain a few regulations which apply only to those uses which may be approved conditionally, as well as the zoning relaxations which may apply to specific zones and proposed uses. Sections 10 & 11 of the By-law have consciously been left out so as to reduce the amount of programming required.

Euclid draws on both this built-in organizational structure as well as the production-rule methodology to provide the framework around which the system is programmed. It should be noted that the production-rule formalism is not new to planning and resembles quite closely the AIDA (Analysis of Interconnected Decision Areas) technique which forms part of the Strategic Choice approach advocated by Hickling (1975, pp. 23-34) and others - particularly if AIDA is
Figure 4.4

The Planner/computer Interface

USER

Main Menu Window
- select action
- read in zoning specifications
- get lot dimensions
- select use

Select a Zone Window
- select a zone

Checking a Zoning By-law Window
- run query_the_user session

Response Window
- report results to the user
- return to main menu window

Error Message Window
- report input errors to user

75
used to generate a solution or decision tree. There is, therefore, strong support from both the Decision Making and AI literature in adopting this approach.

In its present form, Euclid is comprised of several modules, each of which has a specific function. Figure 4.3 illustrates the major program blocks and their relationship to one another. The planner/computer interface governs how the program interacts with those using the system and provides a vehicle for obtaining information from and returning results to the user. The knowledge base forms the bulk of the system and is built up from several program files which correspond roughly to the organizational structure of the Zoning Schedules, as shown in Figure 4.2 above. Thus, one file contains the rules governing outright uses, another the rules which evaluate conditional uses, and yet another which lists the relaxations permitted under various zone/use combinations. The rules which describe the specific zoning regulations are, by necessity, split into two files and the reason for this will be discussed later. Finally, six database files contain the numerical specifications for each of the six zoning schedules. With the exception of the database files, these modules are transparent to the user in the finished application since they are all contained in one big executable file or 'consultation shell' (Tello, 1988, p. 105). For those interested, the actual Turbo Prolog code for each of the modules is listed in Appendix 3 and the contents of each database file is listed in Appendix 1. Fairly detailed comments and predicate descriptions have been included in the code listings and it is not intended to duplicate those comments here. Readers should refer themselves to the Appendices since this may be beneficial in the discussion that follows.
The Planner/computer Interface

The interface between the planner using Euclid and the program is represented diagrammatically by Figure 4.4 and the programming code which represents this module is contained in the EUCLID.PRO file listed in Appendix 3. This interface determines what the user sees on the screen and how information is handled. The heart of the interface consists of several windows, one of which forms the main menu. Other windows are used for the query-the-user process and for displaying results. There are also several windows in which error messages or directions are displayed. While the concept of menus has several disadvantages in terms of user comfort (for example it can be very difficult to remember what level you’re at within the system when there are several sub-menus nested within each other) Turbo Prolog provides several standard predicates which make defining windows and choices very easy and so this is the method used.

Another important function of the interface is that it asks the user to select the zoning schedule which applies to the site in question. The user is also asked to select from a list of uses that use which matches the use proposed in the development application. Since Euclid can deal with six zoning schedules and over 25 uses, asking these two questions serves to narrow the system’s focus to only those questions which are relevant to the case being assessed. Furthermore, asking the user to select the appropriate zone precludes the need for Euclid to contain a geographic database or rules which link property addresses to zoning schedules (again purely in order to keep work levels within manageable limits), though these would be desirable features of a more sophisticated system.
The interface also contains predicates which gather information needed by the rules at various stages throughout the program. For example, information on the site and its dimensions, as well as specifications for the particular zone selected (e.g. floor space ratio values), need to be available to all rules which use them in their evaluations. Thus, the interface contains a predicate which asks the user to supply certain site dimensions and stores these dimensions in a database which can be called upon by any rule at any point in the program. There is also a predicate which, depending upon the zoning schedule selected, accesses the appropriate database file and loads its contents into another database which can also be accessed at any time. These two features remove the need to ask the user repetitive questions.

Finally, the interface contains the predicates which determine how the rules are searched. In a sense, these predicates form part of the inference engine in that they guide how the rules governing outright use, conditional use, and the regulations are to be dealt with. Again, in order for the search space to be narrowed, outright uses are checked first and only if the use is not allowed outright are the conditional use rules checked. Only if a use is allowed either outright or conditionally are the zoning regulations checked since there is no point in checking regulations if the use itself is not allowed.

The Inference Engine

While the interface determines where Euclid starts its evaluation and the order in which rules are searched, the inference engine still needs to be told how rules are to be evaluated and outcomes decided. Even though it is possible to create custom made inference engines, Euclid, for the sake of simplicity, employs the standard pattern matching process (called 'unification') which is an integral part of the Turbo Prolog language. This unification mechanism operates in
a backward chaining manner. Recall that backward chaining operates on the "A is true IF B is true" principle and that, in its simplest form, backward chaining can only handle TRUE/FALSE distinctions. As a consequence, Euclid contains no facility for handling uncertainty and determines the truth of a rule on the basis of answers supplied by the user in response to 'yes/no' questions or, in the case of questions asking for numerical input, on the basis of numerical values supplied by the user.

The Knowledge Base

Turbo Prolog’s unification process relies heavily on the knowledge base in determining the results of rule queries since the rules are linked to each other in a tree type structure. In the Turbo Prolog language, each rule consists of a HEAD and a BODY and generally takes the format:

(1) "HEAD":-"BODY" (note that ":-" is read as "IF")

Thus, using backward chaining, the HEAD of a rule is true if the elements forming the BODY of the rule are also true. Note that the BODY may contain more than one element and that in some cases the entire BODY of a rule must be satisfied in order for the HEAD to be true while in others there may be several options, only one of which needs to be true. The logical operator AND (represented in Turbo Prolog by the comma ",") is used to represent elements of the former rule-type while those of the latter type will contain at least one OR operator (represented by a semi-colon ';'). Note also that the bodies of some rules in Euclid are hybrids and can contain both AND and OR operators. In these cases, the OR operators act to delimit the various options which may be satisfied. All the elements which make up a particular option must prove true in order for that option to succeed. If the first option succeeds, the HEAD of
the rule succeeds, whereas if it fails, each alternative option is tried until the rule either succeeds or fails. For example the rule:

\[(2) \text{ check\_conditional\_app}(Zone,Use) :- A,B ; C ; D,E,F.\]

describes a rule which checks to see if a particular use is allowed conditionally in the zone selected. The rule contains three options, only one of which needs to be true in order for the use to be allowed. The first option is made up of two elements, A and B, both of which need to be true in order for option one to succeed and if it does so the rule succeeds. If it fails, Turbo Prolog tries the next option which has only one element, C, which is either true or false. If the first two options fail, option three is triggered and elements D, E, and F, are assessed. The rule structure described above is used quite extensively by Euclid in describing zoning regulations where there may be several alternative conditions. For example, the height of a building can be either so many feet or so many storeys high. Consequently, if a building fails the first it may comply with the second.

Rules are linked to each other, and the search sequence is defined, by including in the BODY of the current rule the HEAD of the rule which follows it. Thus, any of the elements A - F in rule (2) may represent calls to additional rules. In such cases, Turbo Prolog has to go to that next rule and determine whether it is true before it can determine the outcome for the 'current' rule. In this way too, values can be passed from one rule to the next since variables can 'travel' with the HEAD of a rule. To illustrate this point, rule (2) above has associated with it two arguments. In the example shown, these arguments are two variables (in Turbo Prolog variables start with a capital letter) - Zone, and Use - the values of which are supplied by the
user at the beginning of a consultation session. Thus, if the user selects "RT-4" as the zone, '8' (Infill) as the use, Zone takes the value "RT-4" and Use becomes '8'. Once the values of Zone and Use have been determined they can be used, in conjunction with the rule itself, to transport the user's choices throughout the program.

It should be clear that there may be several versions of a particular rule since there are many possible zone/use combinations and each rule can only describe the conditions for a particular situation. Under such circumstances, this ability for variables to be carried with the HEAD of a rule becomes very useful. It is used to advantage in Euclid since it allows the HEAD of each rule to be used as a filter so that only those rules which apply to the zone/use combination being evaluated are triggered. For example the rules:

(3) \text{check\_outright\_app("RT-6",12) :- BODY A.}
(4) \text{check\_outright\_app("RT-4N",_ :- BODY B.}
(5) \text{check\_outright\_app(_,8) :- BODY C.}
(6) \text{check\_outright\_app(_,_) :- BODY D.}

represent four versions of the rule which checks to see if a given zone/use combination is allowed outright. Rule (3) defines the conditions which apply in a specific situation, namely when the zone in question is RT-6 and the use '12' (Two-Family residential). Only if the user has selected these two choices will the rule be triggered and the conditions contained in BODY A tested. Similarly, rule (4) describes the conditions to be met if the zone is RT-4N. In this rule it does not matter what the use is and as a result, if the zone is RT-4N, the conditions are tested regardless of the use chosen. Rule (5) represents the opposite situation and describes the conditions that
are to be tested if, and only if, the use chosen is '8' (infill). Finally, rule (6) is the same as saying that the conditions listed in BODY D are to be tested regardless of the Zone or the Use selected. This last rule serves as a catchall clause and behaves in a similar fashion to the ELSE component of a Pascal IF-THEN-ELSE statement.

The description of rule structures above applies mainly to the rules that check for outright and conditional uses and, in some cases, zoning regulations which are tied to a specific zone and use. By and large, most of the zoning regulations are not dependent upon a specific zone and/or use and, as a result, the rules that describe them have a slightly different format. This different format is used because these zoning regulations apply universally within a zone regardless of the use proposed and because some of them are common to several of the zoning schedules. Each of these regulations is represented by a specific rule. Consequently, there are regulations which describe height requirements, side yard requirements, floor space ratio requirements, site coverage requirements, etc.. In most cases, each zone has its own specific regulations and, rather than identify each by a separate name, a standard term is used in association with a number; this improves the readability of the code. For example, there are three different regulations describing front yard requirements and these are translated into the following rules:

(7) check_front_yard1 :- conditions.
    check_front_yard2 :- conditions.
    check_front_yard3 :- conditions.

The predicate 'check_rest_of_regs(Zone)' is then used to identify which regulations apply to which zone by listing the requisite rules in the body of the appropriate 'check_rest_of regs(Zone)' rule. Thus the rule:
lists those regulations that apply, regardless of use, only to the RT-4 zoning district. Note that
regulations which are shared among two or more zones are easily handled since each zone has
its own 'check_rest_of_regs' rule and thus the rule for a particular zoning regulation can appear
in more than one 'check_rest_of_regs' clause without creating errors in the way the system works.

In general, the regulation rules take the following format:

(9) check_regulation X :- condition 1;
    condition 2;
    condition 3;
    response message.

Item 9 above contains three conditions and a response message. Each condition consists
of three elements: 1) information supplied by the user; 2) a relational operator (<, ≤, =, ≠, ≥, >); and, 3) the zoning specification to be met. In addition, each condition needs to contain a
question which elicits the correct input from the user as well as predicates which are able to
access the databases and retrieve appropriate specifications. Some of the conditions require
information which has already been obtained as a result of queries relating to previous
regulations. Lot dimensions, for example, are frequently used in evaluating conditions and
asking for this information repeatedly would lead to user frustration and confusion. Consequent-
ly, where information supplied by the user is likely to be required by other regulations later on, it is stored in a special database so that the user does not have to be asked for it again.

Note that the final condition in the rule above is in fact a response message which automatically succeeds. This message provides Euclid with a way of telling the user which regulation the development proposal fails to meet since it is only triggered if all of the conditions within that rule, and above it, fail. Thus if rule (9) above describes a rear yard regulation, the message would say that the development proposed failed to meet rear yard requirements for the zone that had been selected and would then lead to another message saying that the use was not allowed either outright or conditionally.

The combination of the rules described above makes Euclid’s knowledge base quite expressive and it is fairly easy to produce rules for a specific regulation or use. However, the production rule technique also produces a rule base which is memory intensive, cumbersome, and makes it rather difficult to follow the order in which the rules are evaluated. Furthermore, Euclid cannot communicate using the English language and, as a consequence, cannot interpret user input nor provide responses on its own accord. The language used in the Zoning By-laws is quite complex and involved, since many of the regulations contain subtle and not so subtle variations. Consequently, since Euclid uses the standard pattern-matching processes of Turbo Prolog, the questions that it asks have had to be 'hardwired' into the rules as write statements and text strings. In addition, the responses and results that it applies have also had to be treated in this way and thus a fair amount of flexibility has been lost.
The Development Process

Initially, Euclid was built up as a single program containing rules only for the RT-6 schedule. This schedule was chosen as the early development case because it was the most complex of the six schedules and contained several regulations unique to the RT-6 zone. The first task involved translating the written statements into Turbo Prolog code. This process was complicated by the fact that the author was new to the Turbo Prolog language and had to learn how to program Euclid at the same time as trying to translate the zoning By-laws. As a result, a trial and error approach had to be employed and several weeks passed before an appropriate rule structure was developed.

It soon became apparent during the development period that in some cases a direct translation of a regulation into its Prolog equivalent would lead to errors in logic or errors in the way the system responded since Turbo Prolog treats the logical operators (AND, OR, NOT) in very specific ways. Usually, the problem arose with a particular condition of a regulation and was the result of a rule failing when the logic of the program required that it should succeed. The solution in these situations was to program the condition in its opposite logical form. Thus, if one of the conditions to a rear yard regulation, say, required that the rear yard had to be less than or equal to \( \leq \) 35 feet and the rule failed when logically it should have succeeded, the rule was re-written so that the rear yard had to be greater than \( > \) 35 feet.

Debugging these and other types of errors proved to be a time consuming task but once the rules for the RT-6 zone were operating more or less successfully, the user interface portion of the Euclid program was developed. As mentioned earlier, Turbo Prolog's built in window handling predicates made making the menu environment fairly straightforward. The difficult
portion of the interface consisted of developing the initial rules for governing how the knowledge base was to be searched and for eliciting key information from the user and storing it in several distinct databases. It is not intended here to discuss the intricacies of the way these rules work and for a more detailed discussion of the Euclid system reference should be made to the annotated Turbo Prolog code listings in Appendix 3.

To reach the stage where the RT-6 rules and the user interface were working fairly successfully took about 120 person-hours of programming effort. The next phase of development involved adding the rules and specifications for the remaining five zoning by-laws. Since the basic operating processes and rule structure of Euclid had already been determined, adding these by-laws consisted mainly of expanding the rule base to cover all the possible combinations required. However, in some cases, the addition of these rules conflicted with some of the existing rules and produced errors in the way the system had been designed to operate. These conflicts required quite an extensive period of testing and debugging to bring the system back to its previous level of proficiency and one of the principal modifications during this process involved the modification of the RT-6 rules so that they would only apply in cases where the RT-6 zone had been selected. Nevertheless, the rules and database facts for the five remaining zones, as well as the modifications to the rules governing the search strategy, only required about 50 person-hours of additional programming effort to incorporate them into the system.

The debugging of the program was complicated by the fact that the final program code approaches 60 Kb in size and could not, unfortunately, be compiled, in its entirety, into the space remaining in the Turbo Prolog development environment. (A weakness of Turbo Prolog is that it can only handle compiled programs up to 64 Kb in size and the compiled size of a program
is always larger than the size of the actual code - the size of the final compiled version of Euclid, for example, is almost 180 Kb!).

The solution around this problem was to debug the program modules a portion at a time. This in itself is a lengthy process since combinatorial explosion in concert with the numerous questions that the system asks leads to thousands of possible combinations; Euclid supports 162 zone/use combinations alone, not counting the variations in the user’s responses. Consequently, while each of the zone/use combinations have been put through a basic test, it has been impossible to check the outcomes of every single variation in user input.

While this 'debug a portion at a time' methodology allowed the essentials of the system to be tested and proven, it presented no solution to the problem of how to generate a stand alone executable file. Since the file could not be compiled in the Turbo Prolog environment, it had to be compiled externally. This necessitated breaking up the program into seven separate program files (these are the files listed in Appendix 3) and then linking them all together into a file which could be accessed by the Operating System of a personal computer. Breaking up the program required that portions of it be modified to accommodate the new structure and so portions of the new program had to be debugged yet again. Modifying the program structure and creating the executable file also proved to be time consuming and took, approximately 85 person-hours of further programming and editing effort.

The current version of Euclid is contained on one 360 K floppy disk and can be run from a standard IBM AT, or compatible computer. It consists of the main executable program, a Prolog error message file (PROLOG.ERR), and the 6 database files. Since the system relies on fairly straightforward and simple expert systems techniques, it runs very quickly and the user
is not left waiting for the next question to be asked or a result to appear. There is a noticeable pause, however, when loading the program from a floppy drive to the workspace and when the contents of the database file, corresponding to the zone selected by the user, is read into the system. These time lapses are reduced considerably if the system is installed, in its entirety onto a hard drive.

A Sample Run Session

In running Euclid the first thing the user sees is a message window stating that Euclid has been developed as part of the author’s M.A. Thesis research. Pressing the space bar removes this window and reveals the main system window. This contains a welcome message, the main menu, and a copyright notice.

The main menu has 5 choices and under the present version of Euclid only option 1 "Checking a Zoning By-Law" is operational. Options 2 and 3 have been included in the main menu to show how a moderately complete development control system would appear to the user and are at present 'cosmetic'. Option 2 "Find a zone for a Project" would allow planners to pose the question "I have a development project in mind, which zoning schedule is most appropriate?" while Option 3 "Modify or Add to the Database" would allow planners to update or modify zoning specifications. Option 4 is operational and allows the planner to move to the operating system (in this case DOS) to carry out file maintenance etc. and then, by typing 'EXIT', return to the Euclid main menu - a useful feature in large programs which deal with numerous files and where information changes frequently. Finally, Option 5 ends the session and returns the user to DOS.
Figure 4.5

Euclid System Flow Chart

1. Display Main Menu
2. Check the Zoning on a Property
   - get zone schedule
   - load zoning database
   - get lot dimensions
   - get use
3. Is use allowed in zone selected?
   yes
   no
4. check regulations?
   yes
   no
5. Is use allowed outright
   yes
   no
6. Does it meet outright zoning regulations for the zone?
   yes
   no
7. Is it allowed conditionally?
6. Does it meet outright zoning regulations for the zone?
   yes
   no
8. Report the use is allowed outright.
   END
9. Does it meet conditional zoning regulations for the zone?
   yes
   no
   continued on next page......
(10) Report the use is allowed conditionally.

(11) Are any relaxations possible?  
- no
- yes

(12) Are any relaxations possible?  
- no
- yes

(13) Report the relaxations that are available

(14) Report that the use is not allowed outright but may be allowed conditionally,

Ask: "Do you want to check for conditional approval?"

- yes
  - Goto (7) Above
- no
  - END

(15) Report that the use is not allowed, either outright or conditionally

END
To run Euclid, the user must select Option 1. Euclid then displays another window and gives an instruction to select, from the list presented, the Zoning By-law which applies to the site in question. The flow chart contained in Figure 4.5 illustrates the process which Euclid carries out as it evaluates a development proposal. Selecting a zone causes Euclid to read the contents of the appropriate zoning database into the internal database that stores this information for use later on.

Once this has been done, a message appears asking the user to enter the dimensions of the lot. If the lot is rectangular Euclid calculates and displays the site area, otherwise it asks the user to enter the actual area. Once the lot dimensions have been entered they are stored in another internal database designated for this purpose, again so that lot dimensions can be accessed later on without having to keep asking the user for the same information repeatedly.

Next, the user is asked to select the use which is closest to the use proposed, again from a list provided by the system. This list displays all of the uses which are allowed in the six zoning districts but the user can also select 'other use' if the use proposed does not appear.

Euclid then takes the zone/use combination selected and begins to evaluate it, first by checking to see if the use is allowed outright in the zone selected. When Euclid needs input from the user to substantiate a rule (i.e. to prove that the use proposed meets a certain requirement) it will ask a question and indicate what type of response it expects. Every time it needs information that has not already been provided it will ask a question. If the response provided is not what Euclid expects it will, in some cases, catch the error and ask the user to try again. Providing this error trapping facility is made more complicated in Turbo Prolog since the predicates which read user input only work for certain data types, thus entering a letter when an
integer is expected can cause the system to behave in a peculiar manner. While it is virtually
impossible, especially in a demonstration system like Euclid, to guard against all possible input
errors, it is unlikely that such an input error will permanently damage the program and in such
circumstances all the user has to do is restart the system.

If the use itself is allowed outright, Euclid then begins to evaluate specific elements of
the development proposal against the particular zoning regulations for outright uses. Regulations
that pertain to the specific zone/use combination selected are checked first, followed by the more
general regulations. Again Euclid asks the user for information, such as the number of dwelling
units proposed, the width of the side yards, etc. if that information has not already been supplied.
For uses which do not imply a building as part of the proposal (e.g. a park), Euclid establishes
whether or not a building is proposed. If such a use does not contain a building and yet is allowed,
either outright or conditionally for the given zone, it is permitted and a message to that effect
appears in the window entitled "Response to a Zoning Query". If, on the other hand, a building
forms part of such a use, Euclid proceeds to evaluate the use against the zoning regulations. For
uses which imply a building (e.g. a multiple dwelling use), Euclid moves straight into applying
the zoning regulations once the use itself has been checked for outright or conditional approval.

If the development project meets all of the regulation requirements for outright uses in
the zone selected, the use is permitted outright and a message to this effect appears in the response
window. If the use fails any one of the regulations, Euclid assumes the development proposal
fails to meet the necessary zoning regulations. The message it provides contains the reason why
the proposal fails and it may, in some cases, also identify any relaxations which may be applied
for and/or list any guidelines which should be followed. Whether a project is permitted outright
or conditionally depends upon which of the two categories the use proposed falls into. If the use proposed is not allowed outright, Euclid informs the user of this fact and asks if they wish to check for conditional approval. If the user wants to check for conditional approval (since the proposal may still be permitted provided it meets use requirements and the zoning regulations for conditional uses) Euclid will then check to see if these requirements are met, in the same way as for outright uses, and will carry out the same reporting procedures as described above.

If the use itself is not allowed either outright or conditionally, the development proposal may still meet the physical zoning regulations and so may still be approved by Planning Department staff since the Zoning By-laws themselves and the uses they allow are not cast in stone. Thus a use not expressly allowed in the By-law may still meet the height, setback, site coverage requirements, etc. and be allowed at the discretion of the Director of Planning. Euclid, therefore, attempts to deal with these contingencies in its operating structure and search strategy, and reports to the user when such situations arise. In addition, the current search strategy occasionally results in a line of questioning which, in some instances, appears confusing. For example, if a use is allowed outright but fails to meet the zoning regulations for outright uses, Euclid will ask the user if they want to check for conditional approval since the use may still be allowed conditionally. In most instances though, if the user selects this option it is highly likely that, while the use may be allowed conditionally, the application will fail to meet the regulations it failed under outright uses. This problem arises because most of the zoning regulations apply both to conditional and outright uses. Furthermore, the structure of the zoning schedules themselves forces this search strategy to be used in Euclid. In these situations this second go round is therefore irrelevant.
Conclusions

As an initial demonstration exercise, Euclid shows that Zoning By-laws can indeed be translated into a logical program and run on a computer. What Euclid does in effect is make it easier for the planner to evaluate a development proposal since one evaluation takes a couple of minutes at most and Euclid asks only those questions which are relevant to the site being studied. This feature distinguishes Euclid from traditional interactive database systems since Euclid's expert system-based structure means that it only searches the relevant portions of the knowledge base rather than undertaking a brute force search. In a sense, Euclid searches for information in a quasi-intelligent manner. Also, since Euclid evaluates development applications in a formalized way, it should, in theory, lead to more consistent and appropriate development control judgements; however, it has not been possible to evaluate this hypothesis.

In conclusion, Euclid has been developed with junior planning staff in mind and depends for its success upon the user/planners exercising their judgement and being intimately familiar with the Zoning By-laws and the development approval process. Many of the questions Euclid asks have very particular meanings and Euclid expects them to be interpreted correctly. Furthermore, Euclid assumes that the user understands the implications of the input they provide. Since Euclid cannot handle uncertainty and cannot explain what it is it's after, the onus is on the user to provide prudent responses and to evaluate Euclid's results in a critical manner (see Hammond's (1986) and Collins' (1987) comments on the role of the user in developing expert systems and the need to develop systems with the end-user in mind). Unfortunately, these two shortcomings mean that Euclid does appear, to the uninitiated, as a 'black box'. Uncertainty handling and explanation facilities have not been built into the current version of the system.
due to the author's lack of experience in using Prolog and it should be recognized that the first few systems that planners build cannot be expected to meet all of the criteria developed in Chapter Three.
CHAPTER FIVE: AN EVALUATION OF THE DEMONSTRATION SYSTEM - EUCLID

In Relation to the Evaluative Framework

While Euclid illustrates that expert systems can be developed in the Urban Planning sphere, and in particular for the task of development control, it should be clear that it can in no way be considered a comprehensive state of the art tool. Consequently, any evaluation of the system from the stand point of the evaluative framework presented in Chapter Three should not lead towards any categorical conclusions in favour of or against Expert Systems in the planning work place. Nevertheless, the lessons learned from applying the framework to Euclid can help in the formulation of methodologies and expert system structures which may be suitable to planning domains, as well as indicating directions for future research. It is within this context that the framework is used to evaluate Euclid.

A central underlying thrust of this thesis is that development control is not planning in its truest sense and that, in spending the lion's share of their limited resources to administer Zoning By-laws, Planning Departments are failing to deal adequately with the urban problems that cities face in the 1990s. The experience of Euclid shows that it may be possible to incorporate Expert Systems into the development control function and possibly reduce the resources expended in this regard while at the same time perhaps improving the consistency of the approval process. Obviously, systems such as Euclid would be most at home evaluating fairly ordinary small to medium scale development proposals. Mega-projects and other complex proposals such as Pacific Place or Coal Harbour would still require the traditional approach involving a team of planners. In this regard, an expert system which could screen out complex development proposals from the simple ones and evaluate only the latter would be highly valuable. Euclid
represents a very simple implementation of the evaluation component while there are already several expert systems operational which perform screening operations - SCREENER for example (developed by Environmental & Social Systems Analysts Ltd. - a local Vancouver consulting firm specializing in environmental and resource planning) screens out business and development projects according to their compliance with the federal guidelines relating to Environmental Impact Assessment.

By providing a consistent and user-friendly environment in which to assess development projects, Euclid also meets Keen & Morton's description of what decision support implies. By evaluating the project proposed Euclid assists managers to make decisions with regard to zoning approvals since it evaluates a project and makes a 'recommendation'. It is important to note that Euclid's recommendation is based both upon the user's judgement in the first place and on the assumption that the user will not accept Euclid's result as the absolute truth. However, the very fact that Euclid makes a recommendation in this regard could be considered to be dangerous and perhaps should be replaced by a check list informing the user of those regulations the project meets, those it fails, and the reasons why. Euclid could still make a 'recommendation' but with such a feature the User would be in a much better position to judge Euclid's performance. Nevertheless, Euclid has been conceived, in its present form, to support rather than replace the planner's decision making skills as well as to improve the efficiency and consistency (i.e. the effectiveness) of those decisions.

With regard to the evaluative framework itself, Euclid can only be considered a partial success since it has not been possible to program in all of the features which set expert systems apart from traditional procedural programs and thus Euclid is more of a 'black box' than is
desirable. This is more a function of the software tool used and the limited programming knowledge of the author rather than any difficulties inherent in creating expert systems to meet a planning task. Taking each of the ten criteria in turn:

1). THE OUTPUT OF THE TOOL SHOULD BE A) CONSISTENT, B) SUITABLE, AND C) RELEVANT TO A PARTICULAR POLICY CONTENT OR PLANNING PROBLEM.

The structure of Euclid does mean that the output is consistent since Euclid's responses are built up from text strings. If, in testing a particular development proposal, the user provides the same set of answers, Euclid will respond in the same way again and again until a different set of answers is provided. Euclid's evaluation is also consistent since it only recognizes absolute truth and absolute falsity and thus a project either fails or succeeds. In reality, however, the project may meet some requirements and fail others and so may comply with all or only a portion of the zoning schedule. Thus, development projects do not conform to the zoning regulations in purely black and white terms.

But Euclid errs on the side of caution. As soon as the project fails to meet one zoning regulation, Euclid assumes the entire project fails, since a part of the regulations are contravened. Euclid then informs the user which regulation has been breached so that that specific issue can be addressed. Of course, Euclid's consistency depends, to a large extent, upon the input it receives (see criterion 3) but it will always be consistent in terms of the way it evaluates a project.

As far as the development control domain is concerned, Euclid's results are both suitable and relevant to the task at hand. In providing a quick and consistent means of checking a development proposal, Euclid's response system allows planners to identify quickly those projects which meet zoning regulations and those which do not. It thus performs a crude
screening function. At present, Euclid suspends its evaluation as soon as the project under evaluation fails a regulation. A more useful variation in this regard would be for Euclid to evaluate the project against the entire By-law and then produce a report showing where the project complies with and contravenes the regulations.

2). THE ACCURACY OF RESULTS AS WELL AS THE WAY RESULTS ARE ACHIEVED SHOULD BE RECOGNIZABLE. THE TOOL MUST BE ABLE TO SUBSTANTIATE AND EXPLAIN THE RESULTS ACHIEVED AND USERS MUST HAVE THE CONFIDENCE IN THE TOOL'S PROCESS AND OUTPUT.

The current version of Euclid fails to meet this criteria for several reasons, the most significant of which is its inability to explain how it arrived at a particular result, or why it is asking a particular question. Since it cannot substantiate its line of reasoning nor provide a rationale for its results, users are left to judge these results for themselves. A further problem lies in the fact that Euclid cannot handle uncertainty, only recognizing TRUE or FALSE results. The simple reporting system employed also implies absolute accuracy in Euclid's results and, to some extent, forces the user to accept these at face value since the magnitude of any failure to meet the regulations is not displayed. For example, a building may exceed the height allowance by one foot or by 20 feet yet in Euclid both instances lead to the same: "The use proposed fails to meet height requirements..." response. In this sense, Euclid is itself a black box.

Furthermore, Euclid's current, simplistic operating structure cannot inspire great confidence on the part of the user in terms of relying upon its results. However, since Euclid is so closely based upon the structure of the Zoning By-laws it mimics, any lack of faith concerning the way it works implies a lack of faith in the way the Zoning By-laws themselves are structured.
and interpreted. Greater confidence in Euclid would be to provide an expanded reporting structure which lists not only how a particular project stacks up against the regulations but also indicates the magnitude of the difference between the building's dimensions and the zoning specifications which apply to that site.

3). INPUT MUST BE A) ACCURATE, B) VALID, AND C) OF GOOD QUALITY.

As mentioned under criterion 1, Euclid relies heavily on the user to provide accurate and valid input. In one or two instances, Euclid will flag suspect information and query the user to make sure the input provided is correct. For example the area of the ground floor cannot be greater than the floor area of the whole building and if such a situation occurs, Euclid asks the user to confirm the figures or provide new ones. Finally, provided the user operates the system in a responsible manner, the input supplied will usually meet the three conditions above since it will be based upon information submitted as part of the development application.

4). THE TOOL SHOULD BE THEORETICALLY AND METHODOLOGICALLY VALID.

Since Euclid is based upon contemporary Canadian development control practice, it can only be based upon a solid epistemological foundation in so far as the concept of Zoning itself, as a method of controlling development, is theoretically valid. While the practice of zoning has its critics (see Goldberg & Horwood, 1980; Ronis & Rucker, 1971; and Ukeles, 1964, for example), it has nevertheless been firmly established in Canadian municipalities since the early days of planning and so has gained widespread support as a valid planning tool. Therefore, as an evolutionary development of zoning practice, Euclid is itself supported by this theoretical and methodological acceptance of zoning.
5). THE TOOL SHOULD BE HIGHLY INTERACTIVE AND ACT AS A CONDUIT FOR COMMUNICATION BETWEEN 'EXPERT' AND USERS, AS WELL AS BETWEEN USERS.

The query-the-user process which is part and parcel of Euclid makes the system highly interactive - a result is not obtained simply at the stroke of a single key - rather, the user is led, by the series of questions they see, through Euclid's evaluation process. As a conduit for communication however, and hence as an educational tool for learning about zoning, Euclid is less successful since the flow of information tends to be primarily from the system to the user. It is the lack of an explanation facility which is the main reason why Euclid is not very successful in this regard, but then Euclid was explicitly targeted at planners who are familiar with the Zoning By-laws and with the development control review process.

6). THE TOOL SHOULD BE EASY TO LEARN, USE, UNDERSTAND, AND INTERPRET, FOR A RANGE OF USERS.

The use of pre-defined text strings for asking questions and displaying results means that Euclid does not require a natural language interface. While this limits flexibility and tends to lead to the 'hardwiring' of information into the system, it means that Euclid operates in a way which the user can understand quickly. No programming knowledge is required to run the system and since Euclid directs the questioning, gives an indication of the answers it expects, and to a limited extent catches user input errors, the system has proven very easy to operate and learn. In addition, the various responses that Euclid provides in terms of results are also straightforward and thus the result of an evaluation is quite clear.
7). THE TOOL SHOULD BE ROOTED IN BOTH THE CONTEXT OF THE PROBLEM AND IN OUR CULTURE AND COGNITIVE HERITAGE.

Since Euclid is based upon existing Zoning By-laws, it is firmly rooted in the Canadian planning tradition. In addition, as a response to the contemporary pressures facing development control planners in the 1990s, Euclid represents an attempt to deal with the problem of constrained resources and, at the same time, attempts to make the development control process more effective and efficient. Euclid also acknowledges, indirectly, the need for forward, or long-range, planning in light of changing expectations and planning pressures.

8). THE TOOL SHOULD NOT BE DATA INTENSIVE AND MUST BE EASILY UPDATED AND MODIFIED.

The current version of Euclid does not require excessive data in order for it to work but like most AI-based tools is, unfortunately, fairly heavy in terms of its memory requirements. If more Zoning schedules were to be added, then Euclid would become extremely data intensive unless a completely different inferencing strategy and representational structure were to be employed. Euclid's appetite for memory as well as the difficulties which would be involved in increasing the scope and sophistication of the system suggest that its current structure is incorrect. Consequently, if Euclid were to be developed further it would be highly desirable to involve an experienced Prolog programmer in the development team and possibly use a more sophisticated Prolog language.

In terms of updating the current version of Euclid, changes to the zoning specifications would be very easy to make since the database files could be accessed by the standard DOS editor. If the zoning by-laws were to change more dramatically in terms of altered rules and or new zoning requirements, the changes would be less easy to make. During development, it was
found that adding a new rule or altering an existing one could occasionally lead to the system behaving unexpectedly or producing errors. Again this is an indication that Euclid’s representational and inferencing structures should be altered and improved. By and large however, additions or changes to the rules proved to be easier to make in Turbo Prolog than in procedural languages like Pascal.

9). THE INFERENCING SYSTEM SHOULD BE INDEPENDENT FROM THE REPRESENTATIONAL SCHEMA USED TO EXPRESS DATA.

The use of Turbo Prolog’s built in pattern matching capabilities for the inferencing system in Euclid means that the way the system evaluates the knowledge base is kept separate from the information contained within the knowledge base. While some of the rules in Euclid define the order in which Turbo Prolog should search the knowledge base, the results of the comparisons between stored zoning information and user supplied zoning information are determined purely by Turbo Prolog’s Unification procedures. Consequently more rules, covering new zones, could be added to Euclid and, provided they were consistent with the format of similar rules already in the knowledge base, the system would be able to evaluate them.

However, the simplistic nature of Euclid’s current inferencing system (it cannot handle uncertainty for example) and reporting system has, to some extent, necessitated that information be ‘hardwired’ into the knowledge base. For example, the relationships between the characteristics of the development proposal and the requirements of the Zoning By-law [e.g. (height of the project proposed) related to (height allowed under the zoning by-law); (number of dwelling units proposed) related to (number of dwelling units allowed)] are defined explicitly for each condition of each rule within the knowledge base. Thus, major changes to the zoning by-laws could require that the relationships for those conditions affected be re-written. Such
changes could have repercussions in the logic of the rules and in the way the system operates as a whole.

In a production rule system like Euclid, tracing the logic of the rules and the way they are connected to each other becomes harder as the number of rules expands and, thus, changes to the by-laws could lead to the system being 'down' for a period while the revisions are incorporated. Note, however, that provided the relationship for a condition remains the same, changes to zoning specifications (i.e. changing the height limit from 35 feet to 40 feet) would require that only the database containing that specification be accessed and the height figures changed. In such situations the knowledge base itself would require no modifications.

10). THE TOOL SHOULD BE HIGHLY FLEXIBLE AND ALLOW THE USER DISCRETIONARY FREEDOM AND FREEDOM OF PURPOSE.

Euclid does not meet this criterion since in its current form it can only check to see whether or not a proposed development meets the regulations of the zoning schedule relevant to the site in question. In order for Euclid’s results to have any worth, its evaluation process depends upon the development project being fairly well defined and the application being at a fairly advanced stage since details on the design of the project form a major part of the system’s information requirements.

Discretionary judgement and expertise could not be built into Euclid, as they have in some expert systems, for two reasons. First, incorporating such a capability into Euclid would have entailed significant increases in the scale of the programming effort required. And secondly, access to the relevant expertise was not possible; as noted in Chapter One. Conse-
quently, it was not possible to undertake a knowledge elicitation exercise and thus Euclid has had to rely solely on the material contained within the written zoning schedules for its rules.

In terms of giving the user more freedom, Euclid hints, in a crude way, towards using expert systems as a vehicle for evaluating the effectiveness of new, or proposed, zoning regulations and for asking "What if?" questions. Using expert systems in this way could possibly reveal any unintended consequences and provide planners with the means of developing and testing, in a structured manner, the regulations proposed. At the present time planners do not have this capability and so there may be a potential role here for expert systems in planning. The present structure of Euclid's query-the-user process, though, does not give the planner this kind of flexibility in terms of being able to ask 'What if?' questions or in evaluating the impact upon development applications of proposed regulation changes.

While the planner can use Euclid to test variations of the project, to see how these affect the status of the development application, by running several evaluations - each time altering one or more of the answers during the question and answer session - the system cannot store an evaluation session in a file and reactivate it at a later date and this serves to limit Euclid's flexibility. Such a feature would be a desirable component in a fully functional system since it could be used to produce a report which could then form part of the official record and be included in the recommendations presented to Council. City Managers would also be able check the report and or run the system to gain a greater understanding of the project proposed. In addition, the ability to edit the responses of a session would allow project variations to be tested easily and the 'best' alternative identified.
Another beneficial capability would be to allow the evaluation session to be suspended part way through, say if information wasn’t available, and then continued at a later date. This would allow the planner the freedom to see where a project failed the zoning by-laws and what would have to be done in order to rectify the situation. With such a flexible evaluation process, Euclid could expand its current evaluation/prediction capabilities to provide a tutorial environment in which junior staff could gain experience in development control.

Conclusions of Applying the Evaluative Framework

From the analysis above, Euclid is clearly not an "Ideal Problem Solving and Decision Support Tool" since there are many areas where it fails to meet the kinds of flexibility and freedom required of a Planning Support Tool. The analysis does, however, illustrate areas where Euclid could be made a more useful tool. Yet, it should be noted that in order to incorporate the improvements mentioned above, Euclid’s current knowledge base would have to be comprehensively altered and improved and Euclid’s search strategy refined.

Nevertheless, Euclid does demonstrate that there may be some potential for expert systems in the sphere of urban planning, and, in that sense, it is a success. The ease and speed with which users can interact with Euclid suggests that fully fledged expert systems may lead to time savings in the assessment of development applications. The fact that the zoning schedules are contained within a knowledge base, and that Euclid treats the zoning rules within the knowledge base in a consistent manner, may also mean that fewer human errors go undetected. At the very least the work done here suggests that tools developed from Euclid could perhaps be used to confirm and check human decisions. Finally, it appears that a more advanced version
of Euclid could perhaps be used in an interactive way to train new staff members in the nuances of development control.

It should be appreciated, though, that fully fledged implementable systems cannot be created immediately, especially since planners themselves have limited programming expertise. Initial development systems such as Euclid are a good first step towards identifying the parameters necessary for creating operational systems since they help define the structure and methodology of the system, as well as the capabilities which should be incorporated into a professional tool. Euclid represents approximately two person-months of programming effort and, as a result of this expenditure, has produced a much clearer picture of the type of development control expert system which would be of use to planners.

**Turbo Prolog As A Development Tool**

Another area in which the development of Euclid has yielded results is in the types of development tools used. If another attempt was made to develop Euclid, a comprehensive expert system shell, such as Level 5, would probably be used since beneficial use could be made of the built-in explanation facilities.

Turbo Prolog has proven to be quite an easy tool to work with and allows a large degree of flexibility in tailoring the knowledge base to the specific needs of each regulation. It is, however, labour intensive and a lot of programming effort would have to be carried out if the kinds of facilities which come, as standard, with some expert system shells were to be included in the system (e.g. an explanation facility).
In building Euclid, problems were encountered with memory limitations as well as with Turbo Prolog's limited mathematical capabilities - especially when comparing two real numbers. Furthermore, the code produced for Euclid is fairly lengthy - all the files together total in the region of 60 Kb - and it was discovered that as the system expanded it became harder to follow the internal workings of the system when trying to debug errors. Refining the system and checking for, as well as editing, errors thus proved to be exceptionally time consuming tasks. The lesson here appears to be that if a language such as LISP or Prolog is to be used, then it is wise to involve a programmer, experienced in that language, in the development process, so that a comprehensive program can be developed quickly and efficiently. For planners who want to develop systems on their own, or to test the concept, the experience of Euclid suggests that a shell is probably the best environment in which to develop potential applications. Nevertheless, the fact that Euclid has been developed successfully using Turbo Prolog, and with only a basic understanding of the language, is in itself highly encouraging.

Lessons Learned from the Zoning By-laws

The experience of Euclid also exposes several lessons of which both planners and municipal lawyers should be made aware. The translation of the six zoning by-laws used in Euclid into a format which is readable by Turbo Prolog forced a logical rigour upon both the structure and content of these by-laws. Under the glare of such scrutiny the present methodology employed in writing zoning by-laws revealed several weaknesses.

First of all, the entire Zoning By-law assumes that all its readers are fairly adept at interpreting the requirements contained therein and understand the concepts upon which the By-law is based. While planning staff may be familiar with the contents and implications of the
By-laws, the same may not be said for developers or members of the public at large and thus the development control review process may be lengthened unnecessarily while parts of the By-law are clarified, explained and the application revised.

In addition, and beyond the problem of interpretation, it was discovered, in trying to produce logical equivalents for the zoning regulations and requirements, that legal constraints placed upon the way in which regulations could be expressed (such as not allowing mathematical formulas to be incorporated into a regulation, as well as the lack of diagrams) obscure the clarity and simplicity of the regulations. This made translation of the By-laws harder than necessary and resulted in making it very easy to create logical inconsistencies and errors in the content of the knowledge base. Indeed, in some cases (the regulations governing sunlight exposure in the RT-2 zone for example) the intent and meaning of a regulation were so obscured that it was impossible to produce a suitable logical rule. In such situations, the solution chosen was to make Euclid ask a simple question to the effect: "Does the proposal meet the requirements for sunlight exposure, as defined in section 4.10.1 of the RT-2 schedule? (y/n)".

A more frequent, but less serious, problem emerged with regulations which were couched in 'whichever is the greater' or 'whichever is the lesser' terms and these proved somewhat confusing and hard to interpret logically. Another problem which emerged in handling the six By-laws was unintentional differences in wording. In many cases, several By-laws shared the same regulation and yet that regulation in each case was worded differently although the intent clearly remained the same. This lack of consistency between zoning schedules, in terms of the way regulations are expressed, added yet more complications during translation.
Conclusions

Even with a domain (development control) which one would consider to be highly structured due to the legal nature of the By-laws, the experience of Euclid has shown that there exists quite a large amount of flexibility, discretionary freedom, and variation in the content of the By-laws and in the ways the By-laws themselves can be interpreted. Furthermore, much is left to the discretion of the Director of the Planning Department, and while a system such as Euclid may reach a negative judgement it may still be possible for the project to be approved. This freedom and variation added to the difficulties involved in developing Euclid.

On top of these issues there are also serious legal ramifications involved in creating an expert system based upon a Zoning By-law since the Municipality, as well as the system developers, could be held potentially liable for any errors contained within the system, or any wrongful conclusions reached. Euclid's production rule structure exacerbates this problem by making debugging and testing of the system a time consuming and difficult task.

For these reasons, the suitability of zoning by-laws as a basis for building development control expert systems may be suspect and perhaps future system development should focus more on the development control process itself rather than on the substance of the projects proposed. Indeed, this finding seems to be supported by Rodriguez-Bachiller's (1990) recent research at Oxford Polytechnic where such a process-based system has been under development. Nevertheless, the overall conclusions from the development of Euclid have to be positive ones since the system was developed in a short period of time and without any extensive knowledge or prior programming experience with the Turbo Prolog language.
As should be expected with any exploratory system, Euclid has several severe limitations. Despite these, though, the process of developing Euclid has been valuable in that it has identified many improvements and refinements, and has clarified the requirements of Expert Systems if they are to be used in a planning context. Assuming that such improvements and requirements can be incorporated successfully into Expert Systems, that the technology and software improves over the next several years, and that Planners and municipalities are perfectly clear about the role Expert Systems should play within planning, then there does appear to be some potential for Expert Systems as planning tools.
CHAPTER SIX: A GENERAL CRITIQUE OF EXPERT SYSTEMS

While the majority of the literature on expert systems is wholly uncritical or empirical in nature, there exists a small body of literature which has scrutinized extensively both the theoretical as well as the practical basis of expert systems. In order for planners to make informed decisions and to evaluate expert systems knowledgeably, it is essential that these critiques be discussed.

We have already seen the problems associated with even a very simple expert system tool in terms of knowledge elicitation and representation, as well as the practical difficulties which arise in using inference techniques and logical languages. It is of paramount importance to realize that expert systems are in their infancy and will likely remain in such an embryonic state for several years to come despite the claims of the Japanese with their Fifth Generation Computer Project as well as the Americans:

... the science of Artificial Intelligence (AI) - is seen as a key source of innovation and impetus for facilitating the revolution in information technology. More specifically, there exists the dream of so-called "5th Generation" computing .... wherein the hope is that developments in AI will enable the construction of computing machines which, for the first time, will have serious intellectual abilities (i.e. real artificial intelligence). These computers will have access to powerful knowledge bases encapsulating many areas of expertise from which they will make complex inferences (both inductive and deductive) in order to aid decision-makers. (Bloomfield, 1987) (author’s emphasis)

As mentioned earlier, such euphoric statements must be treated with a generous amount of skepticism since the concept of expert systems and AI itself can be attacked from a variety of perspectives. It will suffice here to divide the critique into three segments, namely: Theoretical Limitations, Practical Limitations, Professional Planning Limitations.

112
Theoretical Limitations

A central problem surrounding AI and expert systems is the very nature of knowledge and expertise themselves. What constitutes knowledge? What is expertise? Such questions are almost impossible to answer since both are flexible and dynamic and can change over time. Defining in advance the knowledge or expertise that will be required for a given situation is exceptionally difficult since how does one determine what is important and what isn’t?, especially if that situation has not yet occurred. Collins (1987) as well as Gammack & Young (1985) have noted that there are varying kinds of knowledge ranging from facts, rules, and heuristics, through to cultural, manual, and perceptive skills and other ‘common sense’ elements. Some knowledge is consciously known or ‘shallow knowledge’ but there also exists subconscious knowledge - sometimes referred to as deep or tacit knowledge - which is not directly knowable by the expert and thus remains inaccessible to both the knowledge engineer and to the expert system.

I do not intend to enter into a lengthy treatise concerning the validity of Artificial Intelligence as a field of study since much of the academic discussion is centred around the complex problem of trying to model human brain functions and is of little interest here. Needless to say, the whole question concerning AI and whether computers can actually reason in ways similar to human beings, or demonstrate any real intelligence at all, is currently a hot topic for debate and this should become apparent from the focussed critique presented here. For a critique of AI as a field of study and of computers as ‘minds’ in and of themselves see Searle (1987), Shanker (1987), and Born (1987).
In terms of expert systems applications in planning, the majority of real planning issues are exceptionally complex due to their open-ended nature and the involvement of numerous vested interest groups. Many of these issues are inherently qualitative in nature. Yet as Merry (1985) has noted: "Qualitative reasoning is concerned with .... all kinds of representation and manipulation of 'deep' knowledge and also 'common sense' reasoning". While expert systems researchers have made some inroads in this regard ".... there is still no principled way .... [for expert systems to handle] .... qualitative reasoning and many extremely hard problems remain to be solved". So, in precisely the area where expert systems are supposed to represent a significant advance over traditional approaches there has been little successful progress.

Furthermore, as Shanker (1987) and Pfaffenberger (1988) have noted, expert systems are heavily imbued with powerful misconceptions, biases, and value statements which are difficult to grasp at first since they are an inherent part of the system’s structure. Although, as Born (1987) notes, this is a common failing of the majority of the computer tools available today: "In many cases we do not realize that misconceived conceptions such as outdated ideas about economic growth, built into the computer programs via the theories underlying them, also seep into our world through these channels." (Born, 1987)

**Practical Limitations**

Of more immediate relevance in applying expert systems technology to the field of planning are the practical problems involved in trying to develop workable and useful systems. The major stumbling block which all expert systems face concerns the elicitation and representation of knowledge. As we have seen, knowledge, especially planning knowledge, is extremely complex, dynamic, and is nearly always dependent upon the current situation or
context. Thus, knowledge which is useful in one situation may not be so in another. Furthermore, knowledge can either be tacit (i.e. subconscious) or explicit (e.g. rules, facts, 'shallow' information). It is, therefore, impossible to try and gather the knowledge required to meet every eventuality and even harder to foresee what knowledge will be required for future situations. This is why expert systems have only been successful in domains where only limited amounts of knowledge are required, where the situational permutations are relatively small, and where such information is highly structured and narrowly focussed.

Assuming that the required knowledge exists, it must then be elicited in order to be translated into an expert system. The success of an expert system depends upon how successfully knowledge can be extracted from the relevant expert(s). Some expert systems, including the simple demonstration system presented here, have been based largely upon the knowledge contained in books and manuals. Indeed, Leary (1988a) has argued that Design Guideline Manuals are in effect knowledge bases since they represent an abridged and codified culmination of 'real' planning and development control knowledge. However such 'book knowledge' represents the shallowest of shallow knowledge since it projects an idealized image of the task at hand and fails to take into account the messiness of reality (Collins et al, 1985). The meat of qualitative analysis, then, lies in deep, rather than shallow knowledge; in process rather than substance.

Thus, the heart of an expert system lies in the heuristics and tacit knowledge which experts possess. But, and here's the rub, if the vast majority of this information is tacit or 'forgotten' knowledge (i.e. it’s subconscious) how does one extract it from the expert? The process of knowledge engineering has been developed to try to deal with this problem and
involves workshop sessions between the knowledge engineer(s) and expert(s), at which the knowledge engineers use a variety of techniques in an attempt to draw knowledge from the experts and thus develop a model of the decision making or problem solving process(es) which the expert(s) use. For a discussion of knowledge elicitation techniques see Gammack & Young (1985) and for a description of the knowledge elicitation process at work see Berkowitz & Strzepek (1987). The success of this process not only hinges on the willingness of the experts to take part but more importantly on the ability of the knowledge engineer(s) to both extract and structure valuable knowledge as well as recognize when important heuristics have been uncovered. As one could expect, such knowledge elicitation sessions can produce vast amounts of unorganized information and separating the 'wheat from the chaff' can be a time consuming undertaking. Indeed, even in a relatively simple domain, Berkowitz & Strzepek (1987) found that for every hour spent eliciting knowledge four hours were spent digesting the information gathered.

Once knowledge has been gathered is has to be translated into a representational structure. As mentioned in Chapter Two, the most common representational structure is the production rule system. But as Sloman (1985) has noted, knowledge is so complex and dynamic that it requires a variety of formalisms in order to capture the full nuance of information. As humans, we are uniquely adept at understanding and coming to terms with a variety of representational systems: e.g. maps, models, diagrams, flowcharts, algorithms, etc., and we need to use such systems in order to analyze various problems and tasks successfully. As tools around which thinking and analysis can take place, such representational models are very useful. One possible role of expert systems could be as an additional representational structure in this regard.
While blackboard systems may allow the use of a variety of representational formalisms and so expand the heuristic models available, the overwhelming predominance of the production rule approach in the majority of current expert systems represents a severe weakness. No one representational structure can hope to capture the full force of knowledge nor deal with the varied types of information available. Thus, production rule systems tend to be biased towards 'shallow' knowledge since their structure forces information to be represented in fact and rule terms and as we have seen not all knowledge can be so represented.

Perhaps a more important problem with the production rule representational schema is that it implies a model of human 'thought analysis' where human experts solve problems through a rational and logical assessment of rules and facts. In other words, the process of becoming an expert involves learning what the rules are and how to apply them. This view of 'experts as technicians' raises some serious questions but appears to be inherently flawed. Certainly, it does not correspond to the process by which planners gain experience and expertise. Dreyfus & Dreyfus (1984, 1986) have put forward a convincing model of the process by which expertise is gained. This model is opposite to that assumed by the production rule representation. In short it argues that beginners apply rules and facts only in trying to learn and understand a domain. As expertise is gained, we move away from applying rules towards recognizing exceptions to the rules and finally we learn to analyze tasks on a holistic rather than logical level [holistic here means ".... superimposing the records of whole situations and measuring their similarity." Dreyfus & Dreyfus (1984)]. As an example of the mind functioning holistically rather than logically, Dreyfus & Dreyfus (1984) ask the question: How do we, as humans, recognize a face to be familiar? We do not, they argue, do so by decomposing it into its component parts (eyes, nose, hair, mouth, etc.) but react to it instinctively, intuitively, and as a whole.
At present, there exists no computational representation schema for mimicking such a holistic problem solving technique, hence the problems involved in trying to elicit facts, rules, and heuristics from experts who are thinking holistically:

It seems that a beginner makes inferences using rules and facts just like a heuristically programmed computer, but that with talent and a great deal of involved experience, the beginner develops into an expert who intuitively sees what to do without applying rules. .... The expert is simply not following any rules! He is .... recognizing thousands of special cases. .... If one asks the experts for rules, one will, in effect, force the expert to regress to the level of a beginner and state the rules he still remembers, but no longer uses. (Dreyfus & Dreyfus, 1984, p. 226)

Another problem which is prevalent for systems which attempt to operate within open ended domains (i.e. planning) is the problem of combinatorial explosion and diminishing returns. Combinatorial explosion stems from the way a group of objects can conceivably be arranged into varying combinations. As Brittain (1986) has noted, 10 objects (e.g. rules or facts) can be combined 1,024 ways while there are only 32 ways 5 objects can be combined. On the other hand, 15 objects have 32,768 combinations! Thus we have a geometric progression of the number of ways an expert system could consider a set of facts and rules. As Pfaffenberger (1988, p.68) has noted: "For every arithmetic increment in the system’s breadth and expertise, there occurs an exponential increment in the number of situational contingencies that must be foreseen for the program to operate effectively." In other words in order to get a 10% improvement in performance the number of contingencies (i.e. rules and facts in the knowledge base) must be increased by at least 100% (i.e. a doubling of the knowledge base, see Barr & Feigenbaum, 1981) - this is the principle of diminishing returns since squeezing each extra ounce of performance out of the system requires more and more new knowledge.
This problem is a function of expert systems being situation and context dependent. Recall that the knowledge base is a repository of information about the problem and its context. Ideally, an expert system should be able to deal with and anticipate every conceivable situation and its interrelationships. This is, of course, impossible, since in an open ended domain like planning the number of situations and interrelationships would be infinite. By necessity, the knowledge base has to be finite in order for the interactive user-expert system process to take place. But even for a finite knowledge base, the amount of information required to fully describe even fairly simple situations and limited interrelationships can be quite significant. In a production rule expert system such situations and interrelationships are described using facts and rules, and obviously each situation requires a variety of rules in order to express it. Expanding the scope of an expert system to take account of just one more situation thus requires that a whole host of new facts and rules be added to the knowledge base which describe not only that situation but also how it relates to all the other existing situations. Thus, for each increase in the expressive power of the knowledge base, the number of rules required explodes combinatorially.

While the heuristic approach of expert systems and their use of a variety of control strategies should, theoretically, limit the size of the search space (i.e. that part of the knowledge base which is searched for information) relating to a given problem, expanding the size and performance of the expert system and developing the flexible control strategies necessary to handle all the information remains a fundamental problem in expert systems development. In production rule systems specifically, the order in which information in the knowledge base is assessed determines the effectiveness and efficiency of the system - the rules initially triggered
should reduce the search space dramatically so as to prevent lengthy search processes and user frustration over the time it takes for the system to respond.

Another significant area where expert systems face severe difficulties is in their handling of uncertainty. As mentioned earlier, the majority of the techniques which have been developed in this regard are ad hoc at best (see Mamdani et al, 1985) and lack either a solid theoretical basis or substantial supporting empirical evidence, or both. Indeed, the hopes that expert systems can truly embrace uncertainty within their modus operandi remain bleak:

Any attempt to deal with uncertainty within a strictly classical logic framework is itself an ad hoc method and does not constitute a sound theoretical framework. (Mamdani et al, 1985)

Like knowledge, there are varying kinds of uncertainty since information can be uncertain in a variety of ways. Moreover, it is almost impossible to provide a comprehensive list of these dimensions of uncertainty since many exist across a seamless spectrum of quantitative and/or qualitative 'values' which themselves can be fuzzy and imprecise; consider the terms 'certain', 'probable', 'possible', 'uncertain', 'improbable', 'impossible', for example. While these give an idea of the extent to which uncertainty exists, its complex nature is best illustrated by listing the categories from which 'uncertainty' springs. The seven categories below reveal the nuances involved in uncertainty but are by no means an exhaustive list nor are the categories mutually exclusive. They do, however, serve to indicate the degree to which uncertainty is a central element of human life:

Knowledge and information has associated with it one or more of these elements in varying proportions, and uncertainty can consist of:
BELIEF One’s belief in a given proposition or a piece of knowledge.

LIKELIHOOD The likelihood of a single, compound or conditional event.

EXTENT The extent of a proposition concerning a continuous variable.

IMPRECISION The imprecision about any information.

EXCEPTION Any exception to a general rule.

MANDATE The mandate for performing some action.

RELEVANCE The relevance of one piece of information to another.

(see Mamdani et al, 1985 - for a more comprehensive discussion of uncertainty)

The over-riding lesson here is that, as with representational structures, no one way of handling uncertainty is sufficient, especially in a field like planning where uncertainty and varying values conflict with each other and imbue the planning process so completely.

Professional Planning Limitations

There exists almost no planning literature which evaluates the role of expert systems in planning from a critical perspective. The vast majority of planners currently researching expert systems are undertaking preliminary research into the potential application domains of expert systems and are doing so from within the firmly established rational planning philosophy. Consequently, the critique presented here is cursory in nature and merely represents an intuitive reaction to the possible adverse effects of expert systems upon various elements of contemporary planning practice. There is, thus, a need for expert systems to be evaluated from other planning perspectives but particularly in terms of how the infusion of expert systems might affect community development and the work of radical planners.
From a planning point of view there exists the danger that the introduction of expert systems may entrench the notion that top down rational planning is the preferred planning methodology. The whole process of developing expert systems implies the existence of a computer-literate elite which, in developing these tools, imbue them, whether consciously or not, with their beliefs, values, and biases. Furthermore, the very terminology of both AI and expert systems is imbued with such elitist tendencies; the phrase expert system implying a top-down process, for example.

Furthermore, the possible creation of the notion that planning can be undertaken successfully through planners becoming users who interact with expert systems poses a significant danger and could lead to a dehumanization of the planning process:

It is hard to believe that the company or government that has such systems dealing competently, but not expertly, with suitably selected problems will dominate all competition, as claimed by enthusiasts for Japan’s fifth-generation project and for an expensive American response in kind. The most that can be expected is that expert systems may someday take their place alongside planning charts, management information systems, and professional training programs as useful tools for improving overall performance. ... A genuine danger, however, faces the company or government going that route. To the extent that junior employees using expert systems come to see expertise as a function of large knowledge bases and masses of inferential rules, they will fail to progress beyond the competent level of their machines. With the leap beyond competence to proficiency and expertise thus inhibited, investors in expert systems may ultimately discover that their wells of true human knowledge have gone dry. (Dreyfus and Dreyfus, 1986, p. 121)

(Recall that one of the chief purposes of expert systems is to capture the knowledge of experts so as to have this information on tap once the expert has died or retired and so that it may be used to train junior staff)

The possible dehumanizing effect of expert systems in planning has also been raised in connection with group process. Alan Hickling (personal correspondence, 1990), for example,
has argued that expert systems would have a damaging effect upon the success of group process since expert systems would foster user/expert system interaction in favour of group and community interaction. The planning process also involves numerous interest groups and parties with varying and conflicting values and objectives. The existence of these groups and their conflicting goals and beliefs complicates the processes of knowledge engineering and uncertainty handling still further, and thus could reduce the applicability and acceptance of expert systems technology in planning.

On the other hand, Collins et al (1985) have argued that expert systems could serve to disseminate knowledge from experts to others. Planners could take advantage of this feature of expert systems in order to disseminate knowledge from ’expert’ planners to members of various community groups and the public at large and thus inform people on the planning process. But Collins et al also argue, that in order for this "knowledge transfer" to take place, both the expert and the user must share the same tacit knowledge and a common culture. The experience in planning suggests that they must also share common beliefs, values, and desire similar goals in order for the content of an expert system’s reasoning process and output to be understood and accepted:

It becomes clear, that systems largely filled with information should be easy to build provided they are used by end users who know how to interpret the information. It is also clear, that the more expert the end user, the easier it will be to build a system that will be useful. Thus, the immediate value of expert systems, and of AI in general, is not to replace skilled persons - there is no immediate prospect of this - but to act as aids to increase the productivity of skilled persons and perhaps replace the relatively unskilled. (Collins et al, 1985)

Expert systems could, in fact, be used to support group process since they could make portions of the planner’s implicit knowledge explicit and thus open to scrutiny and debate.
Collins (1987) has argued that expert systems can be used to help render culture visible. Applying this notion to planning should mean that expert systems could be used to render both the planning process and the policies, values, and biases therein, visible. In other words, the processes of creating and using an expert system could be used to gain further insights into the planning process itself and to mediate between the various interest groups by developing workshop sessions in which groups interact with expert systems and then use that experience in group process workshops. Such programs would, of course, require planners highly sensitive to the limitations and co-optive dangers inherent in using expert systems.

The point of using an expert system in qualitative research settings [e.g. Planning] .... is not to create a computer program that is an end in itself; on the contrary, it is to harness microcomputer hardware and software so that it serves as a means to enhanced awareness, theoretical sensitivity, and humanistic understanding. [e.g. of the planning process] (Pfaffenberger, 1988, p.77)

In this way, expert systems, if carefully and sensitively used, could inform and reinforce group process. Thus, the expert system development process itself may be far more important in a planning sphere than the output it produces. This view is contrary to the belief of most expert systems developers who argue (as Burlinson, 1989, has argued) that everyone involved must abide by, accept, and agree with the output of any expert system. Furthermore, we have already seen that problems arise with the fixed set of rules contained within a knowledge base and, as Bloomfield (1987) has noted, such a fixed set of rules implies that consensus over their content is achieved (i.e. that everyone accepts the 'model' of planning which the knowledge base represents). Such a view follows traditional computer philosophy that system output is inherently correct. These system developers though are not working within the planning field and such notions are, I would argue, highly counterproductive in a planning environment since they
reinforce the traditional top-down rational planning model and create a paternalistic view of
group process and community participation.

Conclusions

It is clear from the above discussion, which is by no means an exhaustive one, that there
exist serious and fundamental problems not only with expert systems but their use in the field
of planning as well. However, any tool has its strengths and weaknesses. The current range of
demographic, statistical, and economic/environmental impact models, for example, which are
used quite extensively in the planning profession, are also infused with serious deficiencies and
yet still make valuable contributions to the planner’s efforts.

Despite the full force and weight of the criticisms against expert systems and the difficult
problems inherent in the expert systems development process, the majority of critics still feel
that the field has value for qualitative researchers and hence for planners. Even Dreyfus (1987),
perhaps one of the more vehement opponents of AI, while arguing that current AI and Expert
Systems techniques are in effect moribund, still holds out hope in calling for alternative
methodologies (such as neural nets) to replace the dominant production-rule paradigm. Dreyfus,
thus, implicitly accepts the concept of expert systems as a valid one.

The advantage of expert systems lies in their ability to make that which is implicit explicit
and thus they have the potential to ‘open up’ the planning process (although at present this
potential is quite limited due to the nature of the technology and the embryonic stage of expert
systems research in planning). The process of generating a knowledge base and eliciting
planning knowledge could lead to greater insights into the planning process itself and help in
developing consistent policy stances and regulation interpretations (see Hammond, 1986 for an example). In this role, expert systems can act as 'aide-memoires' and take advantage of the computer's capability to 'remember' and evaluate vast amounts of information. Expert systems can thus support the professional planner in their work and ease workloads by being able to handle the numerous routine (e.g. zoning) enquiries which planning staff receive daily (see Berkowitz & Strzepek, 1987).

However, the responsibility for implementing and using expert systems successfully and conscientiously rests ultimately with the planner involved. It is essential that planners using expert systems understand their limitations and inherent pre-understandings so that the systems and their output are not abused or used abusively. As both Born and Pfaffenberger have noted:

' Clever' users of the programs will (after some mistakes and negative experiences) learn to take the results the programs generate with a grain of salt. They learn to relate results to reality with the help of their newly developed 'participative knowledge' which is needed for an adequate application of the programs. In this way, new 'user/experts' crop up who develop concepts about that part of reality they refer to. They re-invent the wheel, so to speak, and are guided by this wheel in their practice. (Born, 1987)

.... the promise is bright so long as the software and its pre-understandings are clearly understood. .... The utility of [an expert system] would lie not in any indefensible claim that it describes 'how people actually think', but that it would be easily manipulated in a way that could prove fruitful in sessions with informants. .... As the [expert system] takes shape, its properties could be explored with informants, who can scrutinize the system's performance and point out when it performs at variance with native sensibilities. .... Such tools can come into their own in qualitative research .... only when a thorough attempt is made to see them as they are. Like language, technology is our creation, and we imbue it with a symbolism whose potency we are only now beginning to guess. (Pfaffenberger, 1988, p.77).

It should be clear from this last quote that expert systems have the capability to expand community participation and instill a greater public awareness concerning the intricacies of the
planning process. They may provide the planner with more suitable tools for presenting planning issues to communities and for establishing a basis from which discussions can take place. Given this role, the expert system becomes an element within the planning process rather than an end in itself.

Finally expert systems may provide planning departments with a strategy for combating budgetary restraint and increased workloads by allowing time consuming, repetitive, and relatively simple tasks (e.g. giving out zoning information over the counter) to be handled by very junior and technical staff in consultation with an expert system (e.g. in terms of zoning matters, in consultation with a system similar to but more complex than the one developed here). Junior and middle planning staff could be employed more productively, and more often, in 'meatier' work such as policy formulation, long range plan preparation, community involvement, plan monitoring and analysis, etc., and thus, expert systems could perhaps help to redress the imbalance observed by Irwin & Farrow (1989). The danger here, though, is that Expert Systems could be seen by Municipal Administrations as a means of facilitating budgetary and staff reductions rather than as a vehicle for making planning potentially more effective and this is something of which all Planners interested in expert systems should be aware.
CHAPTER SEVEN: CONCLUSIONS

The research contained in this thesis set out to achieve two broad objectives. The first was to provide a synthesis of the existing literature on expert systems, and expert systems in planning, so as to provide a concise but moderately comprehensive basis from which professional planners could assess expert systems technologies and identify potential areas of application within their own spheres of work. Indeed, a somewhat surprising finding of the literature review was the relative abundance of planning literature on the subject of expert systems even though the topic has only been under study for some five years.

Since planning interest into expert systems to date has, to a large extent, been based upon a purely theoretical analysis of their potential, the second objective was to expand the boundaries of current planning research by actually developing a very simple expert system based upon the obvious, but so far relatively unpopular, domain of evaluating development applications. Indeed, the development of Euclid has shown that expert systems may have some potential in planning domains such as development control and that the notion of expert systems applications in urban planning is one which merits further research at the academic level.

Euclid has proven to be a useful exercise since it has helped to provide a much clearer picture of the requirements that Expert Systems must meet if they are to be of use to Planners. In short, the results can be summarized as follows:

- There does appear to be some potential for Expert Systems in Planning and Development Control.
- The use of fully developed expert systems, designed to assist in checking development applications, may result in time savings during the approvals process and may allow staff resources to be deployed on other tasks.
- Expert Systems may have a role in Planning in helping to detect costly human errors and train new members of staff.
• Expert Systems may have the potential to allow Planners to ask "What if?" questions and to develop new regulations and evaluate changes to the Zoning By-law in terms of their effectiveness for controlling development.

• At the present time it is far too early to determine the potential, if any, of Expert Systems in Planning. Certainly, the technology is, at the present time, in its infancy and much more sophisticated software will have to be developed if Expert Systems are to be of real use to Planners in a variety of areas.

A number of other findings also stand out as a result of conducting this research. Firstly, the process of developing Euclid has shown that expert systems, albeit crude in form, can be developed quite easily using a tool like Turbo Prolog, without the expenditure of significant resources and without the need for significant programming experience. Such a finding is good news for the future of expert systems in planning since the opportunity cost of exploring potential applications does not appear to be excessive.

Secondly, development control has proved itself a good domain in which to target such expert systems research since it fulfills most of the criteria which are needed if an expert system is to be both viable and useful. However, even the relatively structured domain of assessing development applications has proven to be less structured and more open-ended than at first thought. This is due in large part to the discretionary freedom which is provided for in the Zoning Schedules. As a consequence Expert Systems may be even harder to build than first imagined. Nevertheless, it is still somewhat surprising that not more effort has been put into developing expert systems in this sphere since there does appear to be some potential for savings in time and in the consistency of the approvals process in so far as expert systems could detect and possibly prevent human errors. Such savings and improvements may be large since most planning departments spend an inordinate amount of time carrying out administrative and development control tasks. Furthermore, the limitations to departmental resources, and the large
number of development applications which are submitted annually, do appear to offer an incentive for building expert systems in the sphere of development control.

Thirdly, while demonstration systems such as Euclid are not fully functional and/or ideal planning support tools, they do appear to provide an excellent way in which to learn about the domain, grapple with the problems which arise, and for identifying which capabilities a fully functional expert system must have if it is to be of everyday use to planners in the work place. Such an empirical approach is necessary at the present time since very few planning expert systems have been developed and we are at the stage where planners need to expand their expertise in this area if potentially useful tools are to be developed.

Fourthly, the choice of software appears to play a critical role in the success and ease with which experimental expert systems are developed. While Euclid has proven to be a moderately successful exploratory tool, it lacks some of the refinements which average expert systems shells can provide. Turbo Prolog has proven to be quite adept at expressing the intricacies of Zoning By-laws but would require the expenditure of more programming effort if a highly flexible and comprehensive tool were to be created. The lesson here appears to be that planners may find it easier to explore the possibilities if they use expert system shells for the initial development work rather than one of the AI languages.

Not only does Euclid point towards the usefulness and potential of expert systems in and of themselves, but it also suggests that the process of development may be just as valuable. Indeed, there appears to be some support for the notion that the process of developing expert systems is more important than the resultant system. The development of Euclid has lead to two
findings which, even if Euclid is extended or improved no further, still remain as valuable contributions to the planner's understanding of development control and Zoning By-laws.

The first of these has been a greater understanding of how Zoning By-laws are supposed to control development. The development of Euclid forced a rigorous logical analysis of the zoning schedules and revealed that the Zoning By-laws implicitly define a process by which development proposals should be evaluated - Outright approval should be checked before Conditional approval, for example. In following this implicit process and in trying to translate the schedules into an equivalent logical form, Euclid provides the means by which the logical consistency of the By-laws themselves can be evaluated. Euclid thus suggests that expert systems may be useful as heuristic devices for evaluating those elements of the planning process which are taken for granted.

The second finding in this regard, which is a by-product of the first, is the discovery of a couple of weaknesses and inconsistencies in the way Zoning By-laws are currently written. Euclid has revealed that Zoning By-laws can be unintentionally inconsistent and thus regulations which are, to all intents and purposes, identical in their intent may be written in slightly different styles depending upon who is tasked with drafting or amending the By-law and when. This finding suggests that greater thought and attention should be paid to the relationships between zoning schedules as well as to the overall consistency of the By-law itself. In addition, the legal constraints placed upon the contents of a By-law lead, in some cases, to regulations which, at their worst, are incomprehensible to the uninitiated or, at their best, confused. Such constraints, as well as the lack of any supporting graphics, also mean, in some cases, that the clarity of the
regulations is obscured and the resultant confusion may, in part, be responsible for making the approval process lengthier than need be.

While there appears to be, on the basis of the work presented here, some future for expert systems in planning, the strength of the arguments put forward by several authors critical of expert systems and the AI community as a whole (see Chapter Six) means that planners should be cautious in the way expert systems are developed and used. It should be stressed that expert systems are in their infancy and that the technology is still very limited. The failure of Euclid to meet all of the criteria, as discussed in Chapter Five, is due in part to the author's limited programming experience and to the use of a language rather than a expert system shell. However, it also suggests that Expert Systems techniques and software need to be improved and expanded before really useful and practical tools can be developed for the Planning profession; certainly, a simple production-rule system like Euclid appears to be insufficient. Such systems could also be a highly volatile force in redefining the role of the planner and the nature of the planning process. In unleashing expert systems upon the profession, planners must be very careful in making sure that the technology serves their needs and not the other way around. One way of addressing this issue is to develop these systems from within a strong theoretical and normative perspective. In other words, planners involved in the development of expert systems must have a solid grasp of contemporary planning practice so as not to develop systems in a theoretical vacuum.

On a more practical level, planners should be aware that there are serious legal ramifications involved with systems such as Euclid. This is especially so where the use of such an expert system could possibly lead to a development application recommendation (either for
or against) which later turns out to have been incorrect. Furthermore, the advent of expert systems in planning could lead to planners becoming dependent upon the technology and thus to a loss of professional skills in the planning workforce, especially if the expert systems introduced operate in a 'black box' fashion. Finally, it is possible that Expert Systems might be viewed by administrations as a means of cutting budgets and staff levels rather than as potentially liberating tools.

So, where does this leave us? Clearly, there is some interest on the part of academic planners worldwide in the potential for expert systems to be developed in planning and related fields. Equally clear, though, is that much more research needs to be done at the academic level, and more advanced pieces of software have to be developed, before truly competent systems can emerge and before systems can be installed with confidence in municipal planning departments and elsewhere. In this regard, the chapter in which Euclid is evaluated suggests numerous avenues for further research and these have arisen out of just one domain - that of development control.

At this early juncture, planning departments should wait until it is absolutely clear that expert systems have a viable place in their day-to-day operation before approving the expenditure of any resources for the development of in-house systems. Thus, the impetus for system development has to come from outside the municipal sphere and yet, while commercial systems developed by Software Developers, and some Planning Consultants, are starting to appear, we are several years, perhaps decades, from of-the-shelf 'intelligent' development control systems. The key to this impasse seems to be the development of more demonstration systems like Euclid but which push the bounds of current knowledge even further. Additionally, research into the
role and uses of Expert Systems within Planning should also help to define the extent to which they may be applied.

In terms of the more distant future, the combination of Expert Systems with Computer Aided Design tools and Geographic Information Systems (G.I.S.) to produce comprehensive Computer Aided Planning systems, is a logical and desirable aim. However, Expert Systems should not be viewed as a panacea and it will be the planner who approaches expert systems (and other new technologies) with cautious optimism and with a healthy degree of skepticism who will most likely contribute the most to research in these areas.
Thesis Bibliography


143


146
APPENDIX 1 ZONING DATA BASE SPECIFICATIONS

RT-2 Zoning District

zone_class("RT-2")
permitted_uses([1,2,3,4,5,6,7,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26])
min_site_area(4800,7200)
ht(30,2,1)
front_yd(24)
side_yards(0.1,5)
side_yds(7)
rear_yard(35)
fsr(0.6,0.75)
site_coverage(0.45)
site_cov(0.40,0.55)
parking_coverage(0.3)
number_of_buildings(3)
accessory_buildings(12,15,5,0.35,520,0.8)

RT-4 Zoning District

zone_class("RT-4")
permitted_uses([1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,20,21,22,24,25])
min_site_area(3300)
height(35,2.5)
front_yd(24)
side_yards(0.1,5)
rear_yard(35)
fsr(0.6,0)
site_coverage(0.45)
parking_coverage(0.3)
accessory_buildings(12,15,5,0.35,520,0.8)

RT-4N Zoning District

zone_class("RT-4N")
permitted_uses([1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,20,21,22,24,25])
min_site_area(3300)
height(35,2.5)
front_yd(24)
side_yards(0.1,5)
rear_yard(35)
fsr(0.6,0)
site_coverage(0.45)
parking_coverage(0.3)
accessory_buildings(12,15,5,0.35,520,0.8)
note_level(35,40,45,60)
APPENDIX 1 ZONING DATA BASE SPECIFICATIONS (continued)

RT-5 Zoning District

zone_class("RT-5")
permitted_uses([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25])
min_site_area(3300)
max_frontage(105)
height(35,2.5)
front_yd(24)
Aside_yards(0.1,5)
rear_yard(35)
fsr(0.6,0.75)
site_coverage(0.45)
parking_coverage(0.3)
accessory_buildings(12,15,5,0.35,520,0.8)

RT-5N Zoning District

zone_class("RT-5N")
permitted_uses([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25])
min_site_area(3300)
max_frontage(105)
height(35,2.5)
front_yd(24)
side_yards(0.1,5)
rear_yard(35)
fsr(0.6,0.75)
site_coverage(0.45)
parking_coverage(0.3)
accessory_buildings(12,15,5,0.35,520,0.8)
noise_level(35,40,45,60)
RT-6 Zoning District

zone_class("RT-6")
permitted_uses([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22])
min_site_area(3300)
max_frontage(105)
height(35,2.5)
front_yard(0.2,5)
side_yards(0.1,5)
rear_yard(35)
fsr(0.6,0.75)
site_coverage(0.45)
parking_coverage(0.3)
dwelling_unit_density1(30)
dwelling_unit_density2(25)
number_of_buildings(1)
exterior_design(1.3,30,10,3,6.52)
accessory_buildings(12,15,5,0.35,520,0.8)
<table>
<thead>
<tr>
<th>Predicate</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_os-</td>
<td>defines the routine for accessing the operating system from within Euclid.</td>
</tr>
<tr>
<td>ans_base()</td>
<td>this keeps track of whether or not the building proposed has a basement. i.e. it stores the user’s answer regarding the existence of a basement so that the user does not have to be asked the same question more than once.</td>
</tr>
<tr>
<td>ans_cell()</td>
<td>this keeps track of whether or not the building proposed has a cellar and functions in the same way as ans_base().</td>
</tr>
<tr>
<td>ans_ht()</td>
<td>this stores the height of the proposed/existing building to prevent having to ask the user redundant questions.</td>
</tr>
<tr>
<td>ans_storeys()</td>
<td>in the same way as ans_ht(), this stores the number of storeys of the building proposed or in existence.</td>
</tr>
<tr>
<td>ans_track()</td>
<td>keeps track of answers the user supplies while the system is checking for outright uses so that these can be used in checking for conditional approval, if need be, without having to ask the user for the same information again.</td>
</tr>
<tr>
<td>check_acc-gfa()</td>
<td>checks to make sure that the gross floor area of any accessory building is within the limits prescribed by the zoning schedule concerned.</td>
</tr>
<tr>
<td>check_acc_roof()</td>
<td>checks to make sure that the height of the roof for any accessory building is within prescribed limits.</td>
</tr>
<tr>
<td>check_acoustics1()</td>
<td>checks to see if the development proposed conforms with acoustical requirements in zones where noise intrusion is deemed a problem.</td>
</tr>
<tr>
<td>check_ansX()</td>
<td>checks answers supplied by the user to make sure they conform to the format required. i.e. this predicate traps input errors where yes/no responses are required.</td>
</tr>
<tr>
<td>check_conditional_app()</td>
<td>this predicate takes the use selected by the user and tests to see if it is allowed conditionally in the zone in question.</td>
</tr>
<tr>
<td>check_dwellings_unit_density1()</td>
<td>checks to make sure that the number of dwelling units proposed are within the density requirements specified for the zone in question.</td>
</tr>
<tr>
<td>check_external_designX</td>
<td>checks to make sure that the building(s) proposed meet basic design requirements in terms of their exterior appearance and that they fit in with surrounding buildings.</td>
</tr>
</tbody>
</table>

150
check_front_yard - checks to make sure that the front yard proposed meets minimum depth requirements and thus that the buildings proposed are set back from the street in the same way as the surrounding buildings.

check_fsr() - checks to make sure that the total floor space area of the proposed building(s) is within the floor-space-ratio requirements for the zone in question.

check_height - checks to make sure that the building(s) proposed are within the height envelope specified or that (in some cases) they are of a similar height to surrounding buildings if those buildings are above the height maximum.

check_horizontal_sun_angle() - checks to make sure that the units proposed have adequate exposure to sunlight.

check_number_of_buildings - returns a message informing the user that more than one principal building per site can be allowed but that certain requirements must be met. It also ascertains whether or not a rear yard relaxation is required.

check_outright_app() - this predicate checks to see if the use selected by the user is allowed outright for the zone in question.

check_rear_yard - checks to see if the development proposal meets minimum rear yard standards for the site/use/zone combination in question.

check_regs() - this predicate lists those regulations which apply to specific zone/use combinations. Once the use as been established as being allowed either outright or conditionally then this predicate evaluates all those regulations which apply to the zone/use combination selected.

check_rest_of_regs() - lists all the regulations which apply to a specific zoning schedule. This predicate is used to make sure that the development proposal meets regulations not covered by the check_regs() predicate - i.e. the non-use specific regulations.

check_side_yards() - checks to make sure that the building(s) proposed are set back a sufficient distance from the sides of the lot.

check_site_coverage() - this predicate checks to make sure that the footprint of the building(s) proposed are less than or equal to the maximum site coverage area allowed. It also checks to see if the space proposed for surface parking is within the area allowed.

check_use() - defines the procedure to be used in checking for zoning approval. This predicate determines that outright approval for the use will be checked first followed by the zoning regulations relating to outright uses. If these fail check_use() will go on to check for conditional approval for the use and will then
go on to check the zoning regulations governing conditional uses. check_use() also provides the mechanism for informing the user of the main outcome of an evaluation - i.e. whether the use is approved outright, conditionally, or not at all.

do() - this predicate translates what the user selects from the main menu into a series of actions which Euclid can understand.

defines the routine which brings the user back to the main_menu once the evaluation session has reached a conclusion.

error_msg() - describes and defines messages which appear on screen in the event of a user input error.

find_use- presents the user with a list of uses to choose from. If the use selected is not allowed in the zone selected, find_use will inform the user and ask them if they want to check to see if the use proposed conforms with the zoning regulations for the zone selected. (The Director of Planning may permit a use not expressly permitted under the zoning schedule provided it meets the necessary regulations).

get_database() - takes the zone selected by the user and retrieves the corresponding database containing details of the zoning specifications. It then loads this information into an internal database within Euclid for use by other predicates within the system.

get_dimensions - asks the user a series of questions about the dimensions of the site. It takes the figures supplied by the user and stores them in an internal database within Euclid designed for this purpose. In this way site dimensions can be referenced by accessing the internal database rather than asking the user for the figure(s).

get_relaxation() - acts as a filter to ensure that (in the event of a development proposal failing to meet a zoning regulation for which a relaxation may be permitted) only those relaxations which pertain to the failed regulation, and to the particular zone/use combination selected, are reported to the user.

main_menu- this predicate lists the action choices the user can select from the main menu window.

member() - this predicate is a recursive predicate which checks to see if the use selected by the user is allowed in the zone selected. The database for each zoning schedule contains a list of uses and member() checks to see if the use selected is a member of the list for the zone in question.

modify_the_database - The predicate at the moment just creates a window from which the user would be able to modify information contained in the zoning databases. This predicate is cosmetic only at present.
proposed_use() - keeps track of the use selected by the user so that Euclid does not have to keep asking for the same information at every question.

query_the_user - defines the actions Euclid carries out if the user selects the "Check the Zoning on a Property" option from the main menu. This predicate lists the zones which are contained within Euclid's knowledge base. The property in question has to be in one of the six two-family zoning districts in order for the system to work. The user is then asked to select the relevant zone - query_the_user then tells Euclid to get the relevant zoning database and starts the evaluation process rolling.

quit - exits Euclid and returns the user to the Operating System - i.e. it returns control to DOS.

relaxation() - defines the relaxations which apply for certain zone/use combinations in relation to specific zoning regulations. Each relaxation() predicate identifies a text string which Euclid reports to the user in the event that that relaxation applies.

test() - acts as a filter to ensure that the check_external_design1 predicate is only triggered if zone in question is RT-6 since the external design regulations only apply for this zone.

testX() - this predicate checks to see what answers the user has provided to certain questions at an earlier stage in the interactive session. It acts as a filter within the check_site_coverage3() predicate since the answers which testX() checks determine which portions of the check_site_coverage3() predicate are to be evaluated.
APPENDIX 3 - EUCLID PROGRAMMING CODE

/* EUCLID.PRO */
/
/*****************************************************************************
*/ /*
/*
/*
/*
/*
/*
/*****************************************************************************
/*
/
/* PROGRAM DOMAINS */
/*
/
/*
/
/*
/*
/
/*
/
/*
/
/*
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/
/ The PROGRAM DOMAINS section specifies the kind of information which each predicate can handle. The standard types (or DOMAINS) are: integers, reals, strings, characters, & symbols. Special kinds of information (e.g., zone, use, height) can also be defined in terms of the 5 standard domains. This makes the program more readable.
/
/* code = 1250 */
/ indicates that the code array memory must be set at 1250 paragraphs in order to be able to compile
/* EUCLID to an executable file. N.B. anything between */ and */ or after a '//' is treated as a comment and is not compiled.
/
/* project "EUCLID" indicates that this file forms a part of a larger program called EUCLID.
/
/* 'include ...' means that the file GLOBALS.PRO is to be incorporated into EUCLID.PRO. In this case GLOBALS.PRO defines the global domains, predicates, and databases which are shared by all the files which make up EUCLID.PRO

/* code = 1250 */
/project "EUCLID"
/include "c:\\programs\\tprolog\\globals.pro"
/
/*****************************************************************************
*/ PROGRAM PREDICATES */
/
/* PROGRAM PREDICATES */
/
/* The PROGRAM PREDICATES section lists all the user defined predicates which describe the facts, rules, and inferencing routines used in the program. The predicates listed below are local and are used primarily to control the main menu, start the evaluation of development proposals, and display the results.
/
predicates
  main_menu
  query_the_user
  find_zone
  modify_database"
The PROGRAM CLAUSES section describes precisely how each of the predicates functions.

% PRINCIPAL CLAUSES
% 'main-menu' - displays the main menu on screen and reads the user's selection.
% 'do(integer)' - starts to carry out the user's choice.
% 'query_the_user' - asks the user to select the appropriate zone.
% 'get_database(integer)' - loads the required database file into the database 'zoninginfo'.
% 'get_dimensions' - asks the user to enter in lot dimensions and stores responses in the 'dimensions' database.
% 'find_use' - asks the user to select the use proposed from a list of uses and tests to see if the use selected is valid by calling 'member(integer, useslist)'.
% 'check_use' - checks the selected zone and use against approved outright and conditional uses for that zone. If the use is approved it goes on to check the regulations via 'check_regs' and 'check_rest_of_regs'. Depending on the outcome of the evaluation 'check_use' also defines the various results which can be returned to the user.

% SECONDARY CLAUSES
% 'member(integer, useslist)' - compares the use selected against the list of uses allowed for the zone in question.
% 'error_msg(integer)' - handles in a limited way input errors by the user.
% 'check_ans(integer)' - provides universal answer testing and error checking.
% 'test(zone)' - ensures that only conditional uses in the RT-6 zone have to meet external design requirements.

% NOTES
% Commas - ',' - are treated as logical AND statements and Semi-Colons - ';' as logical OR statements.
% The '-' symbol means IF and separates the HEAD of a rule from its BODY. The rule takes the form 'HEAD IF BODY' - i.e. if the BODY is TRUE the HEAD is true.
% The period - '.' - indicates the end of a rule or fact.
The 'fail' predicate forces backtracking by automatically failing. The system then goes on to look for alternative solutions. In most cases within EUCLID though 'fail' is used to direct the system to produce a result.

The ',•' symbol (or CUT) is the opposite of 'fail' and prevents backtracking. The cut is extensively used in EUCLID to prevent the evaluation of alternatives in rules with OR statements or to prevent rules with the same HEAD from being called should the current rule fail.

If the head of any other rule appears in the body of the current rule, Turbo Prolog calls that rule and checks to see if its body is true before going on to check the remaining elements in the body of the current rule.

Where an underscore - '_-' - takes the place of an argument (e.g. 'check_use(_,_)' or 'check_outright(_,12)') it is like a wildcard. The underscore will automatically match the value of the corresponding argument in the calling clause. Thus 'check_outright_app(_,12) :- BODY' is the same as saying 'if the use proposed is 12 check the BODY of the clause, regardless of what the zoning district is'. From this it should be clear that the '_-' symbol helps in creating rules which can be used in more than one situation.

Arguments that are variables are indicated by a Capital first letter, thus 'LotArea' is a variable which is bound to the value the user supplies when asked by 'get_dimensions' clause to provide the area of the lot. Subsequently, whenever LotArea appears at later stages of the program it is treated as having that value supplied by the user. Similarly unbound (or free) variables can take on the value of bound variables or of static objects (arguments that are numbers or start with a lower case letter).

Note that there are few true objects or facts in the EUCLID system since most of the information is handled by variables and supplied by the user interactively or read into the database files. The two 'get relaxations' rules are in essence facts and each has two object arguments corresponding to an identity number identity and use specifying the particular relaxation.

Blank spaces in Turbo Prolog are ignored by the compiler thus allowing the code to be arranged for legibility. In addition, when trying to 'prove' a clause Turbo Prolog works from top to bottom looking for the head of a rule that matches the clause. Where there is more than one clause with the same head, Turbo Prolog works from top to bottom trying to find a match. If the head of a rule matches the clause, Turbo Prolog tries to satisfy the body of that rule.

clauses

main_menu :- shiftwindow(5), clearwindow,n1,
write("\tCHECK THE ZONING ON A PROPERTY\t (1)"),n1,n1,
write("\tFIND A ZONE FOR A PROJECT\t (2)"),n1,n1,
write("\tMODIFY OR ADD TO THE DATABASE\t (3)"),n1,n1,
write("\tACCESS THE OPERATING SYSTEM\t (4)"),n1,n1,
write("\tEND THIS SESSION\t (5)"),n1,
write("----------------------------------------"),n1,
write("\tENTER YOUR CHOICE:\t\t"),readint(Choice), do(Choice).

do(1) :- !, query_the_user.
do(2) :- !, find_zone.
do(3) :- !, modify_database.
do(4) :- !, access_os.
do(5) :- !, makewindow(11,7,0,"",0,0,25,80), clearwindow, quit.
do(_) :- shiftwindow(7), clearwindow,error_msg(3), readint(Choice), do(Choice).
query_the_user :- shiftwindow(2),clearwindow,shiftwindow(8),clearwindow,n1, write(" Please select the Zoning Schedule which applies to the property\n in question: "),cursor(R,C),n1,n1, write(" (1) RT-2\n (2) RT-4\n (3) RT-4N\n (4) RT-5N\n (5) RT-5\n (6) RT-6\n"),cursor(R,C), retractall(_ zoninginfo), readint(X), get_database(X), shiftwindow(2), n1, get_dimensions, find_use.

find_zone :- makewindow(12,7,15,"FINDING A ZONE FOR A PROJECT",0,0,25,80,1,-1,"\201\187\200\188\205\186"), clearwindow,n1,n1, write(" The current version of EUCLID does not support this facility. If you have\n"), write(" a development project in mind and want to find which zones support it you\n"), write(" must select choice 1 from the MAIN MENU and try each of the zones listed in\n"), write(" turn to see how they fit the proposed project.").end.

modify_database :- makewindow(6,7,15,"MODIFY/ UPDATE THE DATABASES",0,0,25,80,1,-1,"\201\187\200\188\205\186"), clearwindow,n1,n1, write(" The current version of EUCLID does not support this facility. If you wish\n"), write(" to see the contents of the database and/or edit/update them you must select\n"), write(" choice 4 from the MAIN MENU which will access the operating system (DOS).\n"), write(" You may then use standard DOS commands and the built-in editor EDLIN\"), write(" (or another suitable editor) to modify the files.\n"), write(" NOTE: WHILE THE DATABASE FILES ARE BACKED UP, CHANGING THEM MAY PRODUCE\n"), write(" FATAL ERRORS AND PREVENT THE SYSTEM FROM WORKING.").end.

access_os :- system(""), main_menu.
quit :- exit.

end :- !,shiftwindow(7),clearwindow,n1, write(" Press the SPACE bar"),
readchar(Y), check_ans3(Y),shiftwindow(1),clearwindow,
shiftwindow(3), main_menu.

get_database(1) :- !, consult("RT2.dba", zoninginfo).
get_database(2) :- !, consult("RT4.dba", zoninginfo).
get_database(3) :- !, consult("RT4N.dba", zoninginfo).
get_database(4) :- !, consult("RT5.dba", zoninginfo).
get_database(5) :- !, consult("RT5N.dba", zoninginfo).
get_database(6) :- !, consult("RT6.dba", zoninginfo).
get_database(_ ) :- error_msg(3), readint(X), get_database(X).

error_msg(1) :- shiftwindow(7),clearwindow,n1, write(" Your Choice is not valid,\n"), write(" please press the SPACE bar: "),
readchar(X), check_ans3(X),!,shiftwindow(2),write(" ? ").

error_msg(2) :- shiftwindow(7),clearwindow,n1,n1, write(" Your input is not valid,\n"), write(" please press the SPACE bar: "),
readchar(X), check_ans3(X),!,shiftwindow(2),clearwindow,n1, get_dimensions.
error_msg(3) :- shiftwindow(7),clearwindow,n1, write(" Your Choice is not valid, please try again:
").

check_ans1('y') :- !.
check_ans1('n') :- !, fail.
check_ans1(_) :- error_msg(1), readchar(Y), write(Y),n1,n1, check_ans1(Y),!.

check_ans2('n') :- !.
check_ans2('y') :- !, fail.
check_ans2(_) :- error_msg(1), readchar(Y), write(Y),n1,n1, check_ans2(Y),!.

check_ans3('y') :- !.
check_ans3('n') :- error_msg(1), readchar(Y), write(Y),n1,n1, check_ans3(Y),!.

get_dimensions :- write(" Please enter the following information about the property in questions:"),n1,
write(" Enter Front Lot Width in ft: "), readreal(Front),
write(" Enter Left Lot Length in ft: "), readreal(Left),
write(" Enter Rear Lot Width in ft: "), readreal(Rear),
write(" Enter Right Lot Length in ft: "), readreal(Right),
check_site_shape(Front,Left,Rear,Right,Lotarea),
write(" Enter Area of Ground Floor in sqft: "), readreal(FlrArea),
write(" Enter Total Floorspace in sqft: "), readreal(Flrsp1),
test_floorspace(FlrArea,Flrsp1,FloorArea,Floorspace),
write(" Enter Depth of Front Yard in ft: "), readreal(Frontdepth),
write(" Enter Depth of Rear Yard in ft: "), readreal(Reardepth),!,
retractall(_,dimensions),
asserta(dimensions(Front,Left,Rear,Right,LotArea,FloorArea,Floorspace,Frontdepth,Reardepth),dimensions); error_msg(2).

check_site_shape(_,_,_,_,Lotarea) :- !,
write(" THE SITE IS NOT RECTANGULAR IN SHAPE.

"), write(" Please Enter Total Lot Area in sqft:"), readreal(Lotarea),n1.

check_site_shape(_,_,_,_,Lotarea) :- !, write(" The Total Area of the Lot is: "),Lotarea,n1,n1.

test_floorspace(FlrArea,Flrsp1,X,Y) :- Flrsp1 >= FlrArea,!, X = FlrArea, Y = Flrsp1;
write(" It is unusual for Total Floorspace to be less than the area of

"), write(" the ground floor. Have you entered each figure correctly? (y/n) " ),
readchar(Z),write(Z),n1,n1, check_ans3(Z),! , X = FlrArea, Y = Flrsp1;
write(" Please Re-Enter Ground Floor Area in sqft:"), readreal(X),
write(" Please Re-Enter Total Floorspace in sqft:"), readreal(Y),n1.

find_use :- clearwindow,n1,
write(" What kind of use is proposed?

(1) Accessory Bldg/Use (2) Community Centre (3) Neighbourhood House

(4) Park (5) Playground (6) Material Deposition (7) Material Extraction (8) Infill

(9) Multi Dwelling (10) Multiple Dwelling (11) One-Family Dwelling


(18) Social Service Centre (19) Special Needs Res. (20) Nrhd Grocery Store (21) Bed & Breakfast Accom.

"),
write("\n(22) Public Utility
(23) Club
(24) Ambulance Station
(25) Hospital
(26) Parking Area
(27) Other\n\n(22) Public Utility
(23) Club
(24) Ambulance Station
(25) Hospital
(26) Parking Area
(27) Other")
retractall(_,storage)
writeint(X)
asserta(proposed jjse(X) ,storage)
p e r m i 11 e s (U s e) , m e m b e r • ( X , U s e) ,
z o n e__c1 a ss(Z one) ,
check__u.se(Z one, X)
write( "The use proposed is not expressly permitted, either outright or

conditionally, under the "Use" schedule but may still be allowed by the Director of Planning.

Do you want to check the regulations to see if the use may comply? (y/n)\n"
readchar(X) , write(X),n,check_ans2(X),write( "This ends the current session.\n"
end;
check_rest_of_regs(Z one)
write(" CHECKING REGULATIONS FOR PRINCIPAL BUILDINGS & SITE: ...
"
)
write(" The use proposed is not permitted outright under the "Use" schedule but may still be allowed by the Director of Planning.

This ends the current session.")
end.

member(X , [X;_3]).
member(X , [X, Tail]) :- member(X, Tail).

check_use(Z one, X) :-
shiftwindow(1),clearwindow,n,
write(" CHECKING PROPOSED USE: ...
"
)
write(" CHECKING REGULATIONS FOR PRINCIPAL BUILDINGS & SITE: ...
"
)
write(" The use proposed is permitted outright but must still be approved by the Director of Planning.

Do you wish to check for conditional approval? (y/n)\n"
readchar(Y) , write(Y),n,clearwindow, write(" The use proposed is permitted conditionally but must still be approved by the Director of Planning.

This ends the current session.")
end.

check_use(Z one, X) :-
shiftwindow(2),clearwindow,n,
write(" CHECKING PROPOSED USE: ...
"
)
write(" CHECKING REGULATIONS FOR PRINCIPAL BUILDINGS & SITE: ...
"
)
write(" The use proposed is permitted conditionally but must still be approved by the Director of Planning.

This ends the current session.")
end.
write(" be presented to The Direction of Planning for confirmation."); nl, nl, 
write(" This ends the current session."); end.

check_use (_) :- shiftwindow(1), nl, zone_class(Zone), write(" The use proposed is not permitted, either outright or 
"), write(" conditionally, under the ", Zone, 
" schedule but may still be allowed by the Director of Planning."); nl, write(" This ends the current session."); end.

test("RT-6") :- !, check_external_design1.

test(_) :- true.

/***************************** */ /* INITIAL GOAL STATEMENT */  
/**************************** */

% The GOAL section describes the predicates which are automatically started when the command EUCLID is given at 
% the DOS prompt, once the EUCLID program has been read into the workspace. The program starts with the goal 
% listed below and immediately makes all the windows which will be used while the program is running.
% 
% 'main_menu' starts the program by calling the 'main_menu' predicate listed above.

goal makewindow(9,15,4,"",21,26,3,25), 
makewindow(5,7,7,"MAIN MENU",5,14,16,50,0,-1,"\201\187\200\188\205\186"), nl, 
makewindow(1,7,15,"RESPONSE TO ZONING QUERY",2,5,20,70,1,-1,"\201\187\200\188\205\186"), 
makewindow(3,7,48," EUCLID (1.0) : A ZONING ASSISTANT ",0,0,25,80,1,-1,"\201\187\200\188\205\186"), nl, nl, nl, 
write(" PREFACE TO THE SYSTEM
"), nl, 
write(" This system has been developed as part of the author’s M.A. Thesis 
"), 
write(" researching the technical feasibility of building Expert Systems
"), 
write(" and their potential utility for handling Urban Planning tasks.
"), 
write(" The system was developed at the School of Community and Regional
"), 
write(" Planning, at the University of British Columbia, during the spring
"), 
write(" and summer of 1990, in partial fulfillment of the requirements of
"), 
write(" PLANNING 549 and towards the Degree of Master of Arts (Planning). 
"), 
write(" Julian Atkins, 

July, 1990
"), shiftwindow(9), nl, write(" Press the SPACE bar "), 
readchar(Y), check_ans3(Y), clearwindow, shiftwindow(3), clearwindow, 
makewindow(2,7,15,"CHECKING A ZONING BY-LAW",0,0,25,80,1,-1,"\201\187\200\188\205\186"), nl, 
makewindow(8,7,15,"SELECT A ZONE",2,5,20,70,1,-1,"\201\187\200\188\205\186"), makewindow(7,15,"ERROR MESSAGE",15,12,5,54), 
shiftwindow(3), makewindow(4,7,15,"",1,3,4,74), 
write(" WELCOME TO EUCLID: 
"), 
makewindow(10,7,15,"",21,14,3,50), 
write(" Copyright (c) 1990 by Julian F. Atkins.
"), 
main_menu.
/* OUTRIGHT.PRO */
%
% OUTRIGHT.PRO contains the rules which are used to check whether or not the use proposed can be approved
% outright. The 'check_outright_app' clauses are called from the 'check_use' clause in EUCLID.PRO.
% The rule 'check_outright_app(_,_) :-' covers uses which are not allowed outright and fails thus forcing
% 'check_use' to try and find an alternative solution - in this case 'check_use' goes to check for
% conditional approval.
%
% 'check_acc_roof' and 'check_acc_gfa' test to see whether or not an accessory use complys with the height
% and/or gross floor area (gfa) requirements as specified in the zoning by-laws. (these requirements are
% same for all 6 zones).
%
% Note that the query the user facility is built into each rule thus ensuring that no superfluous questions
% are asked.
%
project "EUCLID"
include "c:\programs\tprolog\globals.pro"

predicates
  check_acc_roof(acc_roof)
  check_acc_gfa(real)

clauses

check_outright_app("RT-5N",X) :- !, check_outright_app("RT-5",X).

check_outright_app("RT-4N",X) :- !, check_outright_app("RT-4",X).

check_outright_app(_,1) :- write(" What type of roof does/will the ACCESSORY building have?
"), write(" Choose one of the following: (1) Flat, (2) Mansard, (3) Gable\n"),
cursor(4,50), readint(Roof), cursor(0,0), accessory_buildings(_,MaxHt,Setback,_,_,_,Ratio),
write(" Is any part of the roof higher than ",MaxHt," ft? (y/n)\t"), readchar(X), write(X),nl,n1,
check_ans2(X),
check_acc_roof(Roof),
write(" Is the accessory building located in the rear yard? (y/n)\t"), readchar(Y), write(Y),nl,n1,
check_ans1(Y),
write(" What is the set back of the accessory building in ft?\t"), readreal(Z),nl,Z >= Setback,
write(" What is the gross floor area of the accessory building in sqft? "), readreal(GFA),nl,
check_acc_gfa(GFA),
write(" How wide is the accessary building in ft?\t"), readreal(Width),nl,
dimensions(_,_,_,Rear,_,_,_,_,_),
Width/Rear <= Ratio,!

check_outright_app(_,1) :- !, shiftwindow(1),clearwindow,nl,fail.
check_outright_app("RT-6",9) :- write(" Does the conversion include any additions? (y/n)"), readchar(X),
assertz(Ans Track(X,'_'), storage), write(X),nl,check_outright_app("RT-6",11),
write(" Will any housekeeping or sleeping units be created? (y/n)"), readchar(Y),
assertz(Ans Track(X,Y),storage), write(Y),nl,check_outright_app(Y),!.

check_outright_app("RT-6",9) :- !, shiftwindow(1),clearwindow,nl,fail.
check_outright_app("RT-6",11) :- !, fail.
check_outright_app(_1,11) :- !,true.
check_outright_app("RT-6",12) :- !, fail.
check_outright_app(_,12) :- !,true.
check_outright_app(_,__) :- shiftwindow(1),clearwindow,nl,fail.
check_acc_roof(1) :- !, write(" How high is the roof of the accessory building at its
"), write(" highest point in feet? " ),readreal(X),nl,
accessory_buildings(Height,_,_,_,_),
X <= Height.
check_acc_roof(2) :- !, write(" How high is the roof of the accessory building at its deck line?"), readreal(X),nl,
accessory_buildings(Height,_,_,_,_),X <= Height.
check_acc_roof(3) :- !, write(" What is the mean height of the accessory building's roof,\n"), write(" from eaves to ridge? " ),readreal(X),nl,
accessory_buildings(Height,_,_,_,_),X <= Height.
check_acc_gfa(GFA) :- dimensions(,_,_,RearWidth,_,_,_,_,_,_,_,_),
accessory_buildings(,_,_,_,,,Percent,Sqft,_),
GFA/(RearDepth * RearWidth) <= Percent,!
accessory_buildings(,_,_,_,_,_,_,_,Sqm,Sqft),
GFA <= Sqft.
/* CQMD
ITML.PRO */

% CONDITNL.PRO contains the rules for testing to see whether of a not a use meets the requirements for conditional approval. 'check_use' calls 'check_conditional_app' if its call to 'check_outright' fails.
%

project "EUCLID"
include "c:\\programs\\tprolog\\globals.pro"

clauses

check_conditional_app("RT-5N",X) :- !, check_conditional_app("RT-5",X).
check_conditional_app("RT-4N",X) :- !, check_conditional_app("RT-4",X).
check_conditional_app("RT-2",8) :- !, fail.
check_conditional_app(_,_) :- write(" Are the dwelling units proposed in conjunction with a neighbourhood grocery store? (y/n) "), readchar(Y), write(Y),nl,nl, check_ans1(Y),!
write(" How many units are proposed?"), readint(N),nl, N <= 2,
write(" Do they conform to section 11.16 of the Zoning By-law? (y/n) "), readchar(Z), write(Z),nl,nl, check_ans1(Z),
check_conditional_app(_,20).

check_conditional_app("RT-6",9) :- !, ans_track(A1,A2), check_ans1(A1),
write(" Are the additions included in the proposed conversion in keeping with the"),
write(" character of the existing building? (y/n) "), readchar(X), write(X),nl,nl, check_ans1(X),!, A2 < 'y',
write(" Will any housekeeping or sleeping units be created? (y/n) "), readchar(Y), write(Y),nl,nl, check_ans2(Y), check_conditional_app("RT-6",8).

check_conditional_app("RT-5",9) :- !, write(" Before making a decision on a Multiple Conversion Use, the Development"),
write(" Permit Board shall consider the quality and livability of the resulting"),
write(" units, the suitability of the building for conversion in terms of age and"),
write(" size, and the effect of the conversion on adjacent properties and the"),
write(" character of the area.

"),
write(" Did the building being converted exist prior to December 8th, 1977? (y/n) "), readchar(X), write(X),nl,nl,check_ans1(X),
write(" Are any additions proposed in keeping with the character of the"),
write(" existing building? (y/n) "), readchar(Y), write(Y),nl,nl,check_ans1(Y),
write(" Will any housekeeping or sleeping units be created? (y/n) "), readchar(Z), write(Z),nl,nl,check_ans2(Z),
check_conditional_app("RT-5",9).

check_conditional_app("RT-4",9) :- !,check_conditional_app("RT-5",9).
Before making a decision on a Multiple Conversion Use, the Development Report shall consider the quality and livability of the resulting units, the suitability of the building for conversion in terms of age and size, and the effect of the conversion on adjacent properties and the character of the area.

Did the building being converted exist prior to June 18th, 1956? (y/n)

How many dwelling units are proposed?

Do at least 50% of the dwelling units in each building have two or more bedrooms?

Are the dwelling units designed specifically for senior Citizens housing or a similar use? (y/n)

Do at least 50% of the dwelling units in each building have two or more bedrooms?

Does the proposed Church comply with section 11.7 of the Zoning By-law? (y/n)

Is the Public Authority Use proposed essential to the District? (y/n)

Does the proposed Elementary School comply with section 11.8?
check_conditional_app(_,17) :- !, write(" Does the proposed Secondary School comply with section 11.8\n"), write(" of the Zoning By-law? (y/n)\t"), readchar(X), write(X),nl,nl, check_ans1(X).

check_conditional_app("RT-6",19) :- !, write(" Does the proposed Special Needs Residential Facility comply\n"), write(" with section 11.17 of the Zoning By-law? (y/n)\t"), readchar(X), write(X),nl,nl, check_ans1(X).

check_conditional_app("RT-4",19) :- !, check_conditional_app("RT-6",19).

check_conditional_app("RT-5",19) :- !, write(" Does the proposed Special Needs Residential Facility comply\n"), write(" of the Zoning By-law? (y/n)\t"), readchar(X), write(X),nl,nl, check_ans1(X).


check_conditional_app(_,20) :- !, write(" Did the grocery store exist prior to July 29th, 1980? (y/n)\t"), readchar(Z), write(Z),nl,nl, check_ans1(Z), write(" Does the grocery store comply with section 11.16 of the Zoning By-law? (y/n)\t"), readchar(W), write(W),nl,nl, check_ans1(W).

check_conditional_app(_,21) :- !, write(" Does the proposed Bed and Breakfast Accommodation comply with section\n"), write(" 11.4 of the Zoning By-law? (y/n)\t"), readchar(X), write(X),nl,nl, check_ans1(X).

check_conditional_app("RT-2",23) :- !,write(" Does the proposed Club involve any commercial activities? (y/n)\t"), readchar(X), write(X),nl,nl, check_ans2(X), write(" Are or will adjacent premises be adversely affected? (y/n)\t"), readchar(Y), write(Y),nl,nl, check_ans2(Y).


check_conditional_app("RT-6",24) :- !, fail.

check_conditional_app(_,24) :- !, true.

check_conditional_app("RT-6",25) :- !, fail.

check_conditional_app(_,25) :- !, write(" Does the proposed Hospital comply with Section 11.9\n of the Zoning By-law? (y/n)\t"), readchar(Y), write(Y),nl,nl, check_ans1(Y).

check_conditional_app("RT-2",26) :- !, write(" Is the proposed Parking Area ancillary to a principal use?\n"), write(" on an adjacent site? (y/n)\t"), readchar(X), write(X),nl,nl, check_ans1(X).

check_conditional_app(_,_) :- true.
/* REGS1.PRO */
%
% REGS1.PRO and REGS2.PRO contain the rules which describe the zoning regulations. REGS1.PRO contains
% all of the 'check_regs' clauses and these describe the rules for checking site area requirements for
% the specific zones. REGS1.PRO also contains the 'check_rest_of_regs' rules which determine which
% particular regulations apply for which zone. The remainder of REGS1.PRO consists of rules describing
% each of the zoning regulation requirements. Each rule consists of a series of conditions, usually
% separated by a semi-colon, and a response to be given should the use/site combination fail that zoning
% regulation.
%
% Notice that only the regulations relevant to the zone in question are triggered during a session and that
% most of the conditions require the user to respond to one or more questions.
%
project "EUCLID"
include "c:\\programs\\tprolog\\globals.pro"

predicates
  check_side_yards2(integer)
  check_fsr1(integer)
  check_fsr2(integer)
  check_fsr3(integer)
  check_site_coverage2(integer)
  check_site_coverage3(integer)
  test1(real, char, char, integer)
  test2(real, char, char, integer)
  check_dwelling_unit_density1(integer)
  check_horizontal_sun_angle1(integer)
  check_acoustics1(integer)
  test_use(use)

clauses
check_regs("RT-4N",X) :- !,check_regs("RT-6",X).
check_regs("RT-5N",X) :- !,check_regs("RT-6",X).
check_regs("RT-2",10) :- dimensions(_,_,_,LotArea,_,_,_), min_site_area(_,Sqft), Lotarea >= Sqft, !;
  shifwindow(1), clearwindow,
  zone_class(Zone),nl,
  write(" The use proposed fails to meet site area requirements\n"),
  write(" for the "Zone," schedule.\n\n"),nl,!,
fail.
check_regs("RT-6",10) :- write(" How many adjoining parcels of land make up the property in question?\t"), readint(X),nl, X <= 2,
  write(" Were the current parcel(s) on record in the Land Titles Office\n"),
  write(" before December 6th, 1977? (y/n)\t"),
  readchar(Y), write(Y),nl,nl, check_ans1(Y),
  dimensions(Front,_,_,_,_), max_frontage(Z),
  Front <= Z, !;
shiftwindow(1), clearwindow,
zone_class(Zone), nl,
write("The use proposed fails to meet site area requirements\n"),
write("for the ", Zone, " schedule.\n\n"), nl, fail.

check_regs("RT-5", 10) :- !, check_regs("RT-6", 10).

check_regs("RT-2", 11) :-!, true.

check_regs("RT-6", 11) :- dimensions(_, _, Lotarea, _, _, _),
min_site_area(Sqft), Lotarea >= Sqft, !;
shiftwindow(1), clearwindow,
zone_class(Zone), nl,
write("The use proposed fails to meet site area requirements\n"),
write("for the ", Zone, " schedule.\n\n"),
relaxation(Zone, 5.1, 11), nl, fail.

check_regs("RT-4", 11) :- !, check_regs("RT-6", 11).
check_regs("RT-5", 11) :- !, check_regs("RT-6", 11).

check_regs("RT-2", 12) :- dimensions(_, _, LotArea, _, _, _),
min_site_area1(Sqft, _), Lotarea => Sqft, !;
shiftwindow(1), clearwindow,
zone_class(Zone), nl,
write("The use proposed fails to meet site area requirements\n"),
write("for the ", Zone, " schedule.\n\n"),
relaxation(Zone, 5.1, 12), nl, fail.

check_regs(X, 12) :- !, check_regs(X, 11).
check_regs(_, _) :- true.

check_rest_of_regs("RT-2") :- proposed_use(X), test_use(X), !;
proposed_use(X),
check_height3,
check_front_yard2,
check_side_yards2(X),
check_rear_yard2,
check_fsr3(X),
check_site_coverage3(X),
check_horizontal_sun_angle1(X).

check_rest_of_regs("RT-4") :- proposed_use(X), test_use(X), !;
proposed_use(X),
check_height2,
check_front_yard3,
check_side_yards1,
check_rear_yard1,
check_fsr4,
check_site_coverage2(X).
check_rest_of_regs("RT-4N") :- proposed_use(X), test_use(X), !;  
    proposed_use(X),  
    check_height2,  
    check_front_yard3,  
    check_side_yards1,  
    check_rear_yard1,  
    check_fsr4,  
    check_site_coverage2(X),  
    check_acoustics1(X).

check_rest_of_regs("RT-5") :- proposed_use(X), test_use(X), !;  
    proposed_use(X),  
    check_height2,  
    check_front_yard3,  
    check_side_yards1,  
    check_rear_yard1,  
    check_fsr2(X),  
    check_site_coverage2(X).

check_rest_of_regs("RT-5N") :- proposed_use(X), test_use(X), !;  
    proposed_use(X),  
    check_height2,  
    check_front_yard3,  
    check_side_yards1,  
    check_rear_yard1,  
    check_fsr2(X),  
    check_site_coverage2(X),  
    check_acoustics1(X).

check_rest_of_regs("RT-6") :- proposed_use(X), test_use(X), !;  
    proposed_use(X),  
    check_height1,  
    check_front_yard1,  
    check_side_yards1,  
    check_rear_yard1,  
    check_fsr1(X),  
    check_site_coverage1,  
    check_dwelling_unit_density1(X).

test_use(4) :- !, write("Are any buildings proposed for the site? (y/n)\t"),  
    readchar(Z), write(Z), nl, nl, check_ans2(Z).

test_use(5) :- !, test_use(4).  
test_use(6) :- !, test_use(4).  
test_use(7) :- !, test_use(4).  
test_use(15) :- !, test_use(4).  
test_use(22) :- !, test_use(4).  
test_use(26) :- !, test_use(4).
test_use(27) := !, test_use(4).

check_side_yards2(10) := side_yds(Feet),
write(" How wide are the side yards in feet?\n"), readreal(X), nl, X >= Feet, zone_class(Zone), write(" Are all the outer walls of the principal building(s) within the space\n"), write(" required, as defined by section 4.5.3 of the "Zone," Schedule? (y/n)\n"), readchar(Y), write(Y), nl, nl, check_ansl(Y),!
shiftwindow(1), clearwindow, zone_class(Zone), nl, write(" The use proposed does not meet side yard requirements for the\n"), write(" ", Zone," Schedule. Therefore:")!, nl, !, fail.

check_side_yards2(_ := check_side_yards1.

check_fsr1(8) := dimensions(_, _, LotArea, _, Floorspace, _),
fsr(Ratio1, Ratio2), Floorspace/LotArea <= Ratio2,
Floorspace/LotArea >= Ratio1,!
shiftwindow(1), clearwindow, nl, write(" The use proposed requires an increase in the allowable\n"), write(" floor space ratio. While it is within the maximum increase\n"), write(" allowed it must still be approved by the Director of Planning\n"), write(" who must first consider:"), nl, nl, write(" - The intent of this schedule and well as any applicable policies\n"), write(" - The height, bulk, location, and overall design of the proposal\n"), write(" - and their effect on the site, surrounding buildings, streets,\n"), write(" and existing views.\n"), write(" - The amount of open space and the effect of the proposal on the\n"), write(" - General amenities of the area.\n"), write(" - The preservation of architectural character and the general\n"), write(" - General amenities desired for the area.\n"), nl, write(" SEE MORE? (y/n)\n"), readchar(B), write(B), check_ans1(B), clearwindow, nl, write(" Therefore, At Present:\n"), !, fail.

check_fsr1(9) := !, check_fsr1(8),
check_fsr1(10) := !, check_fsr1(8),
check_fsr1(11) := !, check_fsr1(8),
check_fsr1(12) := !, check_fsr1(8),

check_fsr1(X) := check_fsr2(X).

check_fsr2(8) := dimensions(_, _, LotArea, _, FloorSpace, _),
fsr(_, Ratio2),
(FloorSpace/LotArea) <= Ratio2,!
shiftwindow(1), clearwindow, zone_class(Zone), proposed_use(Use), nl, write(" The use proposed fails to meet the floor space ratio\n"), write(" requirements for the "Zone," Schedule.\n"), !, relaxation(Zone, 5.5, Use), nl, !, fail.
zone_class(Zone), nl,
write("The use proposed fails to meet angle of daylight requirements
"), nl, !, fail.

check_horizontal_sun_angle(9) :- !, check_horizontal_sun_angle(8).
check_horizontal_sun_angle(10) :- !, check_horizontal_sun_angle(8).
check_horizontal_sun_angle(11) :- !, check_horizontal_sun_angle(8).
check_horizontal_sun_angle(12) :- !, check_horizontal_sun_angle(8).
check_horizontal_sun_angle(19) :- !, check_horizontal_sun_angle(8).
check_horizontal_sun_angle(_) :- true.

check_acoustics1(8) :- zone_class(Zone),
write("All Development Permit applications for dwelling uses in the ", Zone, ", require that an acoustical survey of the dwellings be
"), write("undertaken and a report prepared.
"), write("Has such a report been submitted and do you have accurate information, 
"), write("certified by an acoustics specialist, on noise levels for the following
"), write("Bedrooms, Living Rooms, Dining Rooms, Recreation Rooms, Hallways, Terraces,
"), write("Kitchen, Bathrooms, Hallways, Terraces.
"), cursor(R1,C), readchar(V), write(V), nl, nl, check_ans2(V), !.
shiftwindow(1), clearwindow, nl,
write("Zoning approval cannot be made until a proper acoustical survey
"), write("and report has been made. At present:
"), nl, fail.

cursor(R1,C), R2 = R+3, noise_level(Lv1, Lev2, Lev3, Lev4), cursor(R2,C), nl,
write("Do noise levels in any of the Bedrooms exceed ", Lev1, " db (A weighted Leq)? (y/n) ", write(W), write(W), nl, nl, check_ans2(W),
readchar(W), write(W), nl, nl, check_ans2(W),
write("Do noise levels in the Living Room, Dining Room, or Recreation Room
"), write("exceed ", Lev2, " db (A weighted Leq)? (y/n) ", readchar(X), write(X), nl, nl, check_ans2(X),
write("Do noise levels in the Kitchen, Bathroom(s), or Hallways
"), write("exceed ", Lev3, " db (A weighted Leq)? (y/n) ", readchar(Y), write(Y), nl, nl, check_ans2(Y),
write("Do noise levels on the Terrace(s), Patio(s), or Balconies
"), write("exceed ", Lev4, " db (A weighted Leq)? (y/n) ", readchar(Z), write(Z), nl, nl, check_ans2(Z),!
shiftwindow(1), clearwindow, zone_class(Zone), nl,
write("The use proposed fails to meet acoustics requirements
"), write("for the ", Zone, ", Schedule."") nl, !, fail.

check_acoustics1(9) :- !, check_acoustics1(8).
check_acoustics1(10) :- !, check_acoustics1(8).
check_acoustics1(11) :- !, check_acoustics1(8).
check_acoustics1(12) :- !, check_acoustics1(8).
check_acoustics1(19) :- !, check_acoustics1(8).
check_fsr2(9) :- !, check_fsr2(8).
check_fsr2(10) :- !, check_fsr2(8).
check_fsr2(_) :- dimensions(_,_,_,_,LotArea,_,Floorspace,_,_), 

check_fsr3(10) :- !, check_fsr2(10).
check_fsr3(_) :- check_fsr4.
check_site_coverage2(26) :- !, 

check_site_coverage3(10) :- write(" What is the total area of the site covered by: surface parking\n"), write(" accessory buildings, maneuvering aisles, driveways, loading areas\n"), 

check_site_coverage3(10) :- write(" How many dwelling units are proposed?\n"), readreal(X), nl, X < 3, !; 

check_horizontal_sun_angle1(8) :- write(" Is the plane defined by section 4.10.1 of the "),

cursor(R,C), R1=R-2,nl, write(" over a distance of 80 feet from all exterior walls? (y/n) "), write(" (See sections 4.10.2 and 4.10.3 for information on\n what constitutes an obstruction)."), cursor(R,C), readchar(Y), write(Y), cursor(R,C), nl, check_ans1(Y), !;
shiftwindow(1), clearwindow,

check_fsr3(10) :- !, check_fsr2(10).
check_fsr3(_) :- check_fsr4.
check_site_coverage2(26) :- !, shiftwindow(1), clearwindow,nl,

check_site_coverage3(10) :- write(" What is the total area of the site covered by: surface parking\n"), write(" accessory buildings, maneuvering aisles, driveways, loading areas\n"), 

check_site_coverage3(10) :- write(" How many dwelling units are proposed?\n"), readreal(X), nl, X < 3, !; 

check_horizontal_sun_angle1(8) :- write(" Is the plane defined by section 4.10.1 of the "),

cursor(R,C), R1=R-2,nl, write(" over a distance of 80 feet from all exterior walls? (y/n) "), write(" (See sections 4.10.2 and 4.10.3 for information on\n what constitutes an obstruction)."), cursor(R,C), readchar(Y), write(Y), cursor(R,C), nl, check_ans1(Y), !;
shiftwindow(1), clearwindow,
check_acoustics1(_) :- true.

check_dwelling_unit_density1(8) :- !, write(" How many dwelling units are proposed?\t"), readint(X),nl, dimensions(_,_,_,LotArea,_,_), dwelling_unit_density1(Upac), LotArea/ft_per_acre = LotAcreage, round(X/LotAcreage) == Upac.

check_dwelling_unit_density1(9) :- !, check_dwelling_unit_density1(8).

check_dwelling_unit_density1(10) :- write(" How many dwelling units are proposed?\t"), readint(X),nl, dimensions(_,_,_,LotArea,_,_), dwelling_unit_density2(Upac), LotArea/ft_per_acre = LotAcreage, round(X/LotAcreage) > Upac, shiftwindow(1),clearwindow, zone_class(Zone),nl, write(" The use proposed fails to meet dwelling unit density\n"), write(" requirements for the ",Zone," Schedule.\t"),nl,! , fail; write(" Is more than one principal building proposed? (y/n) \t"), readchar(X), write(X),nl,nl,check_ans2(X),!; !,check_number_of_buildings.

check_dwelling_unit_density1(11) :- write(" Are any dwelling units are proposed? (y/n)\t"), readchar(Z), write(Z),nl,nl, check_ans2(Z),!;
write(" How many? \t"), readint(X),nl, dimensions(_,_,_,LotArea,_,_), dwelling_unit_density2(Upac), LotArea/ft_per_acre = LotAcreage, round(X/LotAcreage) > Upac, shiftwindow(1),clearwindow, zone_class(Zone),nl, write(" The use proposed fails to meet dwelling unit density\n"), write(" requirements for the ",Zone," Schedule.\t"),nl,! , fail; write(" Is more than one principal building proposed? (y/n)\t"), readchar(X), write(X),nl,nl,check_ans2(X),!; !,check_number_of_buildings1.
% REGS2.PRO contains the remaining rules for describing the zoning regulations. Note that since these rules
% are called from the 'check_rest_of_regs' clauses in REGS1.PRO they have to be declared as global predicates.
%
% The clause 'get_relaxation' in the body of check_side_yards1 and check_rear_yard1 ensures that the correct
% relaxation clause contained in RELAXNS.PRO is called for certain uses and not others. If the use is not 'B'
% (Infill) or 'I0' (Multiple Dwelling) 'get_relaxation' fails and leads to 'check_use' producing a 'use is not
% permitted' result.
%
project "EUCLID"
include "c:\\programs\\tprolog\\globals.pro"

clauses

check_height1 :- height(Feet, Storeys), write(" Are the buildings adjacent on both sides of the site higher than\n"), write(" "Feet," feet or "Storeys," storeys? (y/n)\t"), readchar(Z), write(Z), n1, n1, check_ans1(Z)!, cursor(R1, C1), R2 = R1, C2 = C1 + 35, write(" How high is the first in ft?\t"), readreal(A), cursor(R2, C2), write(" The second? "), readreal(B), n1, write(" How high is the PRINCIPAL building(s) in ft?\t"), readreal(X), AverageHeight = (A+B)/2, X <= AverageHeight, write(" How high is the PRINCIPAL building(s) in ft?\t"), readreal(X), n1, height(Feet, _), X <= Feet,!
write(" How many storeys does it(they) have?\t"), readreal(Y), n1, height(_, Storeys), Y <= Storeys,!
shiftwindow(1), clearwindow, zone_class(Zone), n1, write(" The use proposed does not meet height requirements for the\n"), write(" "Zone," Schedule. Therefore:)

check_height2 :- height(Feet, _), write(" How high is the PRINCIPAL building(s) in ft?\t"), readreal(X), n1, X <= Feet,!
height(_, Storeys), write(" How many storeys does it(they) have?\t"), readreal(Y), n1, Y <= Storeys,!
shiftwindow(1), clearwindow, zone_class(Zone), n1, write(" The use proposed does not meet height requirements for the\n"), write(" "Zone," Schedule. Therefore:"

check_height3 :- write(" How high is the PRINCIPAL building(s) in ft?\t"), readreal(X), n1, asserta(ans_ht(X), storage), ht(Feet, _), X <= Feet,!
write(" Does the building(s) have a cellar? (y/n)\t"), readchar(Y), write(Y), asserta(ans_cell(Y), storage), n1, n1, check_ans2(Y)!, !, check_height4,
write(" Excluding the cellar, how many storeys does the building(s) have?\t"), readreal(Z), asserta(ans_storeys(Z), storage), n1, ht(Storeys, _), Z <= Storeys,!
shiftwindow(1), clearwindow, zone_class(Zone), n1, write(" The use proposed does not meet height requirements for the\n"), write(" "Zone," Schedule. Therefore:"

check_rear_yard1 :-
  (Left + Right)/2 < 120,
  write(" Since the average depth of the site is less than 120 feet the depth\n"),
  write(" required for the rear yard may be decreased in accordance with section\n"),
  write(" 11.2 of the Zoning By-law\"), nl !.

check_rear_yard2 :-
  write(" Is the site a riparian site? (y/n)\t"), readchar(X), write(X), nl, nl, check_ans1(X),
  check_rear_yard1 !.

check_fsr4 :-
  (FloorSpace/LotArea) <= Ratio !.

check_fsr3 :-
  asserta(sides(X), storage),
  dimensions(Front, Rear, ..., Sideyards, X, Sideyards, Ratio, Feet),
  (Front + Rear)/2 * Ratio = SideWidth, Feet >= SideWidth, X < SideWidth, fail;
  sides(X), X < Sideyards, Feet, X < Feet, fail;
  sides(X), X = Sideyards, Feet, X = Feet, fail.

check_ans2(X) :-
  check_ans1(X), write(" The use proposed does not meet side yard requirements for the\n"), write(" Zone, Schedule\"), nl, !, get_relaxation(X, Use), !, fail.
zone_class(Zone), nl,
write(" The use proposed fails to meet the floor space ratio
"),
write(" requirements for the ", Zone, " Schedule. Therefore\n"), nl, fail.

check_site_coverage1 :-
  dimensions(_, _, LotArea, FloorArea),
  site_coverage(Percents),
  FloorArea/LotArea =< Percents, nl,
  write(" What is the size, in sqft, of the area used for parking?\n"),
  readreal(X),
  parking_coverage(Percents),
  X/LotArea =< Percents, nl,
  shiftwindow(1), clearwinow,
  zone_class(Zone),
  proposed_use(Use), nl,
  write(" The use proposed fails to meet site coverage requirements\n"),
  write(" for the ", Zone, " Schedule.\n"), nl, fail.

check_external_design1 :-
  zone_class(Zone),
  exterior_design(HeightRatio, Width1, Setback, MinFloor, MaxFloor),
  write(" What is the height and width of the principal facade?\n"),
  write(" closest to the front property line.\n"), nl,
  n1, n1, cursor(R1, C1),
  R2 = R1 - 1, C2 = C1 + 30,
  write("Height:\n"),
  readreal(Height),
  cursor(R2, C2),
  write("Width:\n"),
  readreal(Width2),
  Height/Width2 =< HeightRatio,
  Width2 < Width1,
  nl,
  write(" Is the principal access to the building by means of a straight staircase\n"),
  write(" at right angles to the street and leading to a first storey porch or open-\n"),
  write(" sided verandah?\n"),
  readchar(X),
  write(X), nl, n1,
  check_ans1(X),
  write(" Is there only one principal entrance and is it via a single or double doors?\n"),
  readchar(Y),
  write(Y), nl, n1,
  check_ans1(Y),
  write(" Excluding the principal facade, how far back, in feet, is the\n"),
  write(" closest facade from the front yard?\n"),
  readreal(Z), Z =:= Setback,
  nl,
  write(" How high above grade is the floor of the first storey?\n"),
  readreal(Height),
  Height >:= MinFloor, Height <:= MaxFloor,
  nl,
  write(" Are the facades of the building consistent, in terms of style?\n"),
  write(" form of architecture, and exterior finishes, with those residential\n"),
  write(" buildings in the ", Zone, " District listed in the City of Vancouver\n"),
  write(" Heritage Inventory of August, 1986? \n"),
  readchar(U),
  write(U), n1, nl,
  check_ans1(U),
  write(" Are the windows proposed also consistent with those residential?\n"),
  write(" buildings in the ", Zone, " District listed in the City of Vancouver Heritage Inventory\n"),
  write(" of August, 1986? \n"),
  readchar(W),
  write(W), n1, nl, check_ans1(W),
  write(" Are the roof styles proposed either gable or hip-on-gable? \n"),
  readchar(V),
  write(V), n1, nl, check_ans1(V),
  write(" Are shingles the roof material to be used? \n"),
  readchar(T),
  write(T), n1, nl, check_ans1(T),
  shiftwindow(1), clearwinow,
  zone_class(Zone), nl,
  write(" The project proposed fails to meet external design requirements.\n"),
  write(" The Director of Planning may permit a development which varies\n")
write("from the external design characteristics of the ", Zone, " Schedule\n"),
write("provided that:\n"),
write("- It is consistent with residential buildings in the\n"),
write("Zone," District listed in the City of Vancouver\n"),
write("- Heritage Inventory of August, 1986\n"),
write("- A consistent architectural style and form is achieved\n"),
write("for every building on the site; and\n"),
write("- It reflects the character of the streetscape and is\n"),
write("compatible with the design of buildings on adjoining sites.\n"),
write("- See more? (y/n)\t"),readchar(A), write(A), check_ans1(A), clearwindow,nl,
write("The Director of Planning may also require the retention\n"),
write("of any one or more of the following architectural\nor ornamental features;\n"),
write("- window, door and roof decoration;\n"),
write("- bay windows;\n"),
write("- tower or turret features;\n"),
write("- hand rails, balusters;\n"),
write("- wood tracery or bargeboard.\n"),
write("SEE MORE? (y/n)\t"),readchar(B), write(B), check_ans1(B), clearwindow,!, fail.

check_external_designA := shiftwindow(1), clearwindow,nl,
write("The use meets external design requirements, but:\n"),
write("The Director of Planning may still require the retention\"),
write("of any one or more of the following architectural or ornamental features;\n"),
write("- window, door and roof decoration;\n"),
write("- bay windows;\n"),
write("- tower or turret features;\n"),
write("- hand rails, balusters;\n"),
write("- wood tracery or bargeboard.\n"),
write("DO YOU WANT TO CARRY ON WITH THE EVALUATION? (y/n)?\t"),
readchar(X), write(X), check_ans2(X), clearwindow, fail.

check_number_of_buildings1 := shiftwindow(1), clearwindow,nl, zone_class(Zone),
write("The Director of Planning may permit more than one principal\n"),
write("building on site provided the following are first considered:\n"),
write("applicable policies and guidelines;\n"),
write("- The intent of the ", Zone, " schedule and well as any\"),
write("- The height, bulk, location, and overall design of the\n"),
write("proposal and their affect on the site, surrounding\"),
write("- buildings, streets, and existing views;\n"),
write("- The amount of open space and the effect of the proposal\"),
write("- on the general amenities of the area, and;\n"),
write("- The preservation of architectural character and the\"),
write("general amenities desired for the area.\"),nl,nl,
write("DO YOU WANT TO CARRY ON WITH THE EVALUATION? (y/n)?\t"),
readchar(X), write(X),nl,check_ans2(X),!, clearwindow, fail.

check_number_of_buildings1 := shiftwindow(2), zone_class(Zone),
write("Do any of the buildings proposed require a relaxation of the\"),
write("rear yard requirement as specified in section 4.6 of the\"),
write("Zone," schedule? (y/n)\t"),readchar(X), write(X),nl,nl, check_ans2(X),!;
write(" Are the buildings which intrude into the rear yard higher than 25 feet\n"), write(" or 1 1/2 storeys? (y/n)\t"), readchar(Y), write(Y),nl,nl, check_ans2(Y),!;
shiftwindow(1),clearwindow,fail.

get_relaxation(5.4,8) :- !.
get_relaxation(5.3,10) :- !.
% RELAXNS.PRO contains a list of specific zoning relaxations which may be allowed by the Director of Planning
% for a given use/site combination. If the use/site combo fails a zoning regulation where a relaxation may be
% allowed and the use and zone in question match the corresponding relaxation, that relaxation is included in
% output that EUCLID provides to user upon completion of the zoning evaluation.
%
project "EUCLID"
include "c:\\programs\ptrlog\globals.pro"

clauses

relaxation("RT-6",5,1,11) :- zone_class(Zone),nl,
write(" If the lot in question was on record in the Land Title Office for
"),write(" Vancouver prior to August 10, 1976 the Director of Planning may
"),write(" relax the minimum site area provisions as defined in section 4.1
"),write(" of the "Zone," Schedule.\n").
relaxation("RT-6",5,1,12) :- relaxation("RT-6",5,1,11).
relaxation("RT-6",5,2,1) :- nl,write(" The Director of Planning may relax the area and site coverage
"),write(" limitations for accessory buildings and sections 4.7 and 4.9 of
"),write(" the Parking By-law where he is satisfied that adequate off-street
"),write(" parking cannot otherwise be accommodated, provided that in
"),write(" developments where a carport or garage is planned he also has
"),write(" regard to the effect on neighbouring sites of building height,\n"),write(" shadow, open space and landscaping.").
relaxation("RT-6",5,3,10) :- zone_class(Zone),nl,
write(" The Director of Planning may relax the side yard and/or rear yard
"),write(" provisions of sections 4.5 and 4.6 of the "Zone," Schedule in the
"),write(" the case of multiple dwellings, provided that he first considers
"),write(" all applicable policies and guidelines adopted by Council.").
relaxation("RT-6",5,4,8) :- zone_class(Zone),nl,
write(" SEE MORE? (y/n):\t"),readchar(X),write(X),check_ansl(X),nl,clearwindow,nl,
write(" The Director of Planning may relax the side yard and/or rear yard\n"),write(" provisions of sections 4.5 and 4.6 of the "Zone," Schedule in the\n"),write(" case of infill and the placement of more than one principal\n"),write(" building on the site, provided that he first considers:\n"),write(" All applicable policies and guidelines adopted by Council\n"),write(" The height, bulk, location, and overall design of the\n"),write(" buildings and their effect on the site, surrounding buildings,\n"),write(" streets and existing views;\n"),write(" The amount of open space and the effect of overall design on\n"),write(" the general amenity of the area and\n"),write(" The preservation of the character and general amenity desired\n"),
write(" Therefore, At Present s")


write(" relaxation! "RT-5",5.1,12) := relaxation("RT-6",5.1,12).

write(" relax at ion! "RT-5",5.2,1) := relax at ion! "RT-6",5.1,12)

write(" The Development Permit Board or The Director of Planning, as the\n")
write(" case may be, may relax the area and site coverage limitations for\n")
write(" accessory buildings and sections 4.7 and 4.9 of the Parking By-law\n")
write(" where he is satisfied that adequate off-street parking cannot\n")
write(" otherwise be accommodated, provided that in developments where a\n")
write(" carport or garage is planned he also has regard to the effect on\n")
write(" neighbouring sites of building height, shadow, open space and\n")
write(" landscaping.").

write(" The Director of Planning may relax the side yard and/or rear yard\n")
write(" provisions of sections 4.5 and 4.6 of the "Zone," Schedule in the\n")
write(" case of mu 11 1 i pie 11 i n g s , p 1\n")
write(" He first'consi 1 1 e\n")
write(" adopt\n")
write(" No fewer that 50 percent of the dwelling units within any\n")
write(" a building designed solely for senior citizen housing or other\n")
write(" similar use.").

write(" The Development Permit Board or the Director of Planning, as the\n")
write(" case may be, may relax the side yard and/or rear yard provisions\n")
write(" of sections 4.5 and 4.6 of the "Zone," Schedule in the case of\n")
write(" infill and the placement of more than one principal building on\n")
write(" the site, provided that he first considers all applicable policies and\n")
write(" guidelines adopted by Council. And that he first has regard to\n")
write(" applicable guidelines or policies which City Council may from time\n")
write(" to time determine.").

write(" The Director of Planning may relax the floor space provisions of\n")
write(" section 4.7.1 of the "Zone," Schedule in the case of one family\n")
write(" and two family dwellings where he is satisfied that their design\n")
write(" reflects the character of the streetscape and is compatible with\n")
write(" the design and siting of buildings on adjoining sites, provided\n")
write(" that:\n")
write(" - He first considers all applicable policies and guidelines\n")
writet" adopted by Council, and the submission of any advisory group, \\
write(" property owner or tenant, and)
write(" - The floor space ratio does not exceed 0.75.
write(" SEE MORE? (y/n):"
readchar(X)
write(X)
check_ans1(X)
nl
write(" Therefor,

Therefore, At Present:

relaxation("RT-4",5.1,11) :- relaxation("RT-5",5.1,11).
relaxation("RT-4",5.1,12) :- relaxation("RT-6",5.1,12).
relaxation("RT-4",5.2,1) :- relaxation("RT-5",5.2,1).
relaxation("RT-4",5.2,26) :- relaxation("RT-5",5.2,1).
relaxation("RT-4",5.4,8) :- zone_class(Zzone),nl,
write(" In order to maintain the character of the neighbourhood including \\
write(" where possible the retention of existing buildings, the Director \\
write(" of Planning or the Development Permit Board, as the case may be, \\
write(" may relax the side yard and/or rear yard provisions of sections \\
write(" 4.5 and 4.6 of the "Zone" Schedule in the case of infill,
write(" provided that he first considers all applicable policies and \\
write(" guidelines adopted by Council.").
relaxation("RT-2",5.1,12) :- nl,
write(" The Director of Planning may relax the minimum site area \\
write(" requirements of section 4.1 with respect to two family dwellings \\
write(" provided the lot was on record in the Land Title Office for \\
write(" Vancouver prior to September 7th, 1965.").
The contents of GLOBALS.PRO define domains, predicates, and databases which are to be shared by all the modules of the EUCLID program. That is they are global rather than local (to that module) in scope.

These global domains have been specified to make the program easier to read and to differentiate between the different pieces of information supplied by the user and contained within the databases.

'useslist = integer*' means that the domain useslist is made up of a list of integers. This list defines which uses are allowed in each zone.

'upac' means 'Units Per ACRE' and defines the density allowed for dwelling units as a real number.

global domains
useslist = integer*
zone, windows, roof = string
sqft, feet, storeys, ratio, percent, upac, height_width_ratio, width, min_floor, max_floor = real
height, max_height, setback_ft, area_percent, area_sqft, decibels, ans = real
front, left, rear, right, lotarea, floorarea, floorspace, frontdepth, reardepth = real
number, use, acc_roof = integer

Below are listed the predicates which are shared by all modules of the EUCLID program.

Note that predicates which take one or more arguments (the terms between brackets) also have to have their input/output parameters - e.g. (i,o) (o,i) - specified.

The keyword 'nondeterm' before a predicate indicates a predicate which has more than one possible solution or which may lead to backtracking. Since global predicates are by default deterministic (i.e. they lead to one and only one solution) a nondeterministic global predicate produces an error message. Placing 'nondeterm' in front of the predicate prevents the message and forces Turbo Prolog to handle it correctly.

The following predicates form the analytical core of the EUCLID system. The principal global predicates are:

check_use(zone, use)
check_outright_app(zone, integer)
check_conditional_app(zone, integer)
check_regs(zone, integer)
check_rest_of_regs(zone)

/*****************************/
GLOBAL DECLARATIONS */
/*****************************/
%
% The contents of GLOBALS.PRO define domains, predicates, and databases which are to be shared by all the
% modules of the EUCLID program. That is they are global rather than local (to that module) in scope.
%
% These global domains have been specified to make the program easier to read and to differentiate between the
% different pieces of information supplied by the user and contained within the databases.
%
% 'useslist = integer*' - means that the domain useslist is made up of a list of integers. This list defines
% which uses are allowed in each zone.
%
% 'upac' - means 'Units Per ACRE' and defines the density allowed for dwelling units as a real number.

global domains
useslist = integer*
zone, windows, roof = string
sqft, feet, storeys, ratio, percent, upac, height_width_ratio, width, min_floor, max_floor = real
height, max_height, setback_ft, area_percent, area_sqft, decibels, ans = real
front, left, rear, right, lotarea, floorarea, floorspace, frontdepth, reardepth = real
number, use, acc_roof = integer

/*****************************/
GLOBAL PREDICATES */
/*****************************/
%
% Below are listed the predicates which are shared by all modules of the EUCLID program.
%
% Note that predicates which take one or more arguments (the terms between brackets) also have to have their
% input/output parameters - e.g. (i,o) (o,i) - specified.
%
% The keyword 'nondeterm' before a predicate indicates a predicate which has more than one possible solution or
% which may lead to backtracking. Since global predicates are by default deterministic (i.e. they lead to one
% and only one solution) a nondeterministic global predicate produces an error message. Placing 'nondeterm' in
% front of the predicate prevents the message and forces Turbo Prolog to handle it correctly.
%
% The following predicates form the analytical core of the EUCLID system. The principal global predicates are:
%
% check_use(zone,use)
% check_outright_app(zone,integer)
% check_conditional_app(zone,integer)
% check_regs(zone,integer)
% check_rest_of_regs(zone)
global predicates

nondetm check_use(zone,use) - (i,i)
check_ans1(char) - (i)
check_ans2(char) - (i)
check_ans3(char) - (i)
check_outright_app(zone, integer) - (i,i) (0,0) (i,0) (0,i)
check_conditional_app(zone, integer) - (i,i) (0,0) (i,0) (0,i)
nondetm check_regs(zone, integer) - (i,i) (0,0) (i,0) (0,i)
nondetm check_rest_of_regs(zone) - (i)
nondetm relaxation(zone,real, integer) - (i,i,i) (0,0,0) (i,0,0) (0,i,0)
get_relaxation(real, integer) - (i,i) (0,0) (i,0) (0,i)

% The predicates below are used to describe the regulations which apply for each zone regardless
% of the use proposed. They are accessed via 'check_rest_of_regs(zone)' and are used to test
% whether or not the site/use combination proposed is permitted in the zone in question.
%
check_height1
check_height2
check_height3
check_height4
check_front_yard1
check_front_yard2
check_front_yard3
check_side_yards1
check_rear_yard1
check_rear_yard2
check_fsr4
check_site_coverage1
check_external_design1
check_external_designA
check_number_of_buildings1

quantity

global constants

ft_per_acre = 43560

constantes

% The constants section allows special terms to be defined for use by all the modules of EUCLID. In this
% case 'ft_per_acre' is defined as having the value 43560 and is a conversion factor for calculating
% dwelling unit densities.
%
global database - continue

get加快建设 productive and loaded into continue.

The continue database stores ongoing specifications in the system and captures which information supplied by the user is loaded. The continue database is stored in continue files which are accessed by the database which are accessed by

The continue database stores ongoing specifications in the system and captures which information supplied by the user is loaded. The continue database is stored in continue files which are accessed by the database which are accessed by the

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.

dimensional front, rear, right, left, rear, floor area, floor space, front depth, rear depth.
front_yd (feet)
side_yards(ratio, feet)
side_yds (feet)
rear_yard (feet)
fsr (ratio, ratio)
site_coverage (percent)
site_cov (percent, percent)
parking_coverage (percent)
dwelling_unit_density1 (upac)
dwelling_unit_density2 (upac)
number_of_buildings (number)
noise_level (decibels, decibels, decibels)

exterior_design (height_width_ratio, width, setback_ft, min_floor, max_floor)

accessory_buildings (height, max_height, setback_ft, area_percent, area_sqft, ratio)