NOTICE

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

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

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

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

AVIS

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

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

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

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

An Experiment in Automating Case-Based Knowledge Acquisition

Carol De Koven

A Thesis
in
The Department
of
Computer Science

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

December 1991

ISBN 0-315-73695 X

Canada

Abstract

An Experiment in Automating Case-Based Knowledge Acquisition

Carol De Koven

The knowledge base is an essential part of an expert system. Knowledge acquisition, the process used to collect the knowledge from domain experts, is recognized as a vital link in the development of expert systems. At the same time it is considered to be the most difficult aspect of the undertaking. In this thesis we survey manual and automated knowledge acquisition techniques. In particular, we emphasize the advantages of automatically generating from a set of cases a rule base which is useful for prototyping an expert system. This thesis is based on two assumptions. The first is that domain experts' time is valuable and limited, hence their involvement in the preliminary stages of knowledge acquisition should be limited, reserving their contribution for the later refinement stage. The second assumption is that in many domains there are collections of case data, often in computer readable form, which can be used for knowledge acquisition purposes.

To illustrate the feasibility of our concept, we have undertaken an experiment to automatically acquire knowledge from stored cases in the domain of Blackbox, a computer game. The work involved the design, implementation, and testing of a program used to extract knowledge from selected Blackbox cases. The acquired knowledge was placed in the rule base of a prototype expert system and used to play the game. The test results obtained from this prototype affirm that automated knowledge acquisition from cases is a viable methodology. As well, the tests demonstrate the benefits of applying successive iterations when performing knowledge acquisition and of acquiring knowledge from multiple domain experts.

ACKNOWLEDGEMENTS

Before thanking the individuals who helped me in my work, I must acknowledge the debt I owe to Concordia University. The University's policy of encouraging part-time students afforded me the possibility of pursuing my undergraduate studies, which ultimately led to my graduate work.

I could not have accomplished this work without the support of my supervisor, Dr. T. Radhakrishnan. Dr. Radhakrishnan was always there with helpful advice when I was stumped and a gentle push when I was discouraged. His guidance throughout the course of my graduate studies and the research and writing of this thesis was indispensable. Thank you, Dr. R.

To Cliff Grossner, who advised me to embark on my graduate studies and guided me through my early research into expert systems and knowledge acquisition, as well as always being available for consultation, I owe a special thanks.

Kristina Pitula's help and her incisive input regarding the game of Blackbox were useful in the analysis and design of my knowledge acquisition program. John Lyons contributed to my work by writing the expert system shell on which I could test my work. John also functioned as my resident human expert along with Le Hoc Duong. I thank them all, as well as Christine Nadal, whose assistance in drawing figures and preparing the thesis for printing was essential when the final crunch was on.

Finally, I wish to acknowledge my family's contribution. To my children, Robin, Alan, and Kathryn, whose unfailing pride in my academic endeavours was frequently my motivation, thank you for that and for all the help. And a very special thanks to my husband, Max, for his steadfast support and encouragement through the long years of the process.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 A Framework for the Experiment

Projects involving the development of expert systems have been undertaken in a variety of domains. Often one of the first steps in such ventures is to build a prototype for the purpose of exploring the feasibility and suitability of the proposed expert system. The expertise that prototype expert systems use resides in their knowledge bases, which are generated through the acquisition of domain knowledge from domain experts.

Knowledge acquisition, which is integral to the development of expert systems, often involves direct continual interaction between the knowledge engineer and the domain expert. Numerous attempts have been made to reduce the work of the knowledge engineer by using automated knowledge acquisition programs to interview domain experts and thus create a prototype expert system. This thesis is concerned with using a case-based approach to automated knowledge acquisition in order to minimize the time required of the domain expert.

Two premises form the basis of the thesis. The first is that the domain expert's time is at least as important as that of the knowledge engineer. In many domains there are few experts and the time of those few is extremely valuable. The shortage of readily available expertise to solve problems has contributed to the growing demand for expert systems. However, the many demands on their time leave domain experts

little time to devote to knowledge acquisition. The experts' lack of available time to share their knowledge can make the building of expert systems a prolonged and arduous task. Thus the same factor that has prompted the need for expert systems also impedes their development. This suggests a need to find ways to reduce as much as possible, but not eliminate totally, the expert's involvement in expert system development.

This leads to the second premise for this thesis, namely that in many domains data pertaining to past activities and decisions has been stored and documented and that such data can be used as a source for automated case-based knowledge acquisition. The accumulation of data is a common practice in many fields. In the course of daily operations a considerable amount of information leading to and resulting from actual domain activities is gathered and stored. Today such data is frequently well-documented, categorized, and readily accessible in computer data banks.

1.2 Proposed Work

The second premise for the thesis provides a means to a solution for the problem presented by the first. If knowledge engineers require additional means of knowledge acquisition to overcome the difficulties posed by the experts' lack of time perhaps the accumulated data in some domains could provide a needed source.

It is proposed here-in that by analyzing stored domain data, one can attempt to trace the domain expert's reasoning in arriving at decisions. Since such data has been amassed from actual examples within a given domain, it can serve as a primary source of knowledge from which rules can be extracted and a prototype rule-based expert system can be built. This method of knowledge acquisition is meant to reduce the domain expert's involvement in supplying the knowledge engineer or an automated interactive knowledge acquisition system with the elementary domain knowledge required to build a prototype expert system.

Using the method proposed in the thesis, most of the early work involved in

the development of the prototype would be borne by the knowledge engineer. This would include becoming acquainted with the stored data and writing a program to organize the data into rules for the prototype expert system. It is not expected that the knowledge base will be exhaustive, nor that the human expert's involvement can be eliminated entirely. Rather, the prototype will only be expected solve a limited set of cases. At this point the expert system could create a file listing all the problems that it could not handle and these problems could then be presented to the domain expert for correction or clarification. Thus the human expert's time could be used more effectively in the later stages of the building of the expert system, once the prototype system has been built and tested.

In this thesis I proposed the knowledge acquisition process that is depicted in Figure 1.1 and carried it out using cases of the Blackbox game as the source of knowledge acquisition. The results of the experiment are presented in the thesis.

1.3 Thesis Outline

Chapter 2 of the thesis will describe the various means available for knowledge acquisition, including traditional manual acquisition methods, automated interactive systems, and the acquisition of knowledge from cases. An explanation of the game of Blackbox will be given in Chapter 3, as well as a brief description of the distributed version of the game and previous work done in the domain. Chapter 4 will provide a detailed description of the work that was involved in the case-based knowledge acquisition experiment. The expert system and the generated rule base will be presented in Chapter 5. In Chapter 6, the tests that were carried out using the generated rule bases will be explained and the results presented and discussed. The final chapter in the thesis will present the conclusions resulting from this experiment, as well as a small section proposing possible future work in this research area.

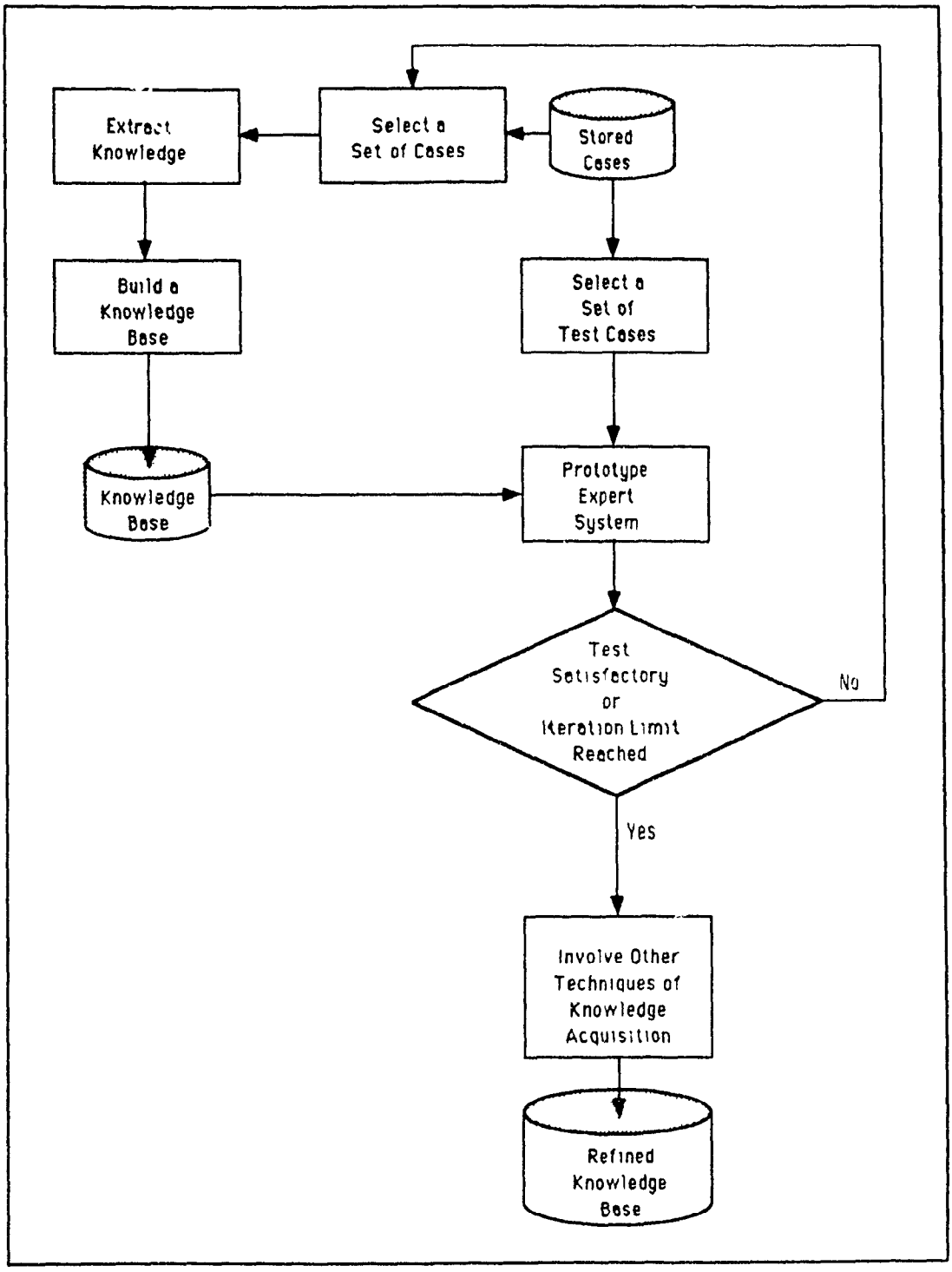

Figure 1.1: Knowledge Acquisition Process

Chapter 2

KNOWLEDGE ACQUISITION - PAST AND PRESENT

2.1 Knowledge Acquisition Function

2.1.1 The Role of Knowledge Base in Expert Systems

Early research in Artificial Intelligence revealed that general problem-solving techniques and heuristic methods were deficient when faced with the complexities entailed in solving ambiguously defined problems with imprecise information. The factor that was found to be missing in the problem-solving programs was knowledge, a resource that people use in various aspects of problem handling. Human problem solvers use knowledge to help define the problem, to propose useful solution methods, to ascertain whether available facts are valid, and to judge if a feasible solution has been found. The realization that expertise in a task domain depends upon significant knowledge about that domain was the start of a major shift in the direction of AI research [5]. From attempting to isolate a few powerful problem-solving techniques, the focus shifted to knowledge, how to obtain it and how to represent it in ways that would facilitate its manipulation. Writing a program to solve problems in a given domain required capturing and encapsulating the expertise that a human expert would use in their solution. This new emphasis on knowledge as the key to achieving expertise in solving problems prompted the use of the term "knowledge-based systems" as a synonym for "expert systems" [7].

The knowledge built into an expert system's knowledge base is meant to imbue the expert system with the ability to solve a given set of problems within the specified domain. The knowledge base is also important for its relationship to the other components of the expert system. The type of knowledge involved in solving the problem, its representation, and the reasoning method appropriate to manipulate it, help determine the type of inference engine that should be used or in which expert system shell the system should be built. The knowledge built into the expert system also influences the design of the expert system's explanation facility.

Each expert system's knowledge base distinguishes it from other expert systems. All knowledge is not relevant to all domains, and within a given domain, different bodies of knowledge are required to address different issues. This great diversity of knowledge across the wide range of human endeavours makes the building of expert system knowledge bases a complex and specialized task. Additional complications are presented by the fact that knowledge is stored and represented in a variety of ways.

2.1.2 Types of Knowledge

Within a specific field, human expertise incorporates general knowledge about the field, an in-depth grasp of inherent problems, and the ability to solve most of them. To effectively acquire knowledge and build it into an expert system it is necessary to first gain an understanding about the different aspects of knowledge as it can be applied to domain expertise and to expert systems. Knowledge may be categorized according to its functionality and according to its provenance. Two broad functional categories of knowledge have been identified: background or shallow knowledge and deep or expert knowledge [15]. Background knowledge can be partitioned into the knowledge that can be used to become familiar with the domain and the knowledge of practitioners in the domain that may be used to implement a preliminary knowledge base. The background knowledge at the practitioner level can be further subdivided into descriptive knowledge and procedural knowledge. Expert knowledge, which is mostly procedural, is used to refine the knowledge base. [15]

Background knowledge for domain familiarization comprises

1. the overall goals of the domain;
2. the vernacular of the domain;
3. the domain's accessible sources of information; and
4. the characteristics of the system's intended users.[15, 23]

Included in the practitioner level knowledge are

1. objects within the domain: their names, their possible attributes, their potential relationships and interactions;
2. data relevant to the solution of problems, and the sources of such data;
3. actions and events associated with the problem solution;
4. restrictions and exceptions that need to be considered;
5. techniques used to interpret the problem and solve it; and
6. how to evaluate effectiveness and gauge success.

Some practitioner level knowledge is known as "Descriptive Knowledge". This encompasses the "object-oriented" items – facts related to domain-specific objects, people, incidents, actions, and concepts. It describes objects, their role, and the relationships and traits they may have. With little or no experience in the field needed to acquire it, most descriptive knowledge is not classified as expert knowledge.

The non-descriptive component of practitioner level knowledge is known as "Procedural Knowledge" and is associated with rules. This category includes the strategies and methods needed to change the conditions of objects. It basically supplies "textbook" answers to domain problems and is the repository of the "if-then-else" rules. Some procedural knowledge is expert knowledge. However, much of it is considered "public knowledge", since it can be acquired without involving an expert.

By contrast with background knowledge, which is essentially quantitative, expert knowledge is basically heuristic and qualitative, resulting from extensive experience in the domain. It includes

1. the correlation among various types of data and actions;
2. understanding the relative significance and pertinence of data and its sources;
3. deductions from limited or flawed data;
4. decisions concerning the relevance and sequence of assorted activities;
5. shortcuts in solution process;
6. heuristics and effective rules of thumb;
7. ability to recognize patterns that lead to problem definition and solution;
8. possible trade-offs, and their results.

Aside from its functional characteristics, the knowledge that underlies expertise can also be divided according to its origin into public and private knowledge [15]. Familiarization-specific background knowledge falls under the public knowledge classification since it is readily available in a published format. Much of the practitioner-level knowledge is also public and can be obtained by consulting accessible sources. In addition to such publicly accessible knowledge, human experts use private knowledge that has not yet been published, or widely disseminated within the domain. Private knowledge also includes the portion of practitioner level knowledge that is not readily available. An expert system must incorporate both the public and private knowledge that is necessary to effectively solve problems in the domain.

2.1.3 Sources of Difficulty for Knowledge Acquisition

Knowledge acquisition is recognized as the most difficult as well as the time-critical task in expert system development [5, 25]. Most knowledge engineers begin the task of building an expert system with an incomplete understanding of the problem domain, a deficiency that must be overcome by researching the field in question. The acquisition of public knowledge is relatively routine and in most domains can be accomplished with little or no difficulty, time being the main variable factor. However, public knowledge is usually only shallow surface knowledge and while it may yield a functional knowledge base for an expert system (indeed, task-oriented superficial knowledge is routinely used by many expert systems [34]), such a system

will not exhibit the type of expertise that can be achieved with expert knowledge. In order to broaden the information acquired it is therefore necessary to attempt to elicit the knowledge of experts in the domain.

The difficulty of the knowledge acquisition task often begins at this point [5]. The domain knowledge possessed by experts was learned over a number of years. A knowledge engineer cannot realistically expect that this knowledge can be elicited from the expert over a short period of time. This can be partially attributed to the fact that frequently experts' knowledge and abilities have become intuitive. Experts often make decisions without being cognizant of the facts or knowledge they use or of the techniques they apply. On the surface, they know and understand the factors that they employ in their work, but often, their decisions may be influenced by other stimuli which they do not acknowledge at a conscious level. Since the experts themselves do not consciously recognize all the varied elements that contribute to their work, it is very difficult for a knowledge engineer to penetrate the process and extract that information.

An additional hindrance to the knowledge acquisition task is the difficulty that many experts have in verbalizing how they approach their work [23]. This may be due partly to their inability to bring to the surface some of their intuitive and unconscious knowledge. It may also be due to the fact that many experts operate at a higher level than laymen or novices in their field and, unless they are accustomed to teaching or training beginners, they are unable to discuss their work at a lower level than their own.

Experts in most fields are in short supply and are often too busy doing their work to help in the building of an expert system. When experts, for lack of time or other unspecified reasons, do not cooperate fully with the knowledge engineer, the difficult job of acquiring the necessary knowledge becomes even more complex and exacting. Thus, in order to overcome the difficulties mentioned and successfully transfer the knowledge of the experts into the system, knowledge engineers must expend a considerable amount of time and effort in the process of knowledge acquisition.

2.2 Manual Knowledge Acquisition

2.2.1 Historical Perspective

Knowledge acquisition has usually been done manually, involving the knowledge engineer intimately with the domain, its practitioners, and its expert(s) throughout the process. Past experience of knowledge acquisition has shown that the process involves several stages and combines a variety of methods and techniques to acquire the necessary domain knowledge and transfer it to the expert system knowledge base. The steps include

1. obtaining background domain knowledge
2. eliciting knowledge from the expert
3. recording the knowledge
4. generating a prototype with the knowledge
5. testing the prototype
6. repeating the process from step (2).

The iteration of steps (2) through (5) is done until the prototype functions satisfactorily. The knowledge acquisition task is ended when the expert system is deemed to be fully operational, unless additional knowledge is required in an evolving problem domain. Many systems incorporate a facility for the system's users to update its knowledge base, thus relieving the knowledge engineer of responsibility for further knowledge acquisition. Within the knowledge acquisition process, steps (1) and (2) involve an assortment of methods and techniques, which may be used in various combinations, depending on the domain and its complexity.

2.2.2 Preparatory Methods

The first preparatory stage entails familiarizing the knowledge engineer with the problem domain. Initially it is necessary for the knowledge engineer to frame the problem clearly and concisely. In addition to declaring the raison d'être of the proposed expert system, the problem definition is useful in clarifying significant

aspects of the system. It helps to ascertain whether the problem is appropriate for an expert system application, and if so, it helps set limits on the knowledge that would be required by an expert system to produce an acceptable solution. As part of the familiarization process the knowledge engineer learns about the system's potential users - their abilities, the functions they perform, and their probable expectations from the system. The user data, along with the problem definition, is used to decide what type of system will meet the needs identified. [15]

While the knowledge engineer is becoming familiar with the problem domain, he or she is also acquiring background knowledge, whether by design or as a by-product of the familiarization processes. Background information can be obtained from a variety of sources including manuals, books, journals, and classes based on the domain, as well as from practitioners in the domain. Some of the techniques used in eliciting knowledge from the domain expert can be used to obtain information from domain practitioners. These methods include observation sessions, unstructured interviews, and questionnaires [14, 25].

Although not all of the background information will be used in the actual implementation of the system, it is a key component of the knowledge engineering task. Its function is to provide the knowledge engineer with sufficient understanding of the domain to be able to discuss its intricacies with the expert. Background knowledge is also essential to gain the expert's trust that the knowledge engineer understands the knowledge well enough to express it faithfully in the system [33]. This will make the eventual task of knowledge elicitation from the expert more productive and less stressful.

The comprehension that the knowledge engineer has gained from the background knowledge also helps him or her to make informed design decisions about the planned system [8, 30]. Some questions answered by these preliminary decisions are

- on what type of machine should the system be implemented?
- which language is best suited to the needs of the system?
- should an inference engine be built or can a commercially available tool be used?

- What type of documentation should be kept during the knowledge acquisition process?
- which knowledge representation formalism is most suited to the system and should more than one type be used?

Having made these decisions the knowledge engineer has a clear idea of the direction that knowledge acquisition for the expert system will take. The background knowledge that has been gained can then be organized into a "first-pass" data base and used as a basis upon which the knowledge to be elicited from the expert will be built [14].

2.2.3 Knowledge Acquisition From the Expert

When the preparation phase is over the knowledge engineer can begin the process of extracting the knowledge of the domain expert. Expert systems research has identified a number of effective techniques for this task. The two that are most commonly used are observation and interviews. Previous experience has yielded certain guidelines to the use of these and other methods of manual knowledge acquisition.

Observation

As the name implies, observation is a technique whereby the knowledge engineer observes the expert performing some typical tasks. This technique has also been called "the method of familiar tasks", as well as "Behavioural Observation" [14]. To yield the most information, observation should take place without interrupting the expert and breaking his or her concentration. It is preferable that the tasks under scrutiny be selected by the expert rather than the knowledge engineer. These constraints on the observation should ensure that the expert is at ease being involved in his or her usual routine. The drawback of observation is that it can be time-consuming and may not by itself give any great insight into the reasoning process used by the expert. But information gained from observation sessions helps the knowledge engineer augment his or her familiarity with the domain. The knowledge engineer also gains an appreciation of the inputs experts require for solutions as well

as the data produced as a result of their tasks. This information can be combined with the background knowledge acquired in the familiarization phase to prepare a preliminary data base that can then be refined using other methods.

Interviews

Two types of interview techniques are used in knowledge acquisition: unstructured and structured. Unstructured interviews can take place while the expert is describing or performing a task in the domain. In this type of interview, the knowledge engineer spontaneously asks the expert questions, particularly when the expert appears to be drawing an inference that is not obvious. The knowledge engineer's questions may be partly based on the knowledge that was included in a preliminary data base [14]. The unstructured interview method has been one of the most frequently used knowledge acquisition tools. Some rule bases have been constructed solely on the basis of knowledge gained in this way [34].

By contrast with the unrehearsed nature of the unstructured interview, the structured interview is a rather formal approach and requires some preliminary work on the part of the knowledge engineer. The interview is conducted using previously prepared questions, which may have been derived from the research done during the familiarization phase or from previous knowledge acquisition sessions. Structured interviews can produce considerable information, but they are usually very time-consuming [14].

Other Techniques

Self reporting or verbal reporting is a knowledge acquisition method that consists of asking the expert to "think out loud" while performing a typical task [24, 33]. Self reporting sessions should be recorded on audiotape or videotape and reviewed after the session.

There are some methods that may be used to gain deeper knowledge and uncover the expert's heuristic reasoning. In one such approach the expert is asked to solve

tasks in which the information has been deliberately limited by the knowledge engineer. Another technique involves changing or constraining the expert's reasoning strategies. The theory is that by either limiting the time available for solving the problem or asking the expert to address only a small portion of the problem, more and different examples of the expert's thinking patterns may emerge. Although at first the expert may not be at ease with the restrictions imposed, both these methods have extensive potential for the acquisition of deep knowledge. [14]

Observing the expert solving a complicated and unusual problem can be very instructive about the more intricate aspects of the decision making process. The rarity of such problems makes it difficult to predict when such observations should be done. To overcome this, the domain expert may be requested to audiotape any difficult cases he or she may encounter, even if the knowledge engineer is not at hand.[14]

2.2.4 Problems in Manual Knowledge Acquisition

Manual knowledge acquisition is usually very time-consuming. Much has been said and written about the difficulties involved with the elicitation of deeply seated knowledge from domain experts [14, 15, 23]. These difficulties sometimes arise as a result of biases that occur when the methods used for the collection of such knowledge are not applied correctly.

A structured interview can yield very valuable information. However, if a structured interview is not conducted carefully, the knowledge engineer risks creating biases with the wording of questions and hence obtaining inaccurate information. To detect such biases, the knowledge engineer should ask the same question in different ways. If the answers to the various questions are not similar, the knowledge engineer must alert the expert to the inconsistency and try to correct the problem.

Unstructured interviews or interruption must also be handled carefully by the knowledge engineer in order to avoid influencing the expert's thinking. Such an impact would be most likely if the work is being done on a rapidly changing problem. In fact, in such a case, the expert is apt to completely ignore the knowledge engineer

and proceed with the work at hand. If this were to happen, the knowledge engineer could continue the learning process by switching to uninterrupted observation and noting questions to be asked soon after the job has been completed.

Self reporting has the disadvantage that many people find it difficult to verbalize their thoughts, particularly while performing important tasks. A further drawback is that this method can disrupt the thought processes of the expert, and as a consequence, impact on his or her effectiveness. Therefore, self reporting sessions are preserved on videotape or audiotape in order to keep a record of the procedure and the expert is not constrained to talk if it is not convenient. To keep the decision process fresh in the expert's mind, it is recommended that the analysis of the recording should be done as soon as possible after the session is over, thus filling in any gaps there may be in the information revealed.

To ensure the accuracy of the knowledge base the knowledge engineer should be aware at all times that there is a likelihood that biases will influence some of the information elicited from the expert. The sources and causes of such biases and errors should be monitored carefully in order to combat their effects on the knowledge base. [33]

2.2.5 Application of Knowledge Acquisition Methods

The techniques that have been described may be combined in various ways depending upon the domain. Some of them can be applied at different points in the knowledge acquisition process and may produce different types of information depending on when they are used. Others serve as more specialized tools and perform best when limited to specific times in the process.

In the preparatory stages of knowledge acquisition, unstructured interviews can help to augment information gleaned from published sources. Unstructured interviews can be a source of deeper knowledge when used after observation or verbal reporting. Observation is best applied after some background information has been assimilated by the knowledge engineer. Thus, the timing of its employment is dictated by the availability of background knowledge.

Self-reporting can be used as the first method for knowledge acquisition after the familiarization phase. Structured interviews are useful in helping to clarify and enhance knowledge derived from self-reporting and other methods [30]. Special tasks such as limited information or constrained processing tasks are recommended to help fill in gaps that the acquired knowledge base may exhibit [14].

Regardless of the methods chosen, the process of knowledge acquisition is an iterative one. That is, knowledge is abstracted from observations and self-reporting. This knowledge is used as the basis for some interviews. More observations may be conducted and result in a larger knowledge base, which can then be refined using structured interviews. Special tasks can then be developed to help complete the knowledge acquired.

It is useful to audiotape and, wherever necessary and possible, videotape the sessions with the expert. All the knowledge acquired should be kept in a knowledge document, written in terms that can be understood by the human expert. With each session the knowledge obtained should be added to the knowledge document. Once sufficient knowledge has been documented, it should be implemented in a prototype system. Whenever a session has yielded enough knowledge the prototype as well as the knowledge document should be altered. Each update of the prototype should be tested using some sample problems and comparing the system's solutions with those suggested by the human expert [30]. Rigorous testing of the prototype is necessary to enlarge upon the existing knowledge and modify it when necessary. A large number of test cases should be used and an effort should be made to select a variety of problems within the domain. Wherever the system disagrees with the human expert's own results, the differences between the two should be examined thoroughly. These differences will often point to missing or incomplete knowledge. Testing the knowledge document by hand is also recommended if the knowledge has not yet been entered into the prototype system [30]. The prototype prepared in this fashion becomes the basis of the final product and the knowledge document will serve as useful documentation for the system.

2.3 Automated Knowledge Acquisition

2.3.1 The Need for Automated Knowledge Acquisition

The difficulty of knowledge acquisition is frequently cited as the main stumbling block in expert system development. The knowledge acquisition task is usually very protracted and often is not consistently productive. Its success hinges largely on the availability and cooperation of the domain expert. Since there are many demands on the expert's time and services, the knowledge engineer's claim for attention is often judged to be the least urgent of a seemingly endless queue. Because of the many hurdles encountered in acquiring domain knowledge the building of a prototype expert system that can function with some level of expertise typically takes from six to twenty-four months [13].

One of the main functions of computers is to speed up lengthy routine tasks such as number and other data processing. The notion of automating an exceptionally lengthy manual process is therefore intuitively appealing to computer scientists. Indeed, it would be an anomaly if knowledge acquisition, a task common to the development of all expert systems, were to continue exclusively as a manual process.

2.3.2 Types of Automated Systems Developed

From the beginning of expert system research, computer scientists have sought some form of automation of the expert system building process. Some of the early expert system developers approached the task by extracting some salient features of their systems that were not domain specific. From these they produced new programs that could be adapted to new expert systems operating in different domains. In this way MYCIN, a medical consultation expert system, gave rise to EMYCIN, an expert system development environment, and PROSPECTOR, an advisory system for finding ore deposits, was the forerunner of KAS [13]. Other programming tools for the development of expert systems, although not directly descended from a specific system, were developed as the result of their designers' experience in building expert systems that dealt with particular problem types. Thus, EXPERT was de-

veloped by a group of designers who had extensive experience in developing medical consultation models. These systems are all frameworks for the building of expert systems and were the forerunners of many of today's expert system shells [13].

Although expert system shells greatly speed up the process of building an expert system, they have not appreciably simplified the "bottleneck" in expert system development, knowledge acquisition. The efficient extraction of knowledge from experts and its subsequent transformation into a form readily used by the expert system is still a job that is often done manually.

Some early automated knowledge acquisition systems were developed for specific systems to help augment and maintain an already existing knowledge base. One such system was TEIRESIAS, developed as an offshoot of MYCIN to assist in acquiring, correcting, and using new knowledge for the MYCIN system [13].

Automated knowledge acquisition has been approached in two ways. One method employs an interactive program to interview an expert and use the answers given to build a knowledge base [1]. Another technique involves case-based machine learning, using historical examples and extrapolating a knowledge base from the solution process used [20].

MOLE is a successful interactive knowledge acquisition tool that is used for systems that solve problems using heuristic classification. The knowledge acquisition tool knows that the problem solving method used makes some key assumptions about the domain and how knowledge can be used to search for problem solutions. Armed with these assumptions a knowledge acquisition program uses simple interactive facilities to elicit only the knowledge required by the problem solving method and to verify its adequacy. [6, 12, 19]

2.3.3 Two Automated Knowledge Acquisition Systems

The type of problem that an expert system is meant to solve helps to dictate the problem solving method that will be used and the types of knowledge that will be useful in the solution process. Two of the problem types undertaken by expert sys-

SYSTEM	PROBLEM TYPE	DEVELOPER
ETS	Classification	John H. Boose, Boeing
AQUINAS	Classification	John H. Boose, Boeing
MOLE	Classification	L. Eshelman, Carnegie-Mellon
BLIP	No problem type	K. Morik, Technical U Berlin
INFORM	Classification	E.A. Moore, Applicon/Schlumberger A.M. Agogino, UCB
SALT	Construction	S. Marcus, Carnegie-Mellon
MORE	Classification	G. Kahn, S. Nowlan, J. McDermott, Carnegie-Mellon

Table 2.1: Knowledge Acquisition Systems and Problem Types

tems are classification or analysis problems and construction or synthesis problems.

Since many of the expert systems written to date deal with classification problems it is not surprising that most knowledge acquisition tools have been designed with a classification system in mind. Table 2.1 lists some knowledge acquisition systems, indicating their problem-solving orientation.

Interactive knowledge acquisition tools often parallel the work done in manual knowledge acquisition. The process involves interviewing experts, transferring their knowledge to some intermediate knowledge document, analyzing the knowledge, verifying its completeness, validity, and consistency, transforming the knowledge into rules or other representation, and testing the problem-solving efficacy of the rules. To a lesser or greater degree, most of the knowledge acquisition tools developed to date have incorporated these tasks into their systems. A brief description of two knowledge acquisition programs, one for classification systems and one for construction systems, follows.

The Expertise Transfer System (ETS) incorporates personal construct theory and repertory grid techniques, both borrowed from the field of psychology, into a

methodology to acquire domain knowledge for solving analytic problems. Using these and some statistical analysis methods, the system interviews an expert, analyzes the acquired knowledge, creates heuristic production rules, and generates knowledge bases for KS-300 and OPS5. The prototypes are then used to run test consultations to establish the merit and sufficiency of the knowledge base. At various stages in the process the expert is shown the generalizations made by the system and may elect to correct any apparent inconsistencies. The prototypes prepared by ETS are not fully operational expert systems. Interviewing of the expert by a knowledge engineer is still required as is further knowledge refinement. The automatic generation of prototypes, however, considerably shortens the process of constructing an expert system. [1]

SALT is a knowledge acquisition system for expert systems that solve construction type problems [9, 19, 18]. A principal characteristic of these expert systems is that their knowledge is partitioned according to its role within the process. SALT uses its understanding of how these expert systems function and of the different types of knowledge to acquire knowledge from the expert. The knowledge acquisition is done interactively. SALT interviews the expert using menus to help the expert select the type of knowledge being addressed. During the process, SALT employs its grasp of knowledge types and their necessary attributes and pre-conditions to prompt the expert and help collect the necessary knowledge. After the knowledge has been analyzed to remove any possible irregularities, and corrected with the interactive help of the expert, the system produces OPS5 rules for a target expert system shell.

These and other interactive knowledge acquisition programs have easy to use interfaces. They do not assume that the human expert being interviewed is familiar with either the knowledge acquisition systems or with Artificial Intelligence concepts. A brief explanation of the systems is sufficient for their use by a novice in knowledge acquisition.

2.4 Case-Based Knowledge Acquisition

2.4.1 The Case-Based Approach

Definition *Case-based knowledge acquisition* is a method used to learn domain specific knowledge from examples drawn from the domain and to generalize the knowledge using inductive techniques.

Definition A *training set* is a set of sample cases of solved problems within a domain that are used to extract pertinent information about the domain.

Definition A *test set* is a set of sample cases of unsolved problems in the domain that are used to verify the accuracy of the information learned from the *training set*.

The case-based approach to knowledge acquisition postulates that a set of examples showing how the expert solved individual problems can present an accurate portrait of the problem domain. Therefore, by extracting knowledge from the examples and expanding it with induction methods, it is possible to build a knowledge base that can be used to solve a set of related problems in the domain.

The process of case-based knowledge acquisition normally spans several stages. Initially problems typically solved in the domain are defined by an expert or non-expert practitioner in the domain. The key elements used in arriving at solutions along with the related solutions are identified at this time. Next, a training set of cases that demonstrate the expert's problem solving expertise is assembled. The design of a program to extract important factors and relations from the training set and an induction algorithm to analyze and generalize the derived data comprise the next step of the process. After the program and the induction algorithm have been written, the actual knowledge acquisition takes place. The knowledge acquisition program extracts relevant information from the training set and the induction algorithm transforms this information into more general rules suitable for solving domain problems. Finally, the rules are applied to a test set of unsolved problems. Any problems within the test set that are not solved by the rules are then used along with their solutions to form part of a new training set for the next iteration of the final two steps. These iterations are intended to contribute to the improvement of the rule base [24].

This method aims to automate knowledge acquisition in a manner that reduces the obstacles presented by the manual process. The interview process where the expert is expected to reveal all the secrets of the trade in response to the knowledge engineer's probing is eliminated, as are many of the other techniques used by knowledge engineers to elicit expertise in problem solving. To suit the requirements of case-based knowledge acquisition the primary role of the knowledge engineer has been altered. After a preliminary period of domain familiarization, the knowledge engineer is primarily occupied with designing and writing the knowledge acquisition program and the induction algorithm. Once this has been accomplished, a case-based knowledge acquisition system can run virtually independently of the knowledge engineer. However, the knowledge engineer would become involved again during the testing and knowledge base refinement phases.

2.4.2 Case-Based Manual Knowledge Acquisition

Case-based knowledge acquisition is also manifested in traditional methods of knowledge acquisition. One of the techniques used in manual knowledge acquisition is that of observation. By observing the expert at his or her regular task the knowledge engineer hopes to gain insight into the workings of the domain and in turn transfer this understanding into a knowledge base. This is a situation in which the knowledge engineer is learning about the problem domain from actual examples. Another such situation involves self-reporting in which the expert describes the actions being taken in the process of solving the problem.

Other manual knowledge acquisition techniques involve learning from sets of examples that have been deliberately devised for that purpose. These are the situations in which the expert is required to solve problems with limited information or problems where constraints have been placed on the expert's resources, thereby changing the reasoning strategies used. Such cases are also used by the knowledge engineer as examples from which knowledge can be gathered.

One of the tasks of the knowledge engineer is to attempt wherever possible to generalize the knowledge that has been acquired. In this fashion, manual knowledge

acquisition, like its automated case-based counterpart, involves using some means to expand the specific knowledge elicited to more general forms.

2.4.3 When To Use Case-based Knowledge Acquisition

It is apparent from the previous section that case-based knowledge acquisition is a characteristic of some manual knowledge acquisition methods. Therefore, what distinguishes case-based knowledge acquisition from the manual process is the fact that in the former learning from examples is the primary method used while in the latter learning from examples is only one of several techniques followed.

There are certain attributes characterizing situations in which case-based knowledge acquisition is the appropriate methodology to pursue. An application domain where experts are not readily available or where experts who are accessible cannot easily convey their decision processes would not be suitable for manual knowledge acquisition. However, if such a domain were finite with clearly definable problems, and if a set of examples representative of the application area were available, the domain would be appropriate for case-based knowledge acquisition.[24]

2.4.4 Limitations of the Case-based Approach

The success of case-based knowledge acquisition depends on and is therefore limited by three factors: the training set, the test set, and the induction algorithm. To ensure that the knowledge extracted from the cases is accurate and complete it is important that the training set be representative of the domain, depicting a sufficient variety of domain problems and issues. Similarly, the test set must also comprise typical examples to affirm that the process of verifying the reliability of the acquired knowledge is effective and accurate.

The responsibility of selecting representative cases for the training and test sets should be undertaken by someone who is well versed in the characteristics of domain problems. In order to confirm the relevance of the training and test sets it may be useful to employ several sets of cases for knowledge acquisition and compare and

combine the elicited information.

In most instances, the knowledge acquired from the training sets of cases is specific to the cases studied. The induction algorithm is the tool used to analyze this specific information and expand it into more generally applicable knowledge. Thus it is important that the induction algorithm be suitable for developing accurate and valid general rules from the limited information extracted from the examples. The induction algorithm's effectiveness depends on isolating the fundamental features of the training cases that should be thoroughly analyzed and elaborated upon to yield general information. These important factors should be identified during the initial stage of familiarization with the domain. The success or failure of the induction algorithm will become apparent when the newly constructed rules are applied to the test cases. If some test cases are not solved satisfactorily by the expert system, and if, in turn, using solutions for these test cases as a training set for a subsequent iteration of knowledge acquisition does not result in improved rules, then the induction algorithm may be faulty and may need to be rewritten. Alternatively it may be determined that the training set is not representative of the domain and needs to be replaced.

Acquiring knowledge from cases may only yield a limited rule base, which can serve as a starting point for further knowledge acquisition. The expansion of the rule base may be done either by using new and more complex training and test cases or by involving the domain expert in improving upon what was done automatically.

2.4.5 Case-based Knowledge Acquisition Research

Case-based knowledge acquisition or learning from examples has been at the heart of much of the research into machine learning and knowledge base construction. It has been the method used in Meta-DENDRAL dealing with the synthesis of chemical knowledge [2], and in AQ11 for soybean diagnosis [20]. It has also been successfully used in industrial contexts, such as in the development of a large expert system for British Petroleum to design hydrocarbon separation vessels [31]. There are also commercial expert system building tools that use induction as a tool for deriving

rules from examples. Among these are RuleMaster, Ex-Tran, and 1st-Class [31, 21]. These systems represent the extracted rules in the form of decision trees.

As in other branches of expert systems research, there is ongoing work into machine learning in general, and learning by examples in particular. Some of the work has concentrated on automating the acquisition of the conceptual knowledge that is the precursor to the actual acquisition of domain specific, problem solving knowledge [10]. Other research has focused on simplifying decision trees, a representation that has frequently been used for rules derived from examples, thereby making the knowledge more comprehensible to human experts and users [31].

Chapter 3

BLACKBOX

3.1 Description of Blackbox

3.1.1 How Game is Played

Blackbox is a game consisting of a square grid that contains a number of hidden balls [28]. The objective of playing the game is to determine in which squares these balls are located. To locate the balls a player fires vertical or horizontal beams into the grid from positions along its four sides. Then the player analyzes the behaviours of the beams and hypothesizes on the possible locations of the hidden balls. The goal is to determine the exact locations of all the balls using a minimum number of shots. Sometimes, the game cannot be solved by analysis and the player is then obliged to stop with a partial solution or to propose a solution by guessing.

A beam that encounters a ball along its trajectory in a square grid of size **n** can exhibit one of the three following possible behaviour patterns:

Absorption: A beam is absorbed if a ball is in (i,j) and the beam travels along row i or column j. The absorbed beam disappears from view.

Deflection: A beam is deflected if a ball is in (i,j) and the beam travels along row (i+1) or (i-1), or along column (j+1) or (j-1). A deflected beam turns 90 degrees away from the ball and continues its trajectory along the column or row immediately before the ball's position.

Reflection: A beam is reflected if any of the following six conditions is true. A reflected beam is turned around 180 degrees and ultimately exits at its origin.

1. a ball is in (1,j) and the beam enters along column (j+1) or (j-1) from the top.

2. a ball is in (n,j) and the beam enters along column (j+1) or (j-1) from the bottom.
3. a ball is in (i,1) and the beam enters along row (i+1) or (i-1) from the left.
4. a ball is in (i,n) and the beam enters along row (i+1) or (i-1) from the right.
5. two balls are in (i,j) and (i+2,j) and the beam travels along row (i+1).
6. two balls are in (i,j) and (i,j+2) and the beam travels along column (j+1).

These beam behaviours are the domain rules of the Blackbox game and are applied in the order of precedence presented above. The game can be played by a person on a computer, using an interactive program developed by John Lyons and Kristina Pitula [16, 26]. The computer screen displays the game-grid and identifies beams' entry and exit locations with pairs of identical alphabet letters. The entry point of an absorbed beam is marked by the computer with the letter 'H' (for hit) and the entry/exit point of a reflected beam is identified with the letter 'R'. There is an inherent symmetry in Blackbox. If a beam fired at point X exits at point Y, then a beam fired at point Y will exit at point X.

Figure 3.1 shows some examples of shots and their trajectories. In the figure, the shots labelled 'a', 'b', and 'd' are deflected one, four, and two times respectively by balls located in their trajectories. Shot 'c' does not encounter any balls in the vicinity of its trajectory and thus passes through the grid without any deviations. The shot labelled 'H' (for Hit) is absorbed by the ball it encounters directly along its trajectory, and shots labelled 'R' are reflected back to their original entry points by balls whose locations in adjacent rows do not permit deflections and further penetration of the grid.

Performance in Blackbox is typically measured using two parameters - the score and the number of errors. In general, the score is a function of the number of shots and their relative behaviours. In our experimental studies a lower score indicates a better level of performance. In this scheme, deflection increments the score by 2 and the other beam behaviours increment the score by 1. The scoring is directly representative of the immediate visual result of the different shot behaviours. When a shot is deflected its entry and exit are both visible, affecting two points on the

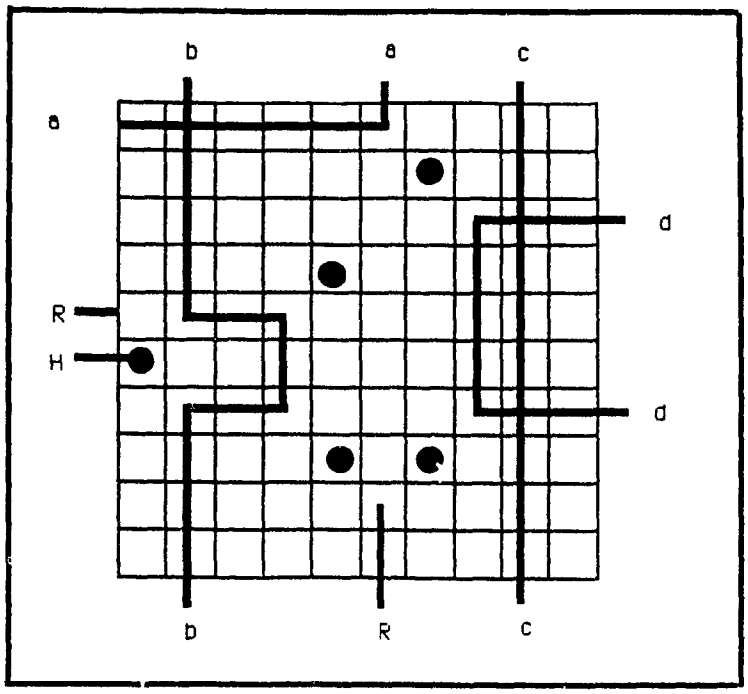

Figure 3.1: Blackbox Shot Trajectories

grid's perimeter. A shot that is absorbed and one that is reflected only show the shots' entry positions. The hypothesis deduced about the grid's internal configuration improves with more information about the grid's perimeter. Thus shots that reveal more information add more to the player's score than those that yield less information. Errors occur when the final hypothesis that the player makes about the ball locations does not correctly identify the locations of hidden balls. The number of shots taken by the player and the time elapsed to solve the game are other possible performance measures.

3.1.2 Distributed Version of Blackbox

In theory, a problem that is too large to be adequately evaluated and analyzed by one individual could be solved if the problem space and hence the problem solving responsibility could be divided into several smaller parts to involve several people. Blackbox, when played on a very large grid, can be representative of such a large problem. Distributed Blackbox (DBB) modifies the Blackbox game into a cooperative game played by more than one player.

Blackbox lends itself naturally to a geographic decomposition of the game space. A variety of partitioning approaches may be used. The grid can be divided into two or four parts, all of which would have several common physical characteristics - the same number of external edges, the same number of internal edges, and the same communication needs with neighbouring spaces. In such a partitioning scheme all the players would require the same skills and approach the problem from a similar perspective. Other methods of partitioning could also be considered. A larger grid could conceivably be divided into more than four parts: six or eight or even more. In such cases the separate games spaces would not all share similar physical characteristics. Some parts would have more external edges than others, and some would have more internal edges. The communication needs would differ among them and consequently the problem solving skills of the players would differ according to the game space they were assigned to solve.

Although the game is readily partitioned along geographical boundaries, the distinct problem spaces designated by these boundaries cannot be solved independently. Shots that originate in one section of the game may pass through other sections and events observed in one area may be indicative of a particular situation in the state of others. To arrive at a solution encompassing the entire game-grid of Blackbox, all the players must cooperate during the problem solving session. Thus Blackbox is suited both for problem decomposition and for cooperative problem solving among distributed components.

Of the various partitioning designs considered for Blackbox, the one dividing the game into four quadrants is intuitively appealing. The physical division of a game-grid with even dimensions (e.g. 8 x 8 or 10 x 10) into four areas of equal size results in square quadrants which outwardly appear like small Blackbox grids. But the similarity ends here. A player in a four-way partitioning of Blackbox has control over the two external edges in the assigned quadrant but must communicate with neighbouring quadrants to obtain information or request activity along the two internal edges. This design allows the player some measure of autonomy, while incorporating sufficient complexity into the cooperative solution process. This scheme refrains from overwhelming the player with multiple communication channels as a

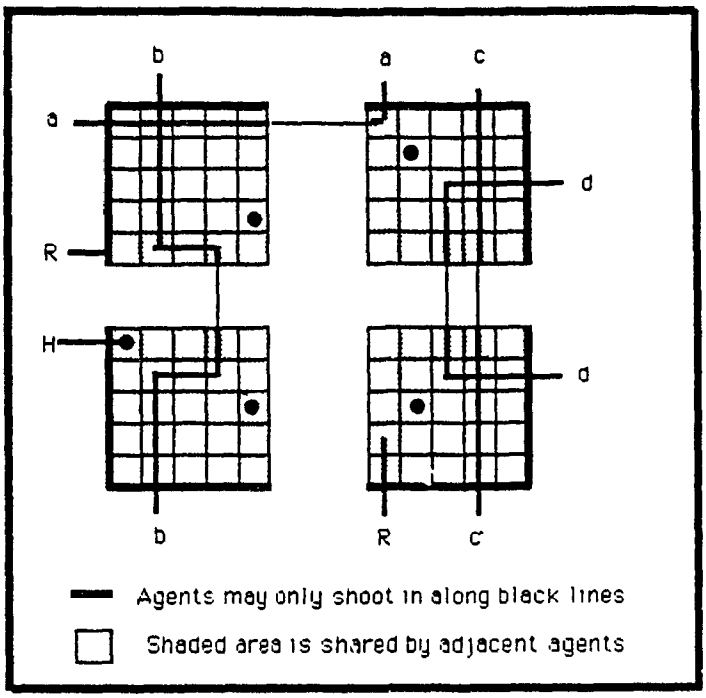

Figure 3.2: Distributed Blackbox

more extensive distribution pattern would do.

In DBB the game grid has been partitioned into four geographically distinct regions each conforming to one quarter of the original grid. In this format of the game each player is responsible for solving a different quadrant of the grid, communicating partial solutions to the other players in an effort to solve the global game. In DBB each player can only shoot beams into the grid from locations along the two exterior edges of the quadrant. The remaining two edges border on the adjacent quadrants and beams in transit can pass through them to neighbouring quadrants. (See Figure 3.2). The protocol used for playing the game has players taking turns shooting beams into the grid using a simple round-robin, token-passing control scheme.

DBB is a cooperative game to which all the players contribute their individual efforts to maximize the group's performance. The involvement and participation of all the players is necessary for the effective solution of the game. The game produces no losers and does not entail competition among its players. Therefore, people are "comfortable" playing DBB. The Blackbox performance measures of score, number of errors, number of shots, and time can be used to evaluate the performance of

teams playing DBB.

The partitioning of Blackbox into DBB was done for the purposes of studying interaction among cooperating computer users. The interaction can be studied quantitatively by comparing the performance measures achieved in DBB with those achieved in Blackbox. A qualitative study of the interaction can also be undertaken to determine the levels of satisfaction derived by the group in playing the game.[4]

3.1.3 Blackbox and Expert Systems

Blackbox and Distributed Blackbox can both be played by human players or by expert systems. In fact Blackbox is eminently suitable for an expert system implementation, for the following reasons:

1. The solution process entails a great deal of ambiguity. That is, although partial solutions, in the form of partial hypotheses, are formulated at intermediate stages of the game playing, these incomplete solutions are not always certain until the comprehensive solution has been determined at the end of the game.
2. The state space representation of the game is straightforward.
3. Hypotheses can be formulated and refined at different stages of the game.
4. The performance evaluation is simple and straightforward.
5. An exhaustive solution is uninteresting.
6. Heuristics can be applied in hypothesis formulation and shot selection.
7. The knowledge engineering involved does not require knowledge from an external expert and hence is not very time consuming.

The last point mentioned makes Blackbox an interesting testbed for a university environment and particularly suitable for this experiment in knowledge acquisition.

3.2 Knowledge Representation in Blackbox

3.2.1 Temporary Knowledge

Definition The *real grid* is an object representing what is known about the Blackbox grid by the *game-managing module* - its size, the number of hidden balls, and their locations.

Definition The *hypothesis grid* is an object representing what the *playing agent* knows about the Blackbox grid and what it determines as the game progresses - the size of the grid, the number of hidden balls, the entry and exit locations of shots taken, the number of balls located, and the contents of each square of the grid, unknown, empty, or a ball.

Definition The *game-managing module* is the part of the Blackbox game system that is responsible for determining the trajectory of each shot, and marking the shot's entry and exit locations in the case of a deflection, or the shot's behaviour in the case of a reflection or an absorption.

Definition The *playing module* or *playing agent* is the part of the Blackbox playing expert system that shoots beams into the grid and makes hypotheses based on the trajectory of the beams determined by the game-managing module.

The temporary knowledge in a game of Blackbox consists of representations of two grids, the real or actual grid and the hypothesis grid. The real game-grid incorporates the knowledge that the game managing module possesses about the grid - its size, the number of balls hidden within its boundaries, and the locations of the hidden balls. While the playing module is also cognizant of the size of the grid and the number of balls, it does not know where the balls are hidden.

The real grid remains static for the duration of one game of Blackbox, changing only with the start of a new game session. The grid contents can be represented by a two dimensional array. This may not be the most memory-efficient method. However, ordinarily the Blackbox grid size used in our experiments is no larger than 10 x 10, and thus the total amount of memory used is not very significant. Even a very large game with a 20 x 20 grid of dimensions would not require an excessive amount of memory, using a maximum of 400 bytes of storage. The information maintained in the grid's array representation is the contents of each element in the array, whether it is empty or a ball. Alternatively, the grid's contents can be maintained in a linked list of grid coordinate pairs, one for each location containing a ball. All locations not mentioned in the list are assumed to be empty. The size of the grid can be stored in an integer variable which would then be used to determine the grid boundaries. The number of balls hidden in the grid can also be kept in an integer variable.

Definition The *available shots* are points around the grid's perimeter which have not yet been used as the entry or exit point of a previous shot.

The hypothesis grid knowledge comprises what the playing expert knows about the real game grid, available shots, shots (entries and exits) as they are made, as well as the player's hypotheses about the locations of the hidden balls. The bulk of the information in this data section is dynamic, changing with each shot that is taken and with each hypothesis that is proposed or modified by the expert system. As in the real grid the size and the number of balls can be represented by integer variables, and the hypothesis grid contents by a two dimensional array. The hypothesis grid locations are all initially labelled "unknown" and, as the game progresses and the hypothesis is modified, individual array elements may be labelled "empty", "ball", or "possible ball".

In addition to the above data structures, an integer variable is used to store the number of balls located in the hypothesis grid. A list of available shots is maintained as a doubly linked list of x and y coordinates, from which locations are removed as shots originate from or exit through them. Two other dynamically changing shot lists are kept. One is a list of uncertain shots, which contains the number, entry and exit locations, and a letter identifying the beam behaviour of fired shots whose trajectory has not yet been analyzed by the expert system. The other, a certain shots list, consists of data describing shots that have been fully analyzed for the purpose of making an hypothesis. Also maintained is a linked list describing the locations of balls located by hypothesizing about shot behaviours.

For the purposes of an explanation facility a list of rules that have been fired along with the specific shots that triggered them is also kept as part of the dynamic knowledge represented in a Blackbox playing expert system.

3.2.2 Permanent Knowledge

Definition A *partial hypothesis* is a modification made to the hypothesis grid by an expert system as the result of analyzing one or more shots.

The permanent knowledge in Blackbox includes the rules that the expert uses to play the game. These rules are divided into analysis rules and shot-selection rules. The analysis rules are responsible for forming partial hypotheses based on the behaviours

exhibited by shots fired into the game-grid. Essentially, an analysis rule consist of an antecedent incorporating a description of a shot whose occurrence within the game causes the rule's consequent to be performed. The consequent involves changing the contents of locations in the hypothesis grid. This is done by placing a "ball", a "possible ball", or an "empty" indicator within squares of the hypothesis grid.

The shot-selection rules' antecedents involve shots that have previously been fired as well as their exit behaviours. These result in consequents that establish from which location the next shot is to be fired.

In addition to the game playing rules of analysis and selection, the permanent knowledge includes some "housekeeping rules" that are used to maintain the various components of the dynamic data related to the hypothesis and its ancillary pieces of information. There are also some "meta-level rules", that can provide further analysis of the hypothesis grid created by the application of the rules or handle the resolution of a partial hypothesis at the end of a game.

A comprehensive explanation of the knowledge representations used for the Blackbox expert system is given in Chapter 5, which describes the design of the actual expert system used for this knowledge acquisition experiment and the rule base that resulted from the experiment.

3.3 Knowledge Acquisition and Blackbox

3.3.1 Manual and Automated Knowledge Acquisition

Section 3.1.3, discussing Blackbox as a suitable problem for an expert system, noted that the knowledge engineering involved in a Blackbox playing expert system would not require an excessive expenditure of time. This is due to the fact that experts in the game were not difficult to locate. In fact, since several projects involving Blackbox as their testbed had already been undertaken within our University, the experts were available in our midst. The question then arose, which methods of knowledge acquisition were appropriate to extract the expert knowledge required to play the game.

The knowledge utilized in the playing of Blackbox can be roughly divided into two components. The first part is a clear understanding of the domain rules of Blackbox as described in Section 3.1.1 and a knowledge of how to use these rules in analyzing shot behaviours. The second part involves the selection of shots to fire in the game, depending on what is known about the grid configuration at a given point in the game.

Manual knowledge acquisition is a viable option for acquiring these elements of Blackbox playing expertise. A combination of interviews and observations could be used to query the expert and reveal his or her knowledge. Automated knowledge acquisition in the Blackbox domain presents a knowledge engineer with a dilemma. Into which genre of problems does Blackbox fall? Is it an analysis problem or is it a synthesis problem? The answer is both. The shot-selection knowledge can be seen as an exercise in synthesis much like planning or scheduling, whereas the shot analysis knowledge is an analysis problem, deriving solutions from exhibited behaviours or symptoms. This combination of knowledge types might be difficult to capture in an automated interactive interview process.

3.3.2 Case-Based Knowledge Acquisition

The alternative approach to manual and automated knowledge acquisition is automated case-based knowledge acquisition or learning from examples. Case-based knowledge acquisition is appropriate for a finite domain in which the problems are clearly definable and a set of examples representative of the application area are available [24]. This method requires little or no interaction with a domain expert. In addition to these criteria, it should also be possible for the knowledge engineer to learn background information and understand which are the key elements used in arriving at solutions as well as the related solutions.

The game of Blackbox meets all of these requirements. The domain is indeed finite, being located in a square grid of limited dimensions. The problem of discovering balls hidden in the grid can be stated clearly and concisely. The background knowledge of the factors that contribute to solving the problem is readily available

from a Blackbox player's manual [28] and from experts in the game.

Definition A *game file* is a text file used to store historical data from a set of previously played Blackbox games.

The most important requisite for case-based knowledge acquisition is the availability of sets of examples for learning and for testing the newly found knowledge. The version of Blackbox written by Kristina Pitula for playing on a PC has the facility to save data from completed games in computer text files known as game files [26]. These files therefore provide historical data on actual domain problems and their solutions in a form accessible for computerized analysis. The game files store the chronological actions taken during the playing of individual games of Blackbox by an expert human player. The score achieved by the expert is also stored as well as any errors that may have been made in the final solutions. The background knowledge of the game can be used to write an analysis program and design an induction algorithm. The first would be used to analyze the data stored in some game files, extracting essential factors and using the inter-relations they exhibit to form preliminary rules. The induction algorithm would focus on specific attributes of the game and analyze the preliminary rules, transforming them into general rules to be used by an expert system to play the game. The grid configurations of other game files can be used as testing sets for verifying the acquired rules and as the basis for future iterations of the knowledge acquisition process.

A final phase in the thorough application of case-based knowledge acquisition would be to have the expert evaluate the acquired rules and their performance and, if necessary, suggest ways in which the rules could be improved. Here too Blackbox meets the necessary condition since experts in Blackbox are not difficult to find.

3.4 Knowledge Acquisition from the Distributed Game

3.4.1 Distributed Knowledge Acquisition

Most expert systems written to date have incorporated knowledge acquired from a single domain expert. This fact has helped reinforce the notion that, in addition to a knowledge engineer equipped with the appropriate tools, such as a suitable programming language or a compatible shell and some knowledge elicitation techniques, one expert is all that is needed to build a knowledge base [13]. This concept has been questioned, particularly the assumption that the knowledge that can be extracted from one expert in a given domain can be said to reveal all expertise on the subject [22].

At the same time, the suggestion has been put forward that knowledge should be elicited from several experts in a domain and combined to capture the essence of wisdom in the field. This is in effect a distributed approach to knowledge acquisition. The different experts may attempt to solve the whole problem from different perspectives using different techniques. Or, alternatively, the different experts may not have expertise sufficient to solving the entire problem but each only possessed proficiency in some portion of the issue. In yet another type of situation, the experts could be involved in solving a problem that has been partitioned for distributed solution. Thus the work involved in extracting the knowledge, comparing, combining, and collating it, is distributed in nature.

3.4.2 Possible Benefits

Acquiring knowledge from multiple sources can provide a solution to some difficulties commonly seen in knowledge acquisition. The inability of experts to dig deep into their subconscious minds and explain how and why they do their work is frequently mentioned in the literature [14]. If several experts in a field are being questioned and observed there is little chance that all of them would share this difficulty in the same aspects of the domain problem. Another problem faced by the builders

of knowledge bases is the expert's inability or reluctance to devote sufficient time to the project. With a number of experts being employed in the process, if one expert is inordinately busy, another can step into the breach and contribute his or her knowledge in the area. Thus using a greater number of experts can be a start in overcoming some of knowledge acquisition's major handicaps.

One of the express aims of expert systems is to acquire sufficient knowledge to be able to solve obscure problems with very little information. The experts attempting to solve a portion of a distributed problem are required to do so with less than complete information. Observing the experts at their distributed tasks and analyzing their individual actions may yield the deeper knowledge and heuristics used with the limited information available within the boundaries of the segments of the problem. Since distributed problem solving employs communication between the partitions, knowledge acquisition could also uncover the communication patterns being used in the problem solving sessions. This communication knowledge could be used in the building of distributed expert systems.

The knowledge acquired from several experts working independently to solve identical problems or cooperatively to solve a distributed problem would need to be compared and then combined to produce a cohesive logical knowledge base. In the case of experts working to solve a distributed problem, the knowledge derived could be compared and combined with that of a single expert solving the whole problem. Such comparisons could give a greater assurance of the validity of the knowledge acquired from either source and the combination of the most powerful heuristics could provide a more robust knowledge base.

3.4.3 Potential Difficulties

Distributed knowledge acquisition could present a great challenge to the knowledge engineer. The task of interviewing and observing several experts at similar jobs or at different parts of the same job could prove to be overwhelming. This would appear to argue for an automated approach, and, if sufficient cases of previous problems exist, for case-based knowledge acquisition. Organizing the cases of different experts might

be a complicated task but one that offers interesting opportunities. For instance, knowledge gained from the cases of one expert could be tested on cases provided by another expert to yield broader experiences.

One problem that might arise in the knowledge acquisition process is that of effectively comparing and combining the knowledge acquired from several expert sources. Since different experts might approach a similar problem from different points of view or using different methods, resolving these differing points of view and finding a representation scheme that is suitable for all might be a difficult task.

Another issue that should be addressed is the validation and comparison of knowledge from different sources. Are performance measures sufficient criteria for selecting one path rather than another? Since the knowledge does come from experts, the resolution of their differences for the purposes of building a coherent knowledge base for an expert system may have to be referred back to them or to another expert who was not at all involved in the elicitation process.

3.4.4 Distributed Blackbox and Knowledge Acquisition

A distributed version of the Blackbox game was developed for an experiment involving human players. It was used to study "organization", "control" [32], the amount of computerized or non-computerized support required, and the patterns of "communication" between the players. In the study four players solved the distributed game and then the results obtained were compared with the performance of an individual playing the non-distributed game [4].

In the course of the experiment the game playing data was saved in computer text files. This or similar data could be the basis of future case-based knowledge acquisition experiments on the distributed game experiences.

Chapter 4

IMPLEMENTATION OF CASE-BASED KNOWLEDGE ACQUISITION

4.1 Synopsis

This chapter will describe the process that transpired in the automated acquisition of knowledge from actual cases of Blackbox games stored in game files. There were several stages in the process, beginning with an initial familiarization period and ending with writing rules in a format acceptable to the target expert system shell.

In case-based knowledge acquisition two essential ingredients are a representative set of cases, known as the *training set*, used to acquire the initial knowledge, and another equally representative set, known as the *test set*, used to validate and improve on that initial knowledge. In addition to the training and test sets a third ingredient that is necessary for automated case-based knowledge is a data analysis program to do the actual acquisition of knowledge from the cases. In the automated acquisition of knowledge from Blackbox both the training set and the test set were drawn from cases in game files that recorded game sessions played by experts. Before these sets could be selected and the data analysis program written to perform the knowledge acquisition, it was necessary to conduct the preparatory phases in the process.

The first preparatory phase entailed becoming acquainted with the domain and developing familiarity with the format of the stored game files. Learning to understand the roles of the individual elements of data in the game files was a necessary prerequisite to making some decisions essential to the planning of the knowledge acquisition. One important decision was on the representation of the acquired knowledge. This was followed by a selection of the type of inference engine that would be suitable for the expert system, backward chaining or forward chaining. Once these design decisions were resolved, the decision involving the selection of cases for a training set and a test set could be made.

The preparatory phase and deciding on the required design features yielded the necessary information to begin writing the program to read the stored data, analyze it, and extract information for the expert system's knowledge base. After the program was written, it was tested on the selected samples and any deficiencies encountered in the testing were rectified. The final version of the program was used to analyze the data available in the training set and write rules suitable for the expert system shell. These rules were then placed in the expert system's rule base and the expert system was ready for testing.

4.2 Familiarization

4.2.1 Background Knowledge

The preparation phase is an essential part of all knowledge acquisition projects, whether manual or automated, interview or case-based methods are used. At first the knowledge engineer must obtain a rudimentary understanding of what happens in the domain, identifying the problem to be solved and some factors that contribute to the solution. This knowledge, like the background knowledge of most knowledge acquisition undertakings, can be obtained from workers in the domain or from published data.

In Blackbox, the domain background knowledge consists of a description of the game board, the game's goals, and the rules used to play the game. This information,

which appears in Section 3.1.1, has been documented in several technical reports and conference papers produced by Cliff Grossner, Kristina Pitula, and Carol De Koven [4, 11, 28, 29, 27]. The game's rules provided an adequate understanding of the functioning of the game. Informal conversations with an expert in the game yielded a deeper understanding that could be applied to the knowledge acquisition program and the induction algorithm. Unlike more complex problem domains, such as medical diagnosis, the Blackbox domain is not too complex, offering the knowledge engineer the possibility to actually play the game and attempt to solve the puzzle. This opportunity to participate in domain problem solving was a further aid in understanding the background knowledge and the data that the game playing produced.

4.2.2 Case Data

Once the background knowledge had been assimilated, the focus of the familiarization process shifted to understanding the case data to be used for knowledge acquisition. It is necessary to become familiar with the structure in which the case data is presented and learn to identify what the individual data elements represent. In many domains this learning process would require the assistance of someone acquainted with the storage of historical data in the domain and with its usage. In some domains, stored data bases may be well enough documented that the knowledge engineer would only need to consult them. Whatever the means used to achieve it, understanding the data and the role it played in solving domain problems contributes significantly to the design decisions that would be addressed in the next phase of the knowledge acquisition.

The case data used for knowledge acquisition in the Blackbox domain was collected automatically as the games of the training set were played. For each game this data was saved in five sections.

1. Game board configuration.
2. Errors in the final solution, if any.
3. Score achieved in playing the game.

4. List of shots taken.

5. List of hypothesis modifications made in the game.

The data in the last two sections is stored and numbered chronologically, thus detailing the sequential steps taken in each game. Figure 4.1 shows data collected from the playing of a 4 X 4 game with two hidden balls.

The GAME BOARD in the game data gives the coordinates of the squares where the balls are hidden. The SHOTS and HYPOTHESIS MODIFICATION sections provide a detailed individual description for each of their respective actions. In each of the sections there is a sub-heading entitled "ACT_NO". These are the sequential numbers given by the "Blackbox game-managing module" (defined in Section 3.2.1) to the individual events occurring in the game. Thus from sequential perusal of the data, one can faithfully reconstruct the process taken to solve the game. It can therefore be observed that the first act in the game is always a shot and the last act is a hypothesis modification. A shot may be followed by a modification or by another shot. In most instances, a modification is followed by a shot or no action.

In the game shown in the figure, there are four shots recorded in the SHOTS section, each described by four sub-headings in addition to "ACT_NO". Each shot has a "SHOT_NO", which is a number reflecting its chronological ordering. Each shot is also assigned a "LETTER", which is displayed on the monitor on the perimeter of the game grid at the origin and exit of the shot, describing at a glance which of the three possible behaviour patterns the shot exhibited. An upper-case letter indicates that the shot exited the grid at a point different from its origin, after zero or more deflections in its trajectory. Upper-case letters are assigned to each shot in alphabetical order. An 'r' signifies a reflection of the shot out of the grid at its point of origin and an 'h' means absorption of the shot caused by a hit. The remaining two sub-headings, "IN" and "OUT", give the grid coordinates for the shot's entry and exit locations, respectively.

In the HYPOTHESIS MODIFICATIONS section of Figure 4.1 three hypothesis modifications are represented as separate sub-sections according to their individual "ACT_NO"s. Within each "ACT_NO" there are three units of information describ-

GAME BOARD
Two Balls located at (2, 1) and (4, 2)

SHOTS	ACT_NO	SHOT_NO	LETTER	IN	OUT
	6	4	r	(5, 1)	(5, 1)
	4	3	B	(5, 4)	(0, 4)
	2	2	A	(0, 2)	(1, 5)
	1	1	h	(0, 1)	(0, 0)

HYPOTHESIS MODIFICATIONS

ACT_NO 7:	LOCATION	OLD VALUE	NEW VALUE
	(4, 1)	unknown	empty
	(4, 2)	unknown	ball

ACT_NO 5:	LOCATION	OLD VALUE	NEW VALUE
	(3, 3)	unknown	empty
	(3, 4)	unknown	empty
	(4, 3)	unknown	empty
	(4, 4)	unknown	empty

ACT_NO 3:	LOCATION	OLD VALUE	NEW VALUE
	(2, 1)	unknown	ball
	(2, 2)	unknown	empty
	(2, 3)	unknown	empty
	(2, 4)	unknown	empty
	(1, 4)	unknown	empty
	(1, 3)	unknown	empty
	(1, 2)	unknown	empty
	(1, 1)	unknown	empty

PLAYER ERRORS **Total Errors: 0**
SCORE 6

Figure 4.1: Blackbox Game Data

ing the modification that affects one square of the grid. The "LOCATION" gives the x, y coordinates of the square on the grid. The "OLD VALUE" reflects the status of the square prior to the hypothesis modification, and the "NEW VALUE" is the square's changed status after the modification.

The last two sections record the player's performance in one game session. The PLAYER ERRORS would list the errors made in the final hypothesis if the TOTAL ERRORS were greater than zero. The SCORE achieved in the game session in the figure is 6. As explained in Section 3.1.1, a hit and a reflection increment the score by 1 and a deflection by 2. Thus in the figure the score includes 1 for each of shots 1 and 4, which are labelled 'h' and 'r', and 2 each for shots 2 and 3.

4.3 Knowledge Representation

4.3.1 Preliminary Data Storage

The understanding of the domain and its data structures that is achieved during the familiarization phases of the knowledge acquisition operation helps in resolving some important design issues. The first is selecting the knowledge representation most suitable for the data at hand.

The Blackbox background knowledge showed that the game is solved by a human player using the domain rules of the game in an attempt to trace out plausible trajectories for previous shots whose only known attributes are their behaviours and their entry and exit points. This information, combined with the observed sequence of events in the game, described by the "ACT"s in the game file, helped to explain the roles assumed by the different data components. The chronology showed how the different actions in a game history created a thread unravelling the puzzle and indicated how the analysis of the data could classify them. The interleaving of shots and hypothesis modifications in the pattern of most games showed that cause and effect relationships could be extracted and used in the formulation of rules on the playing of the game.

The first step was to decide how best to represent the information that would be

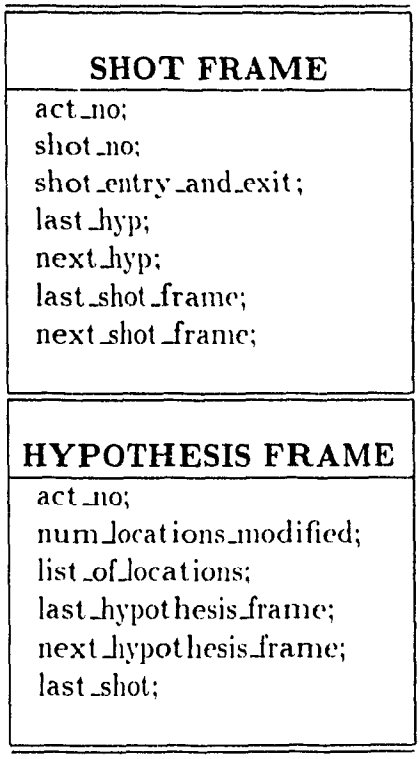

Figure 4.2: Shot and Hypothesis Frames

extracted from the cases. To follow through from the structures used in the storage of the data, a frame-based representation scheme seemed appropriate. Four different frame structures were designed to hold the data extracted from the game files. Two of the frames were to give statistical overviews of the games. One was to contain the statistical information for each game, including the game number, the number of balls hidden in the grid, the errors, if any, and the score. The other statistical frame was to store the measurements for a complete file of games, comprising the number of games played, the average score, and the average error per game. The remaining two frames were a "shot frame" and a "hypothesis frame", whose structures are shown in Figure 4.2.

The shot frame was used to store pertinent data that appeared in the SHOTS section as well as some additional information. The "ACT_NO", "SHOT_NO", "IN", and "OUT" attributes of individual shots were extracted from the SHOTS section and stored in the shot frame. By visually examining the data it was noted that

whenever the "LETTER" assigned to a shot was 'h', the "OUT" of the shot was '00', and whenever the "LETTER" was 'r', the "OUT" of the shot was the same as the "IN". The alphabetic identification given to the shots tended to break from its sequential character in the former two cases, whereas the numerical identification in the "SHOT_NO" was always consistent. In addition the behaviour information given by the "LETTER" could easily be deduced from the "IN" and "OUT" locations. These facts indicated that no new additional information was provided by the "LETTER" attribute, which could therefore be excluded from the shot frame. The shot frame was given two additional slots, the last_hyp and the next_hyp. These were the "ACT_NO" of the last hypothesis modification made prior to the shot and the hypothesis modification immediately following it, respectively.

The hypothesis frame included the "ACT_NO", taken directly from the data, along with the list of the individual grid locations affected by the hypothesis modification and their changed contents. An additional slot was added, the last_shot, identifying the last shot taken in the game prior to the hypothesis modification. This additional slot as well as the ones in the shot frame pointing to the last and next hypothesis frames were created to preserve the patterns that occurred in the games for the creation of the rules.

4.3.2 Initial Rule Format

Once preliminary storage for the data extracted from the game files was designed, the next decision to be made was whether the knowledge that would result from analyzing the data should be represented in the form of rules, and, if so, which data elements would form which parts of the rules.

The chronological structure of the data in the game files and its functional attributes suggested that a rule format would be an appropriate means of attempting to capture these data characteristics. Thus, the antecedents and consequences of rules could be developed by following the progression of the games. The game data showed that players performed two types of actions in the game: hypothesis modifications and shots. To capture the decision process followed to make these actions

one rule type was designated for each action type.

Within each game the first action was always a shot and hypothesis modifications always followed shots. Therefore, rules to decide which hypothesis modifications should be made would use shots as their conditions and modifications as their actions. This type of rule, whose function was to analyze shots that had been made was called an "Hypothesis Rule". With the exception of the final act of the game and some instances where one hypothesis modification was followed by another, a hypothesis modification was almost always followed by a shot. Thus in the rules to decide which shots to fire, the conditions would be hypothesis modifications and the actions would be shots. This rule type that selected shots to be made was called a "Shot Rule". The situations in which one hypothesis modification was followed by another indicated that the player had exited the hypothesis mode of the game before having completed all the changes prompted by the last shot. In reading the game files such consecutive hypotheses would be combined to form one larger hypothesis, simplifying the rule building process.

In the game of Blackbox each action of the player may or may not contribute to formulating a solution to the problem. However, the chronological aspect of the data suggests a progressive building process, where each act is not only a precursor of the act immediately following it but can also contribute to all the acts that follow it in the game. Thus, with the exception of the first shot taken, no act in the game occurs independently. Therefore, in order to capture the entire decision process followed by the human players, the structures of the rules were expanded to include more conditions. Hypothesis Rules' conditions would encompass all the previous hypothesis modifications made in the game, in addition to the shots taken since the previous hypothesis modification, as well as the number of hidden balls remaining to be found. Shot selection rules would have all previous hypothesis modifications as their conditions, as well as the number of balls. In this way, each rule would provide a picture of the state of the grid as it is perceived by the player at the time the action takes place. It is interesting that the two primary data elements in the game, shots and hypothesis modifications, were each assigned different functions (conditions and actions) in the two types of rules extracted from the data. The two

Hypothesis Rule	
IF	C1: Shot_in and Shot_out {and Shot_in and Shot_out...} C2: Ball Number C3: {Previous Hypothesis State}
THEN	A1: Hypothesis Modification A2: {Reduce Ball Number}

Shot Rule	
IF	C1: {Previous Hypothesis State} C2: Ball Number
THEN	A1: Next Shot_in

Figure 4.3: Initial Rule Structure

rule structures are illustrated in Figure 4.3.

4.4 Selection of Examples

4.4.1 Grid Size

In our previous work on Blackbox, the game has usually been played on a 10 X 10 grid with six hidden balls. In this size, the game can provide a level of complexity sufficient to challenge most human players without overwhelming either them or the computer facilities with an excessively large playing surface. The 10 X 10 grid can be presented adequately on most computer monitors, allowing for the display of supplementary information that is required by the player, such as the menu of available options and the current game status indicated by the score and the number of balls to be located. The 10 X 10 grid size was also found to be convenient in

studying the distributed game of Blackbox. The distribution pattern selected there was to subdivide the 10 X 10 game into four separate 5 X 5 quadrants, each of which was to be played by a different agent. This gave each playing agent a non-trivial playing surface, and their performance could be compared against a single agent playing the same game configuration on the entire 10 X 10 grid.

Despite its advantages for playing Blackbox, the 10 X 10 size grid was deemed to be too large for automated knowledge acquisition, especially in the initial experimental stages. Each of the rule types planned for the knowledge representation had as a condition the current hypothesis state, which would comprise the accumulated hypothesis modifications made in the game. Individual hypothesis modifications made in 10 X 10 games could affect many grid locations. Combining large hypothesis modifications for each rule could become unwieldy, particularly in the early experiment trials that were used to learn about the knowledge acquisition process.

Having rejected the 10 X 10 grid for the task at hand, it was necessary to select a suitable size, small enough for the data to be easily manipulated and yet not so small that all complexity had been removed. At first, the 3 X 3 grid was considered, but was discarded as being too simple, offering very little possibilities for variety in approaching the problem. The 4 X 4 grid was small enough to be manageable but could offer some complexities, which would necessitate some planning in playing the game. This size was therefore selected for the initial knowledge acquisition sessions.

4.4.2 Number of Balls

Along with the size it was necessary to choose the number of balls that should be hidden in the grid. For the same reasons of economy and manageability of experimental data, it was decided to keep the number of balls to a minimal number. One ball hidden in a grid of any size offers no complexity. Two balls, on the other hand, can often be hidden in such a way that they affect each other's possibility of being located. The level of difficulty presented by two balls was judged to be sufficient for the knowledge acquisition purpose.

Thus the cases for testing the proposed automated knowledge acquisition pro-

gram would comprise 4 X 4 grids with two balls hidden in each. Such cases could yield some basic knowledge on how to play the game. This knowledge could be furthered by proceeding from the 4 X 4 game format to knowledge acquisition from 5 X 5 games, and then the knowledge would be generalized into rules that could be used to play any size game.

4.4.3 Training and Test Sets

In Section 2.4.1 it was explained that knowledge acquisition from cases involves the use of a *training set* and a *test set* of examples of domain problems. Often these example sets are selected from actual cases within the domain and are provided by workers in the domain. In order that the process should yield useful information, the examples selected must be representative of activities within the domain. The training set is used for the initial elicitation of knowledge, and the test set is used to apply the knowledge learned and ascertain its validity. If errors and omissions occur, the test set becomes a new training set and is used to augment the knowledge learned to date. This process can be repeated as often as is deemed necessary until sufficient knowledge has been obtained.

The Blackbox domain can produce a combinatorially large number of examples. The sub-domain selected for this experiment, consisting of 4 X 4 grids with two hidden balls, reduces the number of possible examples to $\binom{16}{2}$. That is, there are 120 possible grid configurations for two balls hidden in the 16 locations in a 4 X 4 grid. With 120 possible examples it would be feasible to study all of them to acquire the knowledge used to play them. If this were done, the training set would include 120 cases, leaving no examples to be used in the test set for validating the knowledge on similar cases. The case-based knowledge acquisition paradigm requires a training set for acquiring rules and a test set to validate the rules. In order to follow this standard, it was decided to employ some of the 120 possible games containing two balls in a 4 X 4 grid to form the experiment's training set and some of the games for the test set.

Blackbox can be played on pre-configured game grids or on configurations ran-

domly selected by a random number generator accessed by the game managing module. To ensure an unbiased selection of examples two subsets of 4 X 4 games with two balls were selected at random. Of the sets selected one contained 29 game configurations which were played by an expert to provide a data file to serve as the training set for the knowledge acquisition experiment. The other containing 28 games was designated as the test set. These sets will be described more fully in Section 6.1.1.

4.5 Data Analysis

4.5.1 Program Outline

After completing the preparation phase and the selection of the training and test sets the next step involved the actual extraction and analysis of the data presented in the game files. A program was written to perform the various tasks required in the acquisition of knowledge from cases of Blackbox games. As the program was run on the training set, it and the data structures it used were altered and refined.

The program's initial functions were to read in the game files included in the training set, isolate the different units of information, and store them in frames as described in Section 4.3.1. Once all the data from the games in the training set was transferred into the frame structures the program then used the frame data to construct the first-run Hypothesis and Shot Rules.

The next task required of the program was to remove redundancies from the first-run rules and attempt to minimize the number of rules resulting from the training cases. The rules that remained in the rule base after the removal of redundancies could then be analyzed further by another algorithm whose function it was to generalize the rules and hence their applicability.

4.5.2 First-Run Rule Formation

This phase of the knowledge acquisition was intended to produce rules that could be easily analyzed and manipulated in any subsequent processing that would be necessary. Thus the resulting rules should be straightforward, avoiding excess complexity. This was not a characteristic of the initial rule format, which was designed to portray the exact progression of events in the game. The preliminary goal was to translate the actions in the game into a progressive series of rules and then examine the patterns that emerged in the process. Once the reasons for these patterns were understood the rules could be simplified to reflect the analysis of events that had occurred.

The first test runs of the program produced rules in the format that described in Section 4.3.2 and depicted in Figure 4.3. These first rules appeared to contain complexities that could be eliminated without losing important information. Originally the first condition of the Hypothesis Rules was all the shots taken since the last hypothesis modification was made. This derived from the pattern of each Blackbox game having the format [s h s h . . s h], with s representing shot(s) taken and h representing a hypothesis modification. Assuming error-free hypothesizing (each time a hypothesis modification is made, the full implications of the previous shots have been assessed, the hypothesis modification is complete and the player does not reenter the hypothesis mode until another shot is made), each h can only have a length of 1. However, the minimum length of any λ_i (a series of s) is 1, and its maximum length in an n x n game is $4n - \sum_{j=1}^{i-1} \lambda_j$. Thus to record the variable length of λ occurring in any game, the first condition of the Hypothesis Rules could include one or more shots. This pattern in the game could be explained by the fact that the information yielded by several shots may be needed to make the decision reflected by the modification. Another possibility is that only the shot immediately preceding the modification is responsible for the action and the earlier shots may not be related to the modification. The earlier shots may not be resolved until later in the game when more information is available.

Since the reason for the pattern of several shots preceding one hypothesis mod-

ification was obscure, it was difficult to assess which part of the modification was caused by which shot and hence to manipulate the rules for the purposes of minimization and generalization. To simplify the rules for subsequent processing it was decided that each Hypothesis Rule elicited from the examples should have only one shot as its first condition, and, if no hypothesis modification immediately followed the shot, the action part of that rule would be empty.

The next problem with the rule structure was encountered in Condition 2 of the Hypothesis Rules and Condition 1 of the Shot Rules. Both conditions included a representation of the previous hypothesis state, that is the state of the hypothesis grid when the rule is applied. The state of the hypothesis grid at a given instant in the game is the accumulation of all the hypothesis modifications made in the game up to that point. As the game progresses, the hypothesis grid includes the grid locations affected by all the "ACT_NO"s in all the hypothesis modifications that occurred previously. Using the hypothesis grid as a condition in the rules can require a great deal of storage space and causes the rules to become unwieldy and difficult to manipulate. Since each hypothesis modification only occurs as the result of one or more shots, then the rules can be simplified by using shots to represent the hypothesis modifications that follow them. It must however be remembered that in the relevant conditions the shots are representatives of their respective hypothesis modifications. The rule format that was in effect after these revisions is illustrated in Figure 4.4.

Another problem was manifested in the last action of most games. After having located all the balls hidden in the grid, the expert human players modified all other previously unknown locations to empty. While this was a correct action, its effect on the automated rule acquisition and rule merging was to utilize more space in the processing and to result in excess rules whose actions would not be valid if taken in the middle of a game. By ignoring the setting of empty locations in the final hypothesis modification, this problem was solved. If the empty locations set in the final hypothesis modification resulted from the preceding shot, then ignoring those locations would cause the loss of some information related to the effects of that shot. However, the information could be regained if in other games such shots were not

Hypothesis Rule	
IF	C1: Shot_in and Shot_out C2: Ball Number C3: Previous Shot_in and Shot_out {and Previous Shot_in and Shot_out...}
THEN	A1: {Hypothesis Modification} A2: {Reduce Ball Number}

Shot Rule	
IF	C1: Previous Shot_in and Shot_out {and Previous Shot_in and Shot_out...} C2: Ball Number
THEN	A1: Next Shot_in

Figure 4.4: Revised Rule Format

the last ones in their games. A single rule unrelated to any specific shot would suffice to ensure that once all the hidden balls had been located the remaining unmodified squares of the grid could be emptied.

4.5.3 Redundancy, Subsumption, and Inconsistency

The first-run rules that were elicited from the Blackbox data exhibited certain anomalies that needed to be corrected. Some of the rules emerging from the data were similar to others. In the experiment similarities between rules were divided into two categories, which were entitled Explicit Redundancy and Implicit Redundancy. Implicit Redundancy will be explained later in this section.

Explicit Redundancy is a situation that occurs when similarities between two rules are "visually apparent". Explicit redundancies are manifested by similar conditions and similar actions. In the Blackbox rules two types of explicit redundancy occurred, which were designated Complete Redundancy and Partial Redundancy. Complete Redundancy is an explicit redundancy in which two Hypothesis or two Shot Rules have identical conditions and identical actions, taking symmetry into account. Partial Redundancy is an explicit redundancy of two Hypothesis Rules with identical first conditions, dissimilar third conditions, and some common actions. Complete and Partial Redundancies are illustrated in Examples 1 and 2.

Example 1

R1: IN - (0,1); OUT - (1,0) ==> EMPTY (1,1) - (1,2) - (2,1); BALL (2,2)

R2: IN - (1,0); OUT - (0,1) ==> EMPTY (1,1) - (1,2) - (2,1); BALL (2,2)

R1 and R2 form a Complete Redundancy.

Example 2

R3: IN - (0,1); OUT - (1,0) ==> EMPTY (1,1) - (2,1); BALL (2,2)

R4: IN - (1,0); OUT - (0,1) ==> EMPTY (1,1) - (1,2) - (2,1); BALL (2,2)

R3 and R4 form a Partial Redundancy.

The most obvious type of complete redundancy occurs in the situation when two rules (Hypothesis or Shot) have identical conditions and identical actions. It frequently happens that two Shot Rules have equivalent first conditions and identical

actions but their second condition, indicating the number of balls remaining to be located in the game, is not the same. Similarly in Hypothesis Rules there are situations when the actions of two rules are identical as are their conditions except for the second condition, which, like its Shot Rule counterpart, shows the number of balls remaining to be located. Such redundancies are also considered to be complete redundancies, and when they occur either in Shot Rules or in Hypothesis Rules, the second condition is determined to be unnecessary. When complete redundancy is detected, one of the rules is eliminated, and if Condition 2 was deemed unnecessary it is made void in the remaining rule.

Partial redundancy is an explicit redundancy that only occurs in Hypothesis Rules. Partial redundancy occurs in two different situations. In both, two rules are equivalent with respect to condition one but have dissimilarities in condition three. In addition to these characteristics, in one manifestation of partial redundancy the actions of the two rules are equivalent. In the second, the action of one rule represents a complete subset of the action of the other rule. These two types of partial redundancy are also known as "subsumption".

There is another situation in which two rules have equivalent condition 1 but their actions are neither identical nor is one a subset of the other. Rather the two action sets may have some identical elements in common. In such a situation partial redundancy may be present. This can be determined by the rule merging process designed to resolve partial redundancies that is discussed in the next section.

Implicit Redundancies occur as a result of the symmetry of the game. A visual inspection of a game grid with the trajectories of shots drawn through it shows a similarity between shots originating and ending at different locations. This can be seen in Figure 4.5 where shot A is a mirror reflection of shot B, and shot C would resemble shot D if the grid were rotated 90 degrees. In the figure, the O's represent Balls in the grid and the "bullets" trace out the trajectories. Hypothesis Rules elicited from such visually similar situations would have different absolute coordinates for the locations in their conditions and actions. However such visual similarities must be analyzed and means found to detect their occurrences in the

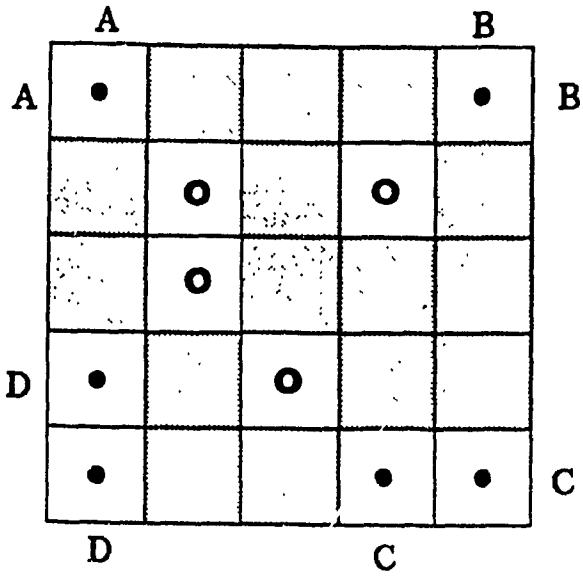

Figure 4.5: Implicit Redundancies

extracted rules. Such rule similarities were named Implicit Redundancies, explained by the fact that the likeness between the rules was not overt but would only become apparent after symmetries are taken into account. The approach used to detect and remove the Implicit Redundancies will be discussed in Section 4.5.7.

There are two broad categories of inconsistencies that can be detected in examining the elicited rules. The first category covers the situation of two rules that have different conditions leading to **equivalent actions**. In the discussion of complete redundancy differences of condition 2 were judged to be insignificant. Discrepancies that occur in condition 3 of two rules may prove upon analysis to be due to a partial redundancy. Inconsistency with respect to condition 1 may be caused by two different shots revealing the same information. Such a circumstance could be the result of an implicit redundancy between two Hypothesis Rules, a fact which does not necessarily indicate that either of the rules in question is incorrect. These implicit redundancies only demonstrate that sometimes more extensive analysis is required to arrive at the essence of a rule.

The second category of inconsistency occurs when two rules with equivalent conditions have **different actions**. This type of inconsistency in Hypothesis Rules can be more critical if analysis of the rules has shown that partial redundancy does

not apply. Such a situation implies a possible error, which can be determined by referring to the game frames related to the rules and ascertaining whether any errors occurred in the games being considered. If the inconsistency in question occurred in Hypothesis Rules it may be that later actions in one of the games reversed the parts of the actions that were inconsistent. If no errors occurred in the source games and if the actions were not reversed, then the rules are given a confidence factor less than 1 to indicate the fact that more than one action is possible in the circumstances.

Inconsistency of conditions or of actions in Shot Rules is not regarded as a serious discrepancy. Rather it illustrates that the game of Blackbox is not one of simple clear-cut solution paths but rather one in which solutions can be found using different approaches. In the thesis inconsistencies related to different conditions resulting in the same action were judged to be insignificant and were ignored. However, inconsistencies of Shot Rules with identical conditions leading to different actions were resolved using a "confidence factor" approach, which is described in Section 5.3.4.

4.5.4 Rule Merging

Rule merging was the method used to reduce the number of rules by correcting the partial redundancies that were characteristic of numerous Hypothesis Rules elicited from the Blackbox data. To identify how to correct partial redundancies, it was first necessary to understand why they occurred. In some circumstances two rules exhibited differences in Condition 3 (the previous shots conditions), but their other conditions and actions were equivalent. Such situations and the rule merging procedure used to resolve them are entitled Rule Merging Case 1 and are illustrated in Figure 4.6. In Blackbox a hypothesis modification is considered to be prompted by the behaviour exhibited by the current shot. Therefore, previous shot(s) in the game might be deemed irrelevant and any differences between two rules with respect to condition 3 could conceivably be eliminated by combining the two rules and completely omitting condition 3. However, sometimes previous shots in the same game are related to the grid area involved in the current shot and contribute to the analysis made by the player. It was therefore necessary to find a way to pre-

If	Rule A.C1 = Rule B.C1 and Rule A.C3 ≠ Rule B.C3 and Rule A.A1 = Rule B.A1
Then	Rule A and Rule B are replaced by Rule C such that Rule C.C1 = Rule A.C1 Rule C.A1 = Rule A.A1 Rule C.C3 = Rule A.C3 ∩ Rule B.C3

Figure 4.6: Rule Merging Case 1

serve potentially relevant previous shots and eliminate those that could be deemed irrelevant.

In this type of partial redundancy, the rule merging process follows the reasoning that any element in the condition common to both rules is potentially relevant and elements that the two rules do not share are not relevant to the current hypothesis. The two rules are replaced by a new rule whose condition 3 is the intersection of the previous shots of the two source rules, whereas the other conditions and the action of the new rule are unchanged.

When the partial redundancy is manifested by one rule's action set being a subset of the other rule's actions then one of the rules may have an incomplete action set. Figure 4.7 illustrates this type of partial redundancy and the rule merging process used to resolve it, known as Rule Merging Case 2. The incremental nature of Blackbox hypotheses are the root of such partial redundancies. Each hypothesis modification in Blackbox involves only the part of the grid whose contents may be changed as a result of the trajectory followed by the current shot. It often happens in a game that some grid locations affected by a particular shot have previously been modified due to another shot that had already occurred. These locations would not be included in the present hypothesis modification. In another game the current shot could be the first to affect the relevant grid squares, causing an action set that includes all the appropriate locations. Thus two identical shots with similar entry

If	Rule A.C1 = Rule B.C1 and Rule A.C3 $\neq$ Rule B.C3 and Rule A.A1 $\subset$ Rule B.A1
Then	Rule C is created such that Rule C.C1 = Rule A.C1 Rule C.C3 = Rule A.C3 $\cap$ Rule B.C3 Rule C.A1 = Rule A.A1 $\cup$ (Rule B.A1 $\cap$ Rule A.C3.A1)
If	Rule C.A1 = Rule B.A1
Then	Rule A and Rule B are replaced by Rule C
If	Rule C.A1 $\neq$ Rule B.A1
Then	Rule A is replaced by Rule C and Rule B remains

Figure 4.7: Rule Merging Case 2

and exit locations could cause one action whose elements are a subset of the other's.

This type of partial redundancy is resolved by creating a new rule from the original two. Condition 1 and condition 3 are treated following the method described in Case 1. The action of the new rule is produced in three steps. Assuming that rule A's actions are a subset of rule B's, first the actions of every rule in the rule base whose condition 1 is equivalent to one of the shots listed in rule A's condition 3 are combined. Then the intersection of this list with the elements in the action of Rule B is determined. The final step produces an action set that is the union of the action set of Rule A with the action set in the intersection. This becomes the action of the new rule. The action of the new rule is compared with that of rule B. If the two are equal then both Rule A and rule B are replaced by the new rule. If the actions are not equal, then the new rule replaces rule A and rule B remains in effect.

Another situation managed by rule merging is one in which the actions of two rules A and B are not equal and one is not a subset of the other. This can occur if both rules' action sets are incomplete due to the effect of previous shots and

If	Rule A.C1 = Rule B.C1 and Rule A.C3 ≠ Rule B.C3 and Rule A.A1 ≠ Rule B.A1
Then	Rule C is created such that Rule C.C1 = Rule A.C1 Rule C.C3 = Rule A.C3 ∩ Rule B.C3 Rule C.A1 = (Rule A.A1 ∩ Rule B.C3.A1) ∪ (Rule B.A1 ∩ Rule A.C3.A1)
If	Rule C.A1 = Rule A.A1
Then	Rule A is replaced by Rule C
If	Rule C.A1 = Rule B.A1
Then	Rule B is replaced by Rule C
If	Rule C.A1 = 0
Then	Rule A and Rule B are not partially redundant and eliminate Rule C
If	Rule C.A1 ≠ Rule A.A1 and Rule C.A1 ≠ Rule B.A1
Then	Rule A and Rule B are inconsistent and Rule A and Rule B remain unchanged and Rule C is added to the rule base

Figure 4.8: Rule Merging Case 3

their respective hypotheses. Another possibility is that the hypothesis modifications recorded in one or both of the rules contained changes resulting from previous shots but only ascertained after the current shot. This situation and its corresponding rule merging procedure, entitled Rule Merging Case 3, is shown in Figure 4.8.

Whether the situation indicates a partial redundancy is determined after the creation of a new rule from rules A and B. Conditions 1 and 3 of this new rule are created in the same method used in the previous cases. The creation of the action of the new rule involves several steps. Rules with condition 1 equivalent to

an element of rule A's condition 3 (previous shots) are identified and their actions are combined and intersected with the elements in the action of Rule B. The same steps are followed with Rule B's condition 3 and Rule A's actions. The union of the actions in the two intersections becomes the action of the new rule.

An examination of the newly created rule will indicate if the rules were partially redundant. Partial redundancy is assumed to have occurred if the new rule's action is not empty. If the new rule's action is equivalent to the action of either of the original rules, the original rule with the equivalent action is replaced by the new rule. Otherwise, the new rule is added to the rule base. Partial redundancy did not occur if the new rule's action field is empty, in which case the new rule is discarded.

4.5.5 Completeness and Minimality

The following concepts are used in this thesis. A *minimal* rule base for a given game problem can be defined as a non-redundant set of rules with which the expert system can solve the game. A *complete* rule base can be defined as a set of rules with which the expert system will not only solve the game but will also achieve a specified level of performance.

The goal in developing expert systems is to construct a complete rule base containing rules that pertain to all facets of domain problems. At the same time, it might be considered desirable to minimize the number of required rules. The benefits of minimization include saving memory, reducing the time required to load an uncompiled rule base into the expert system, and minimizing the cost of maintaining and expanding the rule base. Thus, in building a rule base it is necessary to strive to maintain a balance between completeness and minimality, taking care that in achieving one attribute the other is not neglected. To do this some measurement must be devised to assess the completeness and minimality of a rule base. Before trying to define a measure for completeness and minimality, which can be a difficult task, it is advisable to examine what is meant by these characteristics.

In its broadest sense, completeness is defined as logical completeness. To be logically complete a rule base must include one rule for each combination of the

conditions needed to assert hypotheses. This means that for a rule base with n binary conditions logical completeness entails that there should be 2^n rules. If logical completeness was required in all rule bases it could be measured easily. However, in many problem areas, logical completeness could result in including rules with combinations of conditions that could never occur in the domain, and the rule base would have superfluous rules that would never fire in an expert system. Therefore this notion of completeness is not useful for us, and completeness of a rule base is defined as the inclusion of all rules needed to solve the subset Q of problems in the domain that are targeted by the expert system. That is, each rule in the rule base must be representative of a situation that can actually occur in the domain. The verification of the applicability of each rule in the rule base can be accomplished relatively easily especially when the rules are all acquired from actual domain cases. However it remains difficult to know when **all** the necessary rules have been included, making the rule base complete. Unless all potential domain problems can be analyzed during the building of the rule base, completeness must be achieved in an incremental manner. One way this can be done is to have a dynamically changing rule base that can be augmented whenever a problem that cannot be solved is encountered.

In theory it is possible to assess the minimality of a rule base by repeatedly adding and removing rules. However unless a logically complete set of rules is initially available from which one could select rules for this type of testing, minimality remains difficult to quantify and measure. Nevertheless during the building of the rule base it is possible to strive for minimality by avoiding anomalies that can result from the knowledge acquisition process. For example redundancy and subsumption are not infrequent by-products of knowledge acquisition. If not detected and eliminated, their presence would detract from the desired minimality of the rule base. Uncorrected inconsistencies would also adversely affect the size and efficiency of the rule base.

In the extraction of rules from Blackbox game files, some of the irregularities affecting minimality were encountered. Their properties and the procedures used to eliminate explicit redundancies, subsumption, and inconsistencies were discussed in

previous sections. It can be argued that it is theoretically possible that an attempt to ensure minimality would conflict with an attempt to achieve completeness. Our extensive experience with Blackbox showed that this would not occur in case-based knowledge acquisition. Completeness as it affects minimal size was not an issue in the Blackbox case-based knowledge acquisition study. All the rules acquired were derived from actual case files, hence rules whose sole purpose was to ensure logical completeness were not created. The rules that resulted from the knowledge acquisition reflected actual decisions that were made and were therefore deemed necessary for the rule base, unless they were redundant. The process followed was not expected to succeed fully in achieving completeness and minimality. However, it did ensure that the rule base that resulted from the knowledge acquisition would include all the rules that emerged from the cases examined that did not display unresolved redundancies or inconsistencies.

4.5.6 Minimal Blackbox Rule Base

If the expert system domain is not too complex, it may be possible to categorize and enumerate the different types of situations that would trigger the firing of rules. If this were the case it may be possible to define a number to represent minimality for that domain's rule base. To begin defining a minimal number for Hypothesis Rules in the Blackbox rule base a study of the types of shots that can occur in the game is required. The shots taken in a Blackbox game can be categorized as **corner shots** and **edge shots**, according to their point of entry or exit. In a grid of size n, a shot whose entry location's x or y coordinate is 1 or n is termed a **corner shot**. A shot whose entry location's coordinates are neither 1 nor n is an **edge shot**. There are 8 corner positions and $4(n-2)$ edge positions in a game grid. In order to cover all possibilities, one might think that a rule would be needed for a shot originating in every one of the locations surrounding the grid, or $4n$ rules. However, the symmetry of the game suggests that, given some modifications, a set of rules comprising one rule for each type of shot entry/exit pair might be sufficient to serve every shot taken in the game.

The next step in defining minimality is to identify how many different types of shot entry/exit pairs are needed to classify all the shots that can take place in a game of Blackbox. Each shot into the grid exhibits one of three behaviours. A shot may be absorbed by a hidden ball that it hits; it may be reflected out of the grid through its point of entry; or it may exit the grid through some other location. The number of exits possible for each type of entry determines the number of possible entry/exit pairs that can be used to describe any given shot. Each **corner shot** can exit the grid through n different locations, excluding reflections, and therefore a **corner shot** can be represented as n different exit/entry pairs. Each **edge shot** can exit the grid through $4n-6$ different locations, two of which are corner locations, which would be accounted for under the **corner shot** heading, and one of which is the reflection, exiting at the entry point. Therefore, each **edge shot** can be represented as $4n-9$ different entry/exit pairs, when the corner shots and the reflection are not counted. Absorption and reflection (of any type of shot) account for two entry/exit pairs. Totalling the number of entry/exit pairs for one corner shot and one edge shot results in $5n-7$, i.e. $(4n-9+n+2)$. This would be the number of Hypothesis Rules that a Blackbox rule base would need if the necessary modifications to handle symmetrical entry/exit pairs could be incorporated into the rules.

This number, however, does not cover all the possible actions that could be taken as a result of the various shot types. For instance, when a shot results in a hit there are at least n possible conclusions that could be drawn from it, only one of which would be correct for a given game. In addition, shots that exit the game grid will do so after traversing a trajectory that may deflect them 0 or more times, depending on the game configuration. Thus seemingly identical shots occurring in different games could be indicative of different trajectories and hence different hypothesis modifications. Given these arguments, $5n-7$ is the least number of rules that will serve a given game grid under some conditions but it may not be adequate to actually qualify for a minimal or complete rule base.

Attempting to define a minimal number of Shot Rules in the Blackbox rule base presents a greater difficulty. There are $4n$ possible shots that can be taken in a game. This is therefore the number of actions that could possibly be taken as a

result of the firing of Shot Rules. However the conditions leading to these actions are varied and numerous. Since shots are selected on the basis of previous shots that occurred in the game, and since the order in which shots are taken depend on the playing style of individual players, it is difficult to predict the number of Shot Rules that could result from the case-based knowledge acquisition. In addition the combinations of shots that are taken in games of Blackbox would vary based on the trajectory of different shots and would represent an indeterminate number. Thus it is difficult to derive a number to approximate minimality and completeness for Shot Rules in Blackbox.

4.5.7 Minimizing the Blackbox Rule Base

After complete redundancies were removed and rule merging was employed to adjust partial redundancies and remove subsumption, the Hypothesis section of the rule base consisted of a set of rules whose number in one instance had been reduced from 125 rules in the first run rule base to 55 rules. However the minimization task was not yet completed. The rule base still contained implicit redundancies which had not been detected or removed by the automatic reduction methods employed. Therefore, the next step in improving the rule base was to addre · the issue of implicit redundancies and further minimize the rule base. In addition, the rules in the rule base had up to three conditions leading to a set of actions. Another goal at this time was to transform these rules from multiple condition rules to simple Horn clauses (one condition and an action set for each rule).

According to the structure of the Hypothesis Rules in the rule base, each had as its primary condition a specific shot entering and exiting the grid at particular locations. Thus each rule could fire for one and only one explicitly defined shot rather than for a class or type of shot. This was thought to be the means by which the implicit redundancies caused by the symmetrical composition of the game were introduced into the rules. It was necessary to isolate the shot attributes that contribute to the visual similarities such as those discerned in Figure 4.5 and to devise means of quantifying and computing those attributes. Once computable

characteristics to identify and group shots in relationship classes were isolated an algorithm could be developed to remove implicit redundancies from the rule base.

Shots were initially identified by their entry and exit locations. However, since such definite entry and exit locations were suspected to contribute to implicit redundancies, another manner of identifying shots was required. One method of shot identification was to calculate the distance between a shot's entry and exit locations in the grid. Visual inspection of the perceived similarities indicated that similar shots had the same distance (measured in grid squares) between their entries and their exits. If two shots could be shown to have an identical distance between their respective entries and exits, this similarity might be indicative of other similarities and the Hypothesis Rules related to the two shots could be combined.

Shot entry-to-exit distances were computed by subtracting a shot's exit coordinates from its entry coordinates. In order to ensure that the distance computation was done methodically and consistently, it was necessary to a adopt a labelling scheme universally applicable to all shots. The following example illustrates the need for a uniform labelling scheme. A shot entering the grid at location (0,1) and exiting through (2,0) would have a distance of (-2,1) and another shot entering the grid at (2,0) and exiting through (0,1) would have a distance of (2,-1), suggesting different shot behaviours. However since both shots follow the same trajectory, the above conclusion is erroneous. To prevent such errors from occurring, it was decided to incorporate the fact that in Blackbox the entry and exit of a shot can be interchanged without changing the effect of the shot. Thus, the labelling adopted required that a shot's entry would have a numerical representation that is greater than or equal to the numerical representation of the exit. The numerical representation of a grid location is the juxtaposition of its x and y coordinates. That is, a location of (0,1) is represented by the number 1 and a location of (2,0) by the number 20. If a shot enters the grid at (2,0) and exits at (0,1) its label is 2001, juxtaposing the numerical representation of the entry and the exit in that order. The same label of 2001 is used for a shot that enters the grid at (0,1) and exits at (2,0), since the entry and exit are interchanged before labelling the shot. The entry-to-exit distance attribute of both shots is 2,-1. This methodical distance calculation from

shot labels also helps to identify other rules whose similarities can only be detected from their identical distance attribute. For example, in a 4 X 4 grid a shot labelled 5435 would have a 2,-1 distance attribute, and could therefore be handled by the same Hypothesis Rule as that for 2001.

The distance attribute only addressed similarities in the first condition of the rules extracted from the data. In order to be able to use this to actually reduce the number of rules, the other conditions included in the rules also had to be considered. Inspection of the rules that emerged from the redundancy removal procedures showed that many did not include Condition 2 (the number of balls remaining to be located) since very often that condition was deemed to be irrelevant to individual rules and was discarded. Similarly the rule merging procedures often eliminated some if not all of Condition 3 in treating partially redundant rules. In rule merging only the portions of Condition 3 that were common to both rules under consideration were maintained in the merged rule. In addition, rule merging used the elements of Condition 3 as pointers to other rules whose actions could be used in combining the actions of the similar rules. Thus the significant information that was provided by Condition 3 had already been extracted by the rule merging procedure and was reflected in the merged rules. This prompted the decision to eliminate Condition 2 and Condition 3 from all Hypothesis Rules, after the redundancy check and rule merging had been applied. This would give the Hypothesis Rules a Horn clause structure, fulfilling one of the goals for the rule base, without diminishing the effectiveness of the rules.

Once the rules were restructured as simple Horn clauses it was easier to manipulate them by using the distance attribute as a measure of symmetrical similarity and combining rules based on these similarities. First each Hypothesis Rule was labelled using the labelling protocol described above, and the entry-to-exit distance was computed. The distance was used to replace the entry/exit pair of the original Condition 1 in the rules.

Since the rules were now concerned with relatively situated shots, rather than with specific entries and exits, the action component of the rules needed to reflect

this changed approach to location addresses. The actions of Hypothesis Rules, drawn from the hypothesis modifications in the games, previously included the new contents of some grid locations along with the location's specific address. In the new version of the rules specifically named locations were replaced by locations relative to the shot entry. Thus, regardless of the actual entry and exit of a shot, as long as its entry-to-exit distance was that in the rule's Condition 1, the rule could be fired on the shot's occurrence and the actions would be applicable.

In this way the implicit redundancies in the Hypothesis Rules could be removed. Using the new rule structure, rules with the same Condition 1 (entry-to-exit distance) were now combined into one rule, forming a union of their actions and removing excess identical action elements.

4.6 Reduction Results

The automated knowledge acquisition was initially applied to the *training set* stored in the game file described in Section 4.4.3. The game file consisted of 29 games, 27 of which were different from one another. The total number of shots taken in all 29 games was 125 shots. The first phase of the automated knowledge acquisition created one rule for each shot taken. Hence 125 Hypothesis Rules and 125 Shot Rules were constructed from the first pass on the game file. The Shot Rules were then processed to remove redundancies automatically. This procedure reduced their number to 62 Shot Rules.

Refinement of the Hypothesis Rules section of the rule base was done in three stages. First all complete redundancies were removed reducing the number of Hypothesis Rules to 81. In the next stage rule merging was employed to correct partial redundancies and eliminate instances of subsumption, which resulted in another reduction, down to 55 Hypothesis Rules. The final step involved applying the minimization algorithm to remove implicit redundancies and produce more general rules. This step produced the greatest reduction of all, resulting in 19 Hypothesis Rules.

The rule base that emerged from these automated knowledge acquisition proce-

dures was now ready to install in an expert system and used to play Blackbox. The Blackbox Expert System and the installed rule base will be described in the next chapter. The results of the testing that ensued will be presented in Chapter 6.

Chapter 5

THE PROTOTYPE EXPERT SYSTEM

5.1 Outline

Following the extraction of knowledge from Blackbox game files, the acquired rule base was installed in an expert system and tested. This chapter will describe the rule base and the expert system used for evaluating the rule base.

5.2 The Expert System

5.2.1 Its Components

The Blackbox playing expert system uses the Blackbox Expert Shell, which is made up of several components, implemented as objects in C++. Some of these elements are standard features in many expert system shells and others are unique to the Blackbox Expert Shell. The shell comprises a user interface, the expert manager, the Blackbox game manager, and a working memory. These are combined with an inference engine and a rule base to constitute the Blackbox expert system.

The Blackbox Expert Shell was designed and written by John Lyons in C++ for the purposes of conducting experiments on Blackbox and Distributed Blackbox [17]. The Blackbox Expert System utilizes the CLIPS inference engine which is embedded within the Blackbox Expert Shell [3]. The inference engine controls the

overall execution of the expert system, deciding which rules should be fired to solve the Blackbox game. The forward-chaining inference engine has a Lisp-like syntax based on the OPS5 language.

5.2.2 The User Interface

The Blackbox Expert System works almost completely independently of the user. The user need only provide the configurations of the games to be solved by the system. All the information required by the system to solve the games is provided by the various constituent parts of the system itself. The user interface object controls the interaction between the user and the system within four windows displayed on the monitor. A diagram of these windows is shown in Figure 5.1. As described below, each of the windows has a specific function to perform in communicating with the user.

- **Grid Contents:** While a game is being played by the expert system its actual configuration is shown in this window.
- **Current Knowledge:** This window displays what is currently known by the expert system about the present game. The game grid is shown as are the shots that have been fired in the game along with any hypotheses that have been made by the expert.
- **Dialog:** All commands entered by the user are echoed in the Dialog window. Any messages from the user interface to the user are also shown here.
- **Commands:** The user may communicate with the system using a set of commands which are listed in this window along with their requisite arguments.

The version of the Blackbox Expert System used for this experiment only had access to some of the commands listed in the Command window. The other commands which had not yet been implemented were not required to achieve the goals of this experiment. The available commands and their functions are shown in Table 5.1.

The representation used for the contents of the individual grid locations in the Grid Contents and Current Knowledge windows and for the labels given in the Current Knowledge window to shots fired by the expert is explained in Figure 5.2. The labelling format used is the same as that explained in Section 3.1.1.

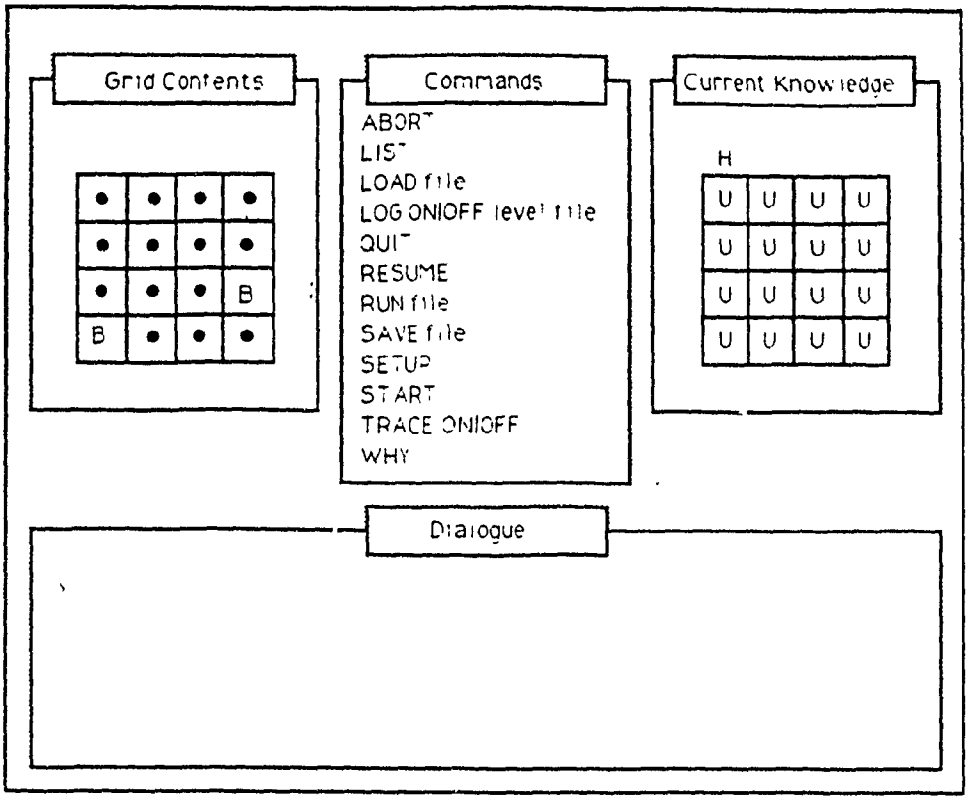

Figure 5.1: Blackbox Expert User Interface

Grid Contents		Shot Labels	
U	Unknown	**h**	Absorbed Shot
B	Ball	**r**	Reflected Shot
•	Empty	**A..Z**	Other Shots
P	Possible Ball		

Figure 5.2: Grid Representation Scheme

COMMAND	ARGUMENT	ACTION PROMPTED
LOAD	file name	load a Blackbox game configuration
ABORT		abort the current game
RESUME		fire another shot into the grid
QUIT		shut down the blackbox game playing system in an orderly fashion
TRACE	ON or OFF	enter or exit trace mode - if the argument ON is used, after each shot the system waits for the user to enter RESUME command before continuing the session
START		initialize the objects necessary to begin playing a new game and fire the first shot.

Table 5.1: User Commands and their Functions

5.2.3 The Working Memory

The working memory object manages the storage of data used by the expert while the game is being played. The information stored by the working memory includes the up-to-date hypothesis made by the expert about the grid's contents, the number of balls that have been located by the expert, the locations of those balls, a list of shots whose trajectories are known, a list of shots whose trajectories have yet to be analyzed with certainty, and a list of locations from which shots could still be fired. In addition to this dynamically changing data, the working memory also knows the grid's size and the number of balls hidden in the grid. The working memory manages its dynamic data structures in response to instructions received from the expert and provides information needed by the user interface to inform the user about the current status of the game.

5.2.4 The Blackbox Game Manager

The Blackbox game manager is the module in the expert system that accepts the shot input and returns the exit. The game manager knows the domain rules of the Blackbox game and maintains the information about the actual configuration of the grid. It uses this knowledge to determine the trajectory of shots fired by the expert and inform the expert of their outcome.

5.2.5 The Expert Manager

The expert manager's mission is to coordinate the activities needed to solve the game such as selecting which shot to fire next, analyzing the fired shots in order to make hypotheses about the contents of the grid, and determining when the playing should be halted. The performance of these tasks requires communication with and coordination between the various components of the system. The expert manager provides this function. It creates the working memory at the start of a new game session. During the game session the expert manager interacts with the working memory to acquire the information required for the expert's own functions and to

inform the working memory about changes to the dynamic data structures that will be needed by the user interface. The information exchange with the working memory involves the properties of the last shot fired, the number of balls found, and changes to the hypothesis grid. The expert manager uses this information to determine its own agenda and to set up the fact list required for rule execution. During the course of the game, the expert manager also communicates with the blackbox game manager, instructing it where to fire shots and receiving information about their outcome.

5.3 The Expert System Rule Base

The Blackbox Expert System rule base is written as an ASCII text file conforming to CLIPS syntax. In CLIPS syntax each rule is identified by the CLIPS keyword "defrule" and a unique name and comprises a collection of conditions and actions. The rule's conditions are patterns that must match facts currently known by the expert system about the state of its world. The facts that the expert system reasons about form a dynamic data base. Facts are asserted and retracted by the expert system when events occur that change the current status of the game. When all the conditions of a rule have been satisfied its actions can be performed.

The Blackbox Expert System rule base is divided into two rule types, Selection Rules and Analysis Rules, corresponding to the Shot Rules and Hypothesis Rules derived from the two types of actions identified in the case-based knowledge acquisition process. The knowledge acquisition process automatically generates an initial set of rules. In addition to these automatically derived rules, the rule base contains some Housekeeping and Meta-Level Rules written to help maintain the regular flow of rule application and the appropriate development of the game. Some are used to prepare the inference engine for rule management at the beginning of each game session; others are used to manage the fact list between rule firings; and still others are used to control the end game, judging when a game has been solved satisfactorily, which leads to termination. The following sections will describe the Selection and Analysis Rules and how they evolved from the Shot and Hypothesis Rules described

```
(defrule SR1001b        ;ascore 6.000(2)        (declare (salience 1))
      (phase selection)
      (OLDSHOT ? 1 0 0 1)
      ?sl <- (SHOTLEFT 0 4)
=>
      (setnextshot 0 1)
      (retract ?sl)
 )
```

Figure 5.3: Sample Selection Rule

in Chapter 4, and the derivation and creation of the Housekeeping and Meta-Level Rules.

5.3.1 Selection Rules

Figure 5.3 shows a CLIPS Selection Rule as it appears in the latest version of the rule base used in the experiment. Except for small modifications, shown in bold type, this rule was derived directly from the revised rule format for Shot Rules as seen in Figure 4.3. The rule's name combines the representations of the previous shot's entry and exit taken from condition 1 of the Shot Rule. The name follows the protocol that the numerical representation of the entry coordinates must be greater than or equal to that of the exit coordinates. The condition in the CLIPS rule matching an OLDSHOT with entry and exit coordinates is a formal statement of the rule's first condition. If the rule's first condition included more than one previous shot, each shot would have a corresponding OLDSHOT condition and be included in the rule's name. The first action statement in the CLIPS rule causes the expert system to call procedures to fire the next shot in the game from the location identified in the action in the Shot Rule format of Figure 4.3. According to the minimization procedures followed in Section 4.5.7, Condition 2 of the Shot Rules was deemed unnecessary and was therefore not included in the CLIPS Selection Rule.

The CLIPS Selection Rule contains statements that are not included in the Initial or Revised Shot Rule formats, although some are derived from the information available either in the rules themselves or in the Shot frame created from the Blackbox game files. The first is the comment that follows the rule's identification statement indicating that the average score (ascore) achieved in the two (2) games from which this rule was extracted was 6.00. The average score is used to resolve apparent inconsistencies in Selection Rules. It contributes to determining the degree of the "salience" declared in the immediately following statement. The salience indicates the level of priority assigned to the rule. If all the other conditions of two or more rules are identical then the higher salience indicates which of the rules is to fire first.

The rule's type is identified in its phase condition (see Figure 5.3). The last condition in the rule states that the location that will be selected by the rule's action as the entry for the next shot has not been used previously in the game. The final action statement in the CLIPS Selection Rule is an instruction to remove facts that were asserted by the firing of the rule.

5.3.2 Analysis Rules

The Hypothesis Rules shown in Figure 4.3 were modified by the rule merging and minimization procedures described in Sections 4.5.7 and 4.5.4. A sample of the most recent version of Analysis Rules that were generated for the CLIPS rule base after these modifications is shown in Figure 5.4. Much of the rule as it appears in the figure was derived from the automatically acquired Hypothesis Rules. The parts of the rule that were not derived from the Hypothesis Rules are shown in bold type in the figure.

The name identifying the Analysis Rule is obtained from the entry-to-exit distance attribute calculated from the shot coordinates in the first condition of the Hypothesis Rule. Also derived from the Hypothesis Rule is the condition that determines the shot type, comprising the shot's behaviour pattern - deflect, reflect, or hit, and its entry-to-exit distance parameters. The shot's behaviour pattern is revealed by its distance parameters. Since a reflected shot enters and exits the grid

```
(defrule S1-3
      (phase analysis)
      (GRIDSIZE ?n)
      ?st <- (SHOTTYPE DEFLECT 1 -3)
      ?ls <- (LASTSHOT ?SN ?IR ?IC ?OR ?OC)
      (not (CERTAINSHOT ?SN))
=>
      (setstatus (- ?IR 1) (- ?IC 0) EMPTY)
      (setstatus (- ?OR 0) (- ?OC 3) EMPTY)
      (setstatus (- ?IR 2) (- ?IC 0) EMPTY)
      (setstatus (- ?OR 1) (- ?OC 3) EMPTY)
      (setstatus (- ?IR 2) (- ?IC 1) BALL)
      (setstatus (- ?OR 1) (- ?OC 4) BALL)
      (assert (SETBALL =(gensym)))
      (makeshotcertain ?SN)
      (assert (CERTAINSHOT ?SN))
      (retract ?st ?ls)
)
```

Figure 5.4: Sample Analysis Rule

at the same location, its distance is "0 0". Since an absorbed shot does not exit the grid, the game managing module assigns it a default exit of "0 0", thus making its distance equal to its entry coordinates. All other shots whose distance parameters do not conform to the previous two criteria are deflected shots.

The remaining statements in the conditions section of the Analysis Rule were not derived directly from the Hypothesis Rules. Instead they were included to acquire dynamic information required to carry out the rule's actions and to ensure the efficient firing of rules. The phase condition statement is used to identify the rule's type. In some rules the distance parameters are a function of the game's gridsize, e.g. a distance of one greater than the gridsize is given as $(n + 1)$. In order to interpret this information, the gridsize statement gets the size of the grid from the expert system. There is also a condition that obtains information about the last shot - its shot number and entry and exit locations. The shot number is used in another condition to ascertain that during the current game this shot has not been declared a certain shot, that is, one that had already been evaluated completely. This is a precaution taken to ensure that hypothesis modifications are not applied repeatedly as a result of the same shot. The shot's entry and exit are used to translate relative grid addresses to specific locations in the current game when the rule's actions are applied.

The first actions in the CLIPS Analysis Rule use the command "setstatus" to instruct the expert system to make modifications to the game's solution hypothesis. These actions are derived from the elements in the first action in the Hypothesis Rule. As described in Section 4.5.7, after the specific grid addresses named in the rules' conditions had been replaced by the entry-to-exit distance parameters, the addresses included in the individual elements of the rules' actions were revised to reflect this changed approach. The "setstatus" actions are the formal end-product of these revisions to the rules' actions. Hence the individual actions in Figure 5.4 specify that the status of a location identified by its distance from the last shot's entry coordinates and exit coordinates be modified to "EMPTY" or "BALL". In some Analysis Rules there are actions setting the status of a location to a "POSSIBLE BALL". The process used to decide whether to modify a location to include a

"BALL" or a "POSSIBLE BALL" will be discussed in Section 5.3.4.

The action immediately following the "setstatus" actions in the Analysis Rule is an adaptation of the second action in the Hypothesis Rules which reduced the number of balls remaining to be found in the current game. The action calls the CLIPS "gensym" function to increase the number of grid locations that had been marked with balls and instructs the expert system to assert a fact with the increased number. This last action is only included in rules that had a "setstatus ..... BALL" as part of their actions.

The remaining actions in the rule are not found in the Hypothesis Rules. Like the added statements in the condition portion of the rule they were included to help the flow of the rule application. The next two actions cause the expert manager to add the shot that has just been reasoned about to the certain shots list, and assert that fact in the fact list. As is the case in the Selection Rule, the final action statement in the Analysis rule is used to instruct the expert system to remove some facts that were asserted by the firing of the rule.

5.3.3 Housekeeping and Meta-Level Rules

The rules described in the previous two sections all contain some information that has its source in the knowledge acquired either automatically or by adaptation from actions in the game files. In addition to these rules there are others that are needed for the efficient functioning of the Blackbox expert system. These are the Housekeeping Rules and the Meta-Level Rules. The Housekeeping Rules have been designated as Selection and Analysis Rules to indicate in which phase of the game they are applied. The Meta-Level Rules are all fired during the analysis phase and are therefore included among the Analysis Rules.

The Housekeeping Rule applicable in the Selection phase is a default rule that fires whenever the expert system needs to know which shot to fire and none of the selection rules in the rule base can apply to the current state of the game. In such a case, the default rule selects the next shot from the list of available entry locations maintained by the working memory.

Among the Housekeeping Rules included with the Analysis Rules one is a set-up rule used to initialize the facts regarding the number of certain balls and the number of possible balls found by the expert in the game. In addition, there are Housekeeping Rules that are used to compensate for the incompleteness of the Analysis section of the prototype rule base. If no Analysis rule can be applied to analyze a current shot, facts that were added to the fact list when the shot was fired are not used or retracted. It is therefore necessary to include rules whose sole function is to clear the fact list.

The Meta-Level Rules are rules that fire as side effects of the actions of other rules. One such rule updates the number of balls placed by other Analysis Rules during the progress of the game. Other Meta-Level Rules are used to infer information from the status of the hypothesis grid. There is a rule that determines if the number of balls hidden is equal to the number of balls that have been found with certainty, in which case the remaining locations in the grid can be marked empty. Other Meta-Level Rules are used to reason about possible balls and when deemed appropriate convert them to definite balls.

Some of the Housekeeping and Meta-Level Rules were prompted by the properties of the CLIPS inference engine. Other rules were written as a result of requirements perceived in the writing of the Blackbox Expert System. And still others were written in response to some shortcomings viewed in the performance of the acquired rules upon preliminary testing. These will be described in the following sections.

5.3.4 Reasoning with Priorities and Certainty in Blackbox

In the Blackbox knowledge acquisition experiments, priorities and certainty factors were the means used to deal with and rectify inconsistencies that became apparent as the rules elicited to play the game were analyzed. Both Selection and Analysis Rules were affected by disparities that needed adjustment.

Inconsistencies arose in the actions of some Selection Rules of the Blackbox rule base. Given the same previous shots, different shot selection decisions were found to have been made in the game files used for knowledge acquisition. This apparent

inconsistency was not too disconcerting. Like many games, the rules in Blackbox do not prescribe a fixed pattern of play. Therefore, different human players will have individual styles of playing the game, and one player may use disparate strategies in various games with seemingly similar scenarios. Unlike human problem solvers, computers are always algorithmic. If an expert system is given a specific rule to follow at a given time, it will do so unfailingly, and, if it is given several rules that can apply in the same situation, depending on its method of firing rules, it will either apply the first it sees and ignore the others or it will require a means of prioritizing these rules in order to fire the best one first. Therefore it was necessary to reconcile the differences in strategy that appeared in the rules acquired from the game files, and determine which of the conflicting rules contributed to a more successful playing of the game.

As redundant Selection rules were identified, the average of the scores achieved in the games from which these rules were acquired was calculated and stored as part of the rule that remained. When the rules were ready for installation in the expert system, the average score achieved using a rule with one action was compared with the average score achieved using another rule that had the same conditions but a different action. The rule with the lowest average score (indicating the best performance in Blackbox) was then awarded the highest priority. Since the knowledge acquisition process is incremental and "everlasting", the rules with the lower priorities were retained for future use, although, given the nature of the game, they would not be fired until the rule base changes. As more knowledge is acquired in future sessions or if the knowledge is refined by a human expert, these rules' priorities may improve, possibly surpassing that of the current best rule.

In the Analysis Rules, adjustment was needed when there were inconsistencies that were not resolved when rule merging was applied to rules with a similar first condition. In the process of detecting redundancies and reconciling similarities, each rule that remained in the rule base was given a "redundancy" count, a "similarity" count, and an "exception" count. Whenever a rule was found to be redundant, the redundancy count of the remaining rule was incremented by 1. When rule merging was applied to similar rules and only one rule remained, the similarity count of

the remaining rule was incremented to confirm the similarity or partial redundancy. The redundancy count and the similarity count both contribute to a higher certainty factor for a rule, since its correctness is assumed to be confirmed by succeeding rules that emulate it.

If after rule merging was applied to two similar rules, one rule was found to have an action whose elements were a complete subset of the action of another, the first rule was removed and the "exception" count of the second rule (with the action superset) was incremented. Likewise, when two similar rules' action sets were not perfect subsets of one another's, both their exception counts were increased. The exception count could be equal to 0 (for rules whose actions did not disagree with any other similar rules), or it could be 2 or higher. The exception count reduces confidence in the rule since it is indicative of dissimilar actions of rules with identical conditions.

The next steps in assigning a certainty factor to the rules occurred as the rules underwent the minimization process to remove implicit redundancies. First an interim certainty factor was assigned to the individual action elements in each rule. This interim factor was called the action's "frequency" rate and was the reciprocal of the rule's exception count, unless the exception count was 0, in which case the frequency rate of the action elements would be 0. In this way the frequency rate was always either 0 or less than 1. The individual action elements also inherited the redundancy and similarity counts of their rules.

The calculation of the certainty factor was continued while combining the actions of **implicitly redundant rules**. In this process, a count was kept of the number of rules that had identical action elements. Whenever an action element was found to occur in one more rule, its frequency rate was increased by the frequency rate of its counterpart in the other rule. The action element's redundancy and similarity counts were similarly accumulated at this stage. At the end of the process, each action element was assigned a certainty factor equal to the quotient of its frequency rate divided by the number of rules contributing this element. If either the action element's redundancy count or its similarity count were greater than its certainty

factor, the certainty factor was assigned a value of 1, to reflect the fact that there was more than one rule with the same action element, thus giving it a higher certainty.

The certainty factor calculated in this fashion was used to determine the degree of confidence that could be placed in an action element that marked a grid location with a "ball". Thus, if such an element had a certainty factor greater than 0 and less than 1, the action would mark the grid location with a "possible ball". Otherwise, the action element would place a "ball" in the location. In this, a certainty factor of 0 was not considered to be a lack of certainty. Rather, it was an indication that the action element came from rules that did not exhibit any discrepancy when compared with other rules.

Among the Meta-Level Rules included in the rule base there are some that decide when the certainty attributed to the "possible balls" in the grid increases and, if necessary, convert them to "balls". This is done by having rules that calculate the number of grid locations with "possible balls" and maintain information about the number of different shots that cause the placement of a possible ball in one location. This information is used by other rules to convert locations that have been referenced by two shots from possible balls to balls, and, at the end of the game, when there are no more shots that can be fired into the grid, to convert possible balls to balls in order to make a final hypothesis.

The issue of certainty is also addressed explicitly in the Analysis Rules. As was explained in the previous section, whenever an Analysis Rule reasons about a specific shot, before the rule fires one of its conditions ascertains that the fact list did not contain a fact declaring that shot certain. If there is an Analysis Rule that can be fired in response to the shot, one of the actions in the rule asserts a fact stating that the shot is certain. If the rule base does not contain an Analysis Rule to cover a specific shot, the shot does not become certain and is displayed on the user interface in reverse video to indicate to the user that there is a potentially useful Analysis Rule missing from the rule base. This information can be useful in the selection of examples for improving the rule base in future iterations.

The method used to assign certainty factors to the rules as they were acquired

from the cases and processed through the various steps of the rule base creation was specifically designed by me for knowledge acquisition in the Blackbox domain. The method's benefits and limitations will be discussed in the next chapter.

Chapter 6

RULE BASE EVALUATION

6.1 Testing the Rule Base

Testing the prototype expert system involved several phases. Two human experts, Le Hoc Duong and John Lyons, who will henceforth be called Player 1 and Player 2 respectively, independently played the same two sets of games, called the *training set* and the *test set*. The data resulting from these game sessions was recorded and saved in four separate game files, one for each expert with each set of games. At first the automated knowledge acquisition process acquired one rule base from each training set game file. Thus in the first iteration of the knowledge acquisition process two rule bases were developed. Each rule base was examined in a preliminary run to remove any superfluous elements. In the first test runs, each of the rule bases was used to play the games in the training set and then the games in the test set.

These first test runs were followed by similar tests on different rule bases. One set of tests was conducted on rule bases acquired from the combined training and test sets of each of the human experts. Another set of tests was performed on rules acquired from the combined training set game files of the two human expert players.

The following sections describe the composition of the sets of games used in the experiment, detail the testing process, and present the results obtained from the tests.

	TRAINING	TEST1	TEST2	TOTAL GAMES	DISTINCT GAMES
TRAINING	2			29	27
TEST1	5	1		28	22
TEST2	0	0	0	71	71

Table 6.1: Game Sets

6.1.1 The Game Sets

The expert system using the acquired rule base was tested by playing the games in the training set game file (from which the rules had been extracted) and in two test sets. As described in Section 4.4.3, the games in the training set and in the first test set were selected at random. The second test set was constructed to include all the games of a 4 X 4 grid with two balls which were not included in the other two sets. Table 6.1 shows the composition of the three game sets in relation to one another. The training set had 29 games in total, two of which were identical to two others in the set, resulting in 27 distinct games. TEST1, the first training set, contained 28 games with one game identical to another in the test set and five others identical to games in the training set, yielding 22 additional different configurations. The second test set, TEST2, contained 71 games, which by design were all distinct. Figure 6.1 gives a set representation of the three sets of games within the universe of 4 X 4 games with 2 balls. Hence the knowledge for the expert system rule bases was acquired from 49 games and tested on 120 games.

6.1.2 Preliminary Examination

The two rule bases used in the initial testing were acquired from the training set games played independently by two human experts. These automatically acquired rule bases were supplemented by "Housekeeping Rules" (described in 5.3.3) and loaded into the expert system at separate times to be used to play the games in the training set and in the first test set. This preliminary examination revealed that

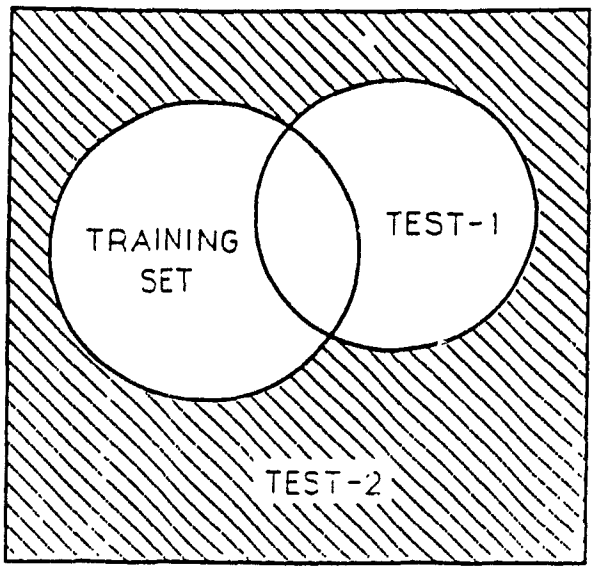

Figure 6.1: Set Representation of Training and Test Sets

some rules had not been adequately refined by rule merging during the knowledge acquisition process. This was due to the fact that not all rules underwent rule merging, since many of the derived rules were unique and were thus translated intact from the game files to their respective rule bases. As a result of the preliminary testing, any superfluous actions that were found to cause incorrect hypothesis modifications were removed from the rules.

The initial tests also demonstrated the need for the inclusion of some means of deciding when to change hypothesized "possible balls" to "balls" and when to terminate a given game session. Without rules to handle such decisions, the expert system did not know when to stop firing shots and often placed more "possible balls" in the hypothesis grid than were actually hidden in the game grid. Thus it was not possible to perform any useful measurements on the performance of the expert system. The Meta-Level Rules described in Section 5.3.3 were developed in response to this need. These rules were not acquired automatically but the heuristics used in their contents were derived from the inspection of automatically acquired rules. The rule bases used in the formal testing of the prototype expert system would henceforth include both Housekeeping Rules and Meta-Level Rules.

6.1.3 First Test Run Results

In the first testing phase the two rule bases (from the two experts' training sets) were used individually to play the games in the training set and the games in the first test set. Tables 6.2 and 6.3 give the results obtained in testing the rule base acquired from the games played by Player 1. Similarly Tables 6.4 and 6.5 show the results achieved in testing the rule base acquired from the games played by Player 2. In each of these tables, column 2 reports the scores achieved by the human expert; column 3 gives the scores achieved by the expert system playing with the rules acquired from the training set; and column 5 gives the deviation of the expert system's scores in those games from the human expert's scores.

The scoring used in the experiment was based on the convention described in Section 3.1.1. The maximum, or worst score, that could be achieved in each 4 X 4 game is 16. In the tables, any games that remained unsolved or whose solution contained an error received a score of 17. Unsolved games were marked with a 'U' and errors were marked with an 'X' after the score. Games that duplicated the grid composition of other games in the same set were not included in the tables or in the calculations. Thus the average score in each set was based on the number of distinct games in the set. If there were any unsolved games or games with errors, two averages were calculated - one including the scores of all the games, and one for the solved games only.

The average deviation was calculated using the sum of the simple deviation of each game's score from its counterpart in the human player's column. This method was elected over the mean squared deviation since it gave a more accurate picture of the performance of the expert system. Squaring the deviations would have distorted the expert system's performance by failing to differentiate between instances in which it outperformed the human expert and those in which the opposite occurred.

Table 6.2 shows that the average score obtained in playing the training set games with Player 1's initial rule base was 11.1852 without any errors or unsolved games. In Table 6.3 playing the test set games with the same rule base resulted in an average of 11.2222 for all the games. However, this test run had one unsolved game, reducing

the average score per solved game to 11. The deviations of these test run results from Player 1's own scores were 5.6667 for the training set games and 5.6923 for the solved test set games.

In Table 6.4, Player 2's first rule base obtained an average score of 11.5556 in playing the training set games. This average was reduced to 10.6087 when the four unsolved games were removed from the calculation. Table 6.5 shows that the test set games played with this rule base had an average score of 12.1852 with 8 unsolved games which resulted in an average per solved game of 10.1578. The deviations from Player 2's own scores were 4.9565 for the solved training set games and 4.7895 for the solved test set games.

6.1.4 Incremental Knowledge Acquisition

The second iteration of the knowledge acquisition process involved acquiring rules from the combined training and test cases saved by each human expert. These rule bases were also tested by playing the games in the training set and those in the test set. The scores achieved in these test results can also be seen in column 4 of Tables 6.2, 6.3, 6.4, and 6.5. The deviation of the expert system's performance from the human expert's results is given in column 6 of these tables.

These test results were compared with those of the rule bases obtained in the first iteration of the knowledge acquisition and the comparisons are detailed in this section. The rule bases acquired in the second iteration performed better than the earlier ones. The rule base acquired from Player 1's combined game files solved all the games in the training set, achieving an average score of 10.8519, which was an improvement of 2.98% over the first rule base. The deviation for this test was 5.3333, reducing the previous rule base's deviation by 5.88%. The same rule base correctly solved all the games in the test set, scoring an average of 9.9630 with a deviation of 4.6296. This score was 9.43% better than the solved games average for the first rule base, and the improvement in the deviation was 18.67%.

The second rule base derived from Player 2 also solved all the games in the training set games without any errors, scoring an average of 10.2222, with a deviation

GAME	PLAYER 1	RB1	RB1'	DEVIATION	DEVIATION'
GA1	6	6	6	0	0
GA2	5	14	14	9	9
GA3	6	15	15	9	9
GA4	5	16	16	11	11
GA5	4	14	14	10	10
GA6	8	13	13	5	5
GA7	6	14	14	8	8
GA8	4	4	4	0	0
GA9	6	14	14	8	8
GA10	4	13	13	9	9
GA11	8	16	16	8	8
GA12	5	15	15	0	10
GA13	5	14	14	9	9
GA14	6	6	6	0	0
GA15	4	4	4	0	0
GA16	6	7	7	1	1
GA17	8	5	5	-3	-3
GA18	5	5	4	0	-1
GA19	5	14	14	9	9
GA20	5	10	6	5	1
GA21	6	10	10	4	4
GA22	4	4	4	0	0
GA23	3	8	4	5	1
GA25	6	15	15	9	9
GA26	8	16	16	8	8
GA27	6	14	14	8	8
GA28	5	16	16	11	11
AVERAGE	5.5185	11.1852	10.8519	5.6667	5.3333
UNSOLVED GAMES (U)		0	0		

LEGEND	
PLAYER 1	first human expert's scores
RB1	expert system's scores using rules from training set
RB1'	expert system's scores using rules from combined sets
DEVIATION	deviation of RB1 scores from Player 1 scores
DEVIATION'	deviation of RB1' scores from Player 1 scores

Table 6.2: Rules From: Player 1 - Tested On: Training Set

GAME	PLAYER 1	RB1	RB1'	DEVIATION	DEVIATION'
GB1	3	8	4	5	1
GB2	3	12	3	9	0
GB3	8	11	8	3	0
GB4	4	4	3	0	-1
GB5	5	6	5	1	0
GB6	5	12	5	7	0
GB8	6	12	6	6	0
GB9	8	16	16	8	8
GB10	8	13	13	5	5
GB11	5	16	16	11	11
GB12	4	4	4	0	0
GB13	6	14	14	8	8
GB14	6	17 U	15	11	9
GB15	6	14	16	8	10
GB16	7	14	14	7	7
GB17	5	16	16	11	11
GB18	4	5	4	1	0
GB19	5	14	14	9	9
GB20	4	4	4	0	0
GB21	4	5	4	1	0
GB22	8	12	12	4	4
GB23	5	15	15	10	10
GB24	5	14	14	9	9
GB25	4	5	4	1	0
GB26	5	14	14	9	9
GB27	5	10	10	5	5
GB28	6	16	16	10	10
AVERAGE	5.3333	11.2222	9.9630	5.8889	4.6296
UNSOLVED GAMES (U)		1	0		
AVERAGE SOLVED		11		5.6923	

LEGEND	
PLAYER 1	first human expert's scores
RB1	expert system's scores using rules from training set
RB1'	expert system's scores using rules from combined sets
DEVIATION	deviation of RB1 scores from Player 1 scores
DEVIATION'	deviation of RB1' scores from Player 1 scores

Table 6.3: Rules From: Player 1 - Tested On: Test Set

GAME	PLAYER 2	RB2	RB2'	DEVIATION	DEVIATION'
GA1	5	6	6	1	1
GA2	3	14	15	11	12
GA3	6	14	15	8	9
GA4	6	16	9	10	3
GA5	5	17 U	14	12	9
GA6	7	13	13	6	6
GA7	6	-4	-4	-2	2
GA8	-4	5	5	1	1
GA9	6	5	5	-1	-1
GA10	6	16	12	10	6
GA11	8	16	16	8	8
GA12	8	10	8	2	0
GA13	5	17 U	13	12	8
GA14	7	8	8	1	1
GA15	-4	-4	-4	0	0
GA16	6	11	7	5	1
GA17	7	13	10	6	3
GA18	-4	5	5	1	1
GA19	-4	17 U	13	13	9
GA20	5	8	7	3	2
GA21	7	16	13	9	6
GA22	-4	7	7	3	3
GA23	-4	7	6	3	2
GA25	6	17 U	15	11	9
GA26	7	16	16	9	9
GA27	6	14	14	8	8
GA28	-4	16	16	12	12
AVERAGE	5.5556	11.5556	10.2222	6	4.6667
UNSOLVED GAMES (U)		4	0		
AVERAGE SOLVED		10.6087		4.9565	

LEGEND	
PLAYER 2	second human expert's scores
RB1	expert system's scores using rules from training set
RB1'	expert system's scores using rules from combined sets
DEVIATION	deviation of RB1 scores from Player 2 scores
DEVIATION'	deviation of RB1' scores from Player 2 scores

Table 6.4: Rules From: Player 2 - Tested On: Training Set

GAME	PLAYER 2	RB2	RB2'	DEVIATION	DEVIATION'
GB1	3	7	6	4	3
GB2	3	4	4	1	1
GB3	6	11	13	5	7
GB4	4	12	5	8	1
GB5	5	6	6	1	1
GB6	6	13	6	7	0
GB8	6	17 U	7	11	1
GB9	7	17 U	16	10	9
GB10	7	16	14	9	7
GB11	6	17 U	16	11	10
GB12	5	5	5	0	0
GB13	8	4	4	-4	-4
GB14	8	17 U	16	9	8
GB15	5	16	10	11	5
GB16	6	14	11	8	5
GB17	7	16	16	9	9
GB18	5	8	7	3	2
GB19	7	14	12	7	5
GB20	6	4	4	-2	-2
GB21	5	11	6	6	1
GB22	5	13	14	8	9
GB23	5	14	15	9	10
GB24	5	17 U	13	12	8
GB25	4	5	5	1	1
GB26	4	17 U	15	13	11
GB27	5	17 X	8	12	3
GB28	7	17 U	16	10	9
AVERAGE	5.55555	12.1851	10	6.62962	4.44444
UNSOLVED GAMES (U)		7		0	
ERRORS (X)		1			
AVERAGE SOLVED		10.1578		4.78947	

LEGEND	
PLAYER 2	second human expert's scores
RB1	expert system's scores using rules from training set
RB1'	expert system's scores using rules from combined sets
DEVIATION	deviation of RB1 scores from Player 2 scores
DEVIATION'	deviation of RB1' scores from Player 2 scores

Table 6.5: Rules From: Player 2 - Tested On: Test Set

of 4.6667. The average score improved by 3.61% over the first rule base average for solved games. The deviation improved by 5.85%. Player 2's second rule base also solved all the test set games correctly, achieving an average score of 10 and a deviation of 4.4444. The average score improved by 1.55% over the solved games using the previous rule base and the average deviation improved by 7.21%.

Analysis of the significance of this and other test results will be presented in Section 6.2.

6.1.5 Combining Experts

As the next step in the experiment the two game files of the human experts playing the training set were combined to generate another rule base. The scores that resulted when this rule base was used to play the training set games and the test set games appear in Tables 6.6 and 6.7, respectively. In both tables column 1 shows the scores achieved in playing the games and the other columns show how this rule base's performance compared with others. The combined experts' rule base, RB3, solved all the games in the training set and all but one in the test set. This was equivalent to the experience of the RB1 rule base and superior to that of the RB2 rule base, which failed to solve four games in the training set and eight games in the test set.

Columns 3 and 4 in the tables show that the average score achieved by RB3 in each of the game sets was better than that of RB1 and RB2. Columns 5 and 6 give the deviation of RB3's results from those of the human experts. In playing the training set games, the combined rule base's average score for the games solved was 6.29% lower than that achieved by the first rule base of Player 1, and 1.2% lower than Player 2's first rule base. When the combined rule base was used to play the test set, its average score for the solved games was 3.85% lower than Player 1's first rule base and 4.12% higher than Player 2's rule base. This last negative result was tempered by the fact that the combined rule base only left one game of the test set unsolved whereas the first rule base derived from Player 2 left eight unsolved games.

GAME	RB3	DEVIATIONS			
		RB1-RB3	RB2-RB3	RB3-P1	RB3-P2
GA1	8	-2	-2	2	3
GA2	14	0	0	9	11
GA3	16	-1	-2	10	10
GA4	6	10	10	1	0
GA5	14	0	3 U	10	9
GA6	13	0	0	5	6
GA7	5	9	-1	-1	-1
GA8	5	-1	0	1	1
GA9	5	9	0	-1	-1
GA10	9	4	7	5	3
GA11	16	0	0	8	8
GA12	15	0	-5	10	7
GA13	14	0	3 U	9	9
GA14	8	-2	0	2	1
GA15	4	0	0	0	0
GA16	10	-3	1	4	4
GA17	10	-5	3	2	3
GA18	5	0	0	0	1
GA19	14	0	3 U	9	10
GA20	9	1	-1	4	4
GA21	9	1	7	3	2
GA22	6	-2	1	2	2
GA23	7	1	0	4	3
GA25	15	0	2 U	9	9
GA26	16	0	0	8	9
GA27	14	0	0	8	8
GA28	16	0	0	11	12
AVERAGE	10.4815	0.7037	1.0741	4.9630	4.9259
UNSOLVED GAMES (U)		0			
AVERAGE SOLVED			0.7826		

LEGEND	
RB3	expert system's scores using rules from both Players
RB1 - RB3	deviation of RB1 scores (Table 6.2) from RB3
RB2 - RB3	deviation of RB2 scores (Table 6.4) from RB3
RB3 - P1	deviation of RB3 scores from Player 1 (Table 6.2)
RB3 - P2	deviation of RB3 scores from Player 2 (Table 6.4)

Table 6.6: Rules From: Combined Experts - Tested On: Training Set

GAME	RB3	DEVIATIONS			
		RB1-RB3	RB2-RB3	RB3-P1	RB3-P2
GB1	7	1	0	4	4
GB2	4	8	0	1	1
GB3	11	0	0	3	5
GB4	4	0	8	0	0
GB5	6	0	0	1	1
GB6	13	-1	0	8	7
GB8	13	-1	4 U	7	7
GB9	16	0	1 U	8	9
GB10	13	0	3	5	6
GB11	16	0	1 U	11	10
GB12	5	-1	0	1	0
GB13	5	9	-1	-1	-3
GB14	17 U	0 U	0 U	11	9
GB15	14	0	2	8	9
GB16	14	0	0	7	8
GB17	16	0	0	11	9
GB18	6	-1	2	2	1
GB19	14	0	0	9	7
GB20	4	0	0	0	-2
GB21	7	-2	1	3	2
GB22	13	-1	0	5	8
GB23	15	0	-1	10	10
GB24	14	0	3 U	9	9
GB25	5	0	0	1	1
GB26	14	0	3 U	9	10
GB27	10	0	7 U	5	5
GB28	16	0	1 U	10	9
AVERAGE	10.8148	0.4074	1.3704	5.4815	5.2593
UNSOLVED GAMES (U)	1	1	8		
AVERAGE SOLVED	10.5769	0.4231	0.8947		

LEGEND	
RB3	expert system's scores using rules from both Players
RB1 - RB3	deviation of RB1 scores (Table 6.2) from RB3
RB2 - RB3	deviation of RB2 scores (Table 6.1) from RB3
RB3 - P1	deviation of RB3 scores from Player 1 (Table 6.2)
RB3 - P2	deviation of RB3 scores from Player 2 (Table 6.1)

Table 6.7: Rules From: Combined Experts - Tested On: Test Set

6.1.6 Supplementary Testing

The universe of 4 X 4 games with 2 hidden balls from which the knowledge acquisition cases were drawn is finite and not too large. Therefore it was feasible to test the rule bases acquired in the experiment on a set of games within that universe that are disjoint from the games in the training set and the primary test set. A set of 71 such games was prepared and the expert system played them with each of the rule bases acquired in the experiment.

The results of this phase of the experimental testing is shown in Tables 6.8 and 6.9. RB1 and RB2 are the rule bases acquired from each player's training set sessions. RB1' and RB2' are the rule bases derived from adding the test set games to those in the training set. RB3 is the rule base from the combined training sets of the two experts.

These results are not as straightforward as those of the other tests. The overall performance of RB1' showed an improvement over that of RB1. The enhanced rule base left no unsolved games, compared with 2 unsolved by RB1, and the average score of RB1' was 4.67% lower than that of RB1, when only the solved games were considered in the average of RB1. These results are consistent with those witnessed in testing these rule bases on the training set and the test set games.

By contrast, whereas RB2' also reduced the number of unsolved games from 11 for RB2 to 3, its use also resulted in two errors as opposed to one for RB2. The overall average score for all 71 games in the set achieved by RB2' was an improvement of 3.47% over the equivalent measure for RB2. However, when all the unsolved games and errors were removed from the equation, the average score of RB2' showed a degradation of 1.45% from that of RB2. This is not a fair comparison since 12 games with a score of 17 were removed from the total scores for RB2 before calculating the average of the remaining 59 games, whereas only five games with a score of 17 were removed from the total for RB2'. All but one of the unsolved games from RB2 that were solved by RB2' had a score that was higher than the average score, thereby contributing to an increased average. The error that occurred in the testing of RB2' and not in RB2 was the result of a difference in the order of shots

GAME	RB1	RB2	RB3	RB1'	RB2'
GX1	16	16	16	16	16
GX2	16	16	16	16	16
GX3	15	15	15	15	15
GX4	14	14	14	14	15
GX5	14	14	14	14	14
GX6	13	13	13	13	14
GX7	13	16	16	13	16
GX8	15	13	16	15	15
GX9	15	16	16	15	16
GX10	8	3	3	7	3
GX11	11	7	7	14	7
GX12	17 U	17 U	17 U	10	17 U
GX13	12	4	4	14	4
GX14	14	12	14	6	15
GX15	5	7	7	2	7
GX16	8	17 X	10	8	17 X
GX17	8	9	9	3	4
GX18	12	17 U	12	13	13
GX19	2	5	5	3	5
GX20	10	10	10	10	4
GX21	15	14	15	16	15
GX22	4	6	7	4	7
GX23	10	17 U	10	10	17 U
GX24	14	12	14	10	15
GX25	10	4	4	4	4
GX26	9	9	9	10	7
GX27	13	17 U	11	13	16
GX28	8	8	8	8	11
GX29	16	16	16	16	16
GX30	14	16	16	14	16
GX31	11	17 U	14	14	14
GX32	14	17 U	14	11	14
GX33	14	16	14	14	14
GX34	15	17 U	15	15	14
GX35	17 U	17 U	17 U	6	9
GX36	13	6	12	13	8
GX37	10	11	11	8	13
GX38	6	6	6	6	6
GX39	7	8	8	9	8
GX40	4	6	6	4	7
GX41	8	17 U	5	8	17 U

Table 6.8: All Rule Bases - Tested On: Test Set 2(Part 1)

GX42	4	6	6	4	8
GX43	6	10	8	5	17 X
GX44	3	12	5	3	9
GX45	16	9	11	16	8
GX46	11	13	13	11	13
GX47	5	7	7	5	7
GX48	4	4	4	4	4
GX49	9	9	9	10	8
GX50	14	15	15	14	15
GX51	12	9	9	10	8
GX52	13	13	13	10	13
GX53	11	4	4	8	4
GX54	11	8	8	6	8
GX55	14	14	14	4	14
GX56	16	16	16	6	16
GX57	16	16	16	6	16
GX58	16	16	16	6	16
GX59	7	9	9	9	9
GX60	16	16	16	16	16
GX61	14	16	14	14	14
GX62	15	15	15	15	15
GX63	14	17 U	14	14	14
GX64	10	10	10	12	10
GX65	15	15	15	15	15
GX66	7	10	10	8	7
GX67	16	16	16	16	16
GX68	16	16	16	16	16
GX69	14	16	14	14	14
GX70	14	17 U	14	14	15
GX71	10	12	12	5	8
AVERAGE	11.5775	12.1690	11.4789	10.8873	11.7465
UNSOLVED GAMES (U)	2	11	2	0	3
ERRORS (X)		1			2
AVERAGE SOLVED	11.4203	11.1864	11.3188		11.3485

LEGEND	
RB1	using rules acquired from Player 1's training set
RB2	using rules acquired from Player 2's training set
RB3	using rules acquired from combined games in both Players' training sets
RB1'	using rules acquired from combined games in Player 1's training and test sets
RB2'	using rules acquired from combined games in Player 2's training and test sets

Table 6.9: All Rule Bases - Tested On: Test Set 2(Cont.)

fired in the game, as prescribed by the selection rules that resulted from the second iteration. A Meta-Level Rule was used to change a "possible ball" to a "ball", based on a heuristic that if two separate rule firings indicate that a location has a "possible ball" then it is likely that that location actually contains a "ball". In this case the rule did not work. A more accurate rule would have waited until all possible shots had been exhausted before changing any "possible ball" to a "ball". However such a rule would have resulted in the highest possible scores for all games in which possible balls were an issue, leading to a significant degradation of overall performance. Since this rule only caused such an error in two of 120 cases the benefits of its inclusion seemed to outweigh the risks.

The rule base combining the expertise of both experts had an overall performance that was better than that of RB1 and RB2, which were obtained individually from the two experts. Like RB1, RB3 only had two unsolved games. The average score of RB3, excluding unsolved games, was 0.89% lower than the equivalent score of RB1. When compared with RB2, RB3 has the same problem as RB2'. RB3 had only two unsolved games and no errors, compared with 11 unsolved and one error for RB2. In the average score that included all games, RB3 was 5.67% lower than RB2. However, when the unsolved games and errors were removed, the same phenomenon that affected the comparison between RB2 and RB2' took effect. In this case, six of the unsolved games from RB2 that were solved by RB3 had a higher score than the average and four had a lower score. Given this distribution, the average score of RB3 was pushed up slightly, resulting in an increase of 1.18% over that of RB2.

6.2 Analysis of Test Results

Several important facts emerge from the test data presented above. The first is that the rule bases acquired in the experiment were able to solve most domain problems presented to them. If the goal of the expert system was to solve the problem and the only performance measure was whether the solution was correct, the rule bases in the experiment acquitted themselves reasonably well.

The rule bases acquired from Player 1 performed particularly well, with the first rule base solving 97.5% of the 120 games tested. Player 1's second rule base, derived from the combined games of the training and test sets, performed even better, solving 100% of the games tested. While the rule bases acquired from Player 2 did not perform as well, they were adequate. Player 2's first rule base solved 80% of the 120 games tested and the second rule base solved 95.83% of the games. The rule base acquired from the two experts' training set game files was also very effective, solving 97.5% of the games tested.

A second fact that becomes apparent is that, if comparative performance measures such as scores are considered, the rule bases did not perform as well as the human players. The average deviations of the rule bases' performances from the average scores achieved by the human experts ranged from 80% (refer to Table 6.5 columns 2 and 6 AVERAGE) to 106.73% (refer to Table 6.3 column 2 AVERAGE and column 5 AVERAGE SOLVED) of the human experts' average scores. Thus, in the case of Blackbox where the goal is to solve the games with the lowest possible score, the rule bases acquired from the cases were not sufficient for the task at hand.

Using the average deviations as a criterion of relative performance, the worst performances recorded were those of the rule bases acquired from the training set and the best were those of the second rule bases acquired from the combined training and test sets. There was a wide divergence in the improvement observed in these deviations, ranging from 5.85% to 18.67%. However these results, as well as the improvement in average performance of the enhanced rule bases, demonstrated that incremental benefits can be realized by using an iterative approach in automated knowledge acquisition.

The rate of improvement recorded in the performance of the expert system when additional examples were used as a source of knowledge acquisition ranged from 1.55% to 9.43%. The reason for this variation may be due to the fact that very little care was taken in the selection of the examples in either the first training set or in the test set which became the incremental training set. In well-selected sets of training examples for incremental knowledge acquisition, it is expected that the increase

in performance would be monotonic. In the Blackbox knowledge acquisition experiment the selection of individual examples was not done methodically. Therefore, we do not attach any great importance to the significance of the level of improvement rates observed. Rather, the trend itself is considered to be the important aspect of the results of these tests. What is significant is that, without exception, the incrementally acquired rule bases performed better than their earlier counterparts. How long this trend can continue would be a function of the size of the available case base and of the methodology for selecting cases for knowledge acquisition.

It is reasonable to deduce from these results that as more cases are added to the training set, more diverse situations can be studied, additional rules can be included in the rule base, and previously incomplete rules can be augmented. In addition, the careful selection of the training cases could maximize the benefits of incremental knowledge acquisition.

In the Blackbox domain, additional cases can be selected to provide solutions for observed deficiencies in the rule base. As the expert system is currently designed, if it cannot fire a rule to analyze a shot, it displays to the user which rules it is missing (Section 5.3.4). To help fill the observed void in the rule base, examples that would yield the missing rules can be added to the training set. If these examples are not available, the domain expert can be asked to solve the game in question, following the same sequence of actions used by the expert system, thus providing a new example for the training set.

Another interesting factor perceived in the test results was the performance of the combined experts' rule base. Although in these tests the average score did not improve from that of the first rule bases in all the game sets, the combined experts' rule base did solve more games that its earlier counterpart. Like the observation made regarding incremental knowledge acquisition, the results of knowledge acquisition from combined expertise show a trend to improved performance. This would therefore suggest that using multiple experts as a source of knowledge acquisition from cases would have beneficial results.

Overall the results of the testing were encouraging. Although the expert system

did not come close to matching the performance of the human experts, it was able to solve most of the games it was given. Some of the weakness in the performance could be traced to deficiencies in the Analysis Rule section of the rule bases. In the process of the testing some shots were not fully analyzed by the expert system, and others were not analyzed at all. This necessitated the firing of more shots into the grid and raised the scores of the games. This lack of hypothesis formation on the part of the expert system was due to the fact that the exact sequence of actions used by the expert system in some games differed from that which occurred in the training cases. Thus, the expert system fired shots that may not have been used in the game files, meaning that the rules to analyze these shots could not be acquired from the cases. The incompleteness manifested by some other rules could be traced to the same causes. Each Analysis Rule in the rule base was meant to include various shots with the same entry-to-exit distance attributes. If certain shots had not occurred in the actual acquisition cases, then the action required to handle them was not included in the Analysis Rule. These deficiencies argue for the using more iterations of the knowledge acquisition process with carefully selected new cases which might help to improve the rule base.

The performance of the Blackbox expert system was also affected by the Selection Rules which were not complete nor very powerful. In the games tested the expert system often had to resort to the default rule for selecting shots from a predetermined ordered list of grid entry points. It is possible that the Selection Rules section of the rule base requires more iterations of the knowledge acquisition process to increase the number of specific situations covered by the rules. An important part of the deep strategy used to play Blackbox is the selection of the appropriate shots at the right juncture in the game. The emphasis of the knowledge acquisition method used in this experiment was on the hypothesis making aspects of the game. A change in focus might serve to improve the shot selection process.

Some of the positive results of the testing of the rule base could be attributed to the calculation of certainty factors that was described in Section 5.3.4. The certainty factors helped to determine whether an Analysis Rule that placed a "ball" in the grid would do so in all circumstances and game configurations. The primary

contribution of the certainty factors was in some rules that could select among two or more locations when allocating a "ball". The calculation of the certainty factor was designed to indicate for such rules that a "possible ball" rather than a "ball" should be marked on the grid locations, thus leaving the final decision to a later, more certain Analysis Rule or to a Meta-Level Rule. Another, less important contribution of the certainty factors was that they facilitated the process of removing superfluous actions from rules during the preliminary testing. In some rules that had not been sufficiently refined by rule merging the certainty factors changed "ball" placement to "possible ball" and thus prevented errors in the preliminary testing. The weakness of this calculation of certainty factors was that whereas a low level of certainty was used to alter some Analysis Rules by changing "balls" to "possible balls", a higher level of certainty was not utilized to enhance the power of other rules. Another problem is that with the small training set that was used in the thesis, the power of the certainty factors as a determining tool in placing confidence in rules could not be explored and tested adequately. This may also be the reason that the certainty factor calculation did not prove to be sufficiently powerful to be used automatically to remove extraneous elements from rules.

6.2.1 Analysis Summary

The rule bases generated in the experiment performed as expected. It was not assumed that the role of the domain expert in knowledge acquisition could be eliminated completely. The expectation was that automated knowledge acquisition could generate sufficient rules to construct a prototype expert system and that this prototype could be used to focus attention on salient issues in knowledge acquisition.

In the tests conducted in the experiments, it was apparent that the rule bases generated were not complete and needed improvement. However, the tests also indicated that some of this improvement could be achieved in an automated fashion by using repeated iterations of the knowledge acquisition process to incrementally build a rule base. Careful selection of the cases used in the process would be required to yield more consistent incremental benefits. However, this iterative process cannot

be expected to continue indefinitely. The nature of expert systems is that they can only solve problems in a limited domain. Therefore, at some time in the knowledge acquisition process the finite nature of the domain would become a factor and the improvement of the rule base would reach a saturation limit. At this point any subsequent improvement that might occur would be of a minimal nature. In order to detect the approach of a saturation limit and thus avoid performing extensive repetitive example analysis with little or no new knowledge being acquired, it would be necessary to continue measuring the rate of improved performance.

In planning this experiment it was also anticipated that after the automated knowledge acquisition from cases was complete the involvement of the domain expert would be required to critique the rule base and improve it where necessary. The Blackbox rule bases acquired in this experiment could benefit from the intervention of a human expert. The domain expert could help improve the process by helping to identify a method of case selection that would yield a representative set of preliminary training examples or a set of suitable examples for incremental knowledge acquisition. In addition the domain expert could participate by helping to add missing rules or to complete incomplete rule in the rule base. Such a participation by the human expert could strengthen the rule base and correct the deficiencies in the performance of the expert system.

Chapter 7

CONCLUSION

7.1 Contributions

The thesis demonstrated the feasibility of automated knowledge acquisition from a set of available cases through an example study. It showed that a knowledge base so generated could be used to build a prototype expert system with very little direct involvement of a domain expert. Thus, the domain expert's valuable time was reserved for the later stages of the knowledge acquisition process, when the focus would shift to refining and improving the prototype, requiring direct application of the domain expert's knowledge. By lessening the dependence on a domain expert's contribution in the early development stages, his or her limited time would be utilized more efficiently.

The conclusions that were reached in this thesis were based on the specific experiment involving the game of Blackbox. One wonders whether these conclusions can be generally extended to other domains beyond the realm of Blackbox. At this stage, we do not know the answer to this. For a domain to be considered suitable for the application of the knowledge acquisition method used in this thesis, the following conditions would need to exist:

C1. the existence of many actual cases stored in computer readable form

C2. data in which the roles of "cause" and "effect" or condition and action is either apparent or can be easily extracted based on a preliminary explanation of the composition of the cases

C3. data that lends itself to "easy" knowledge representation

C4. an effective method of selecting appropriate cases for training and testing, perhaps based on the commentary of the domain expert.

Three additional major points emerge from the results of the thesis. The first of these is that the knowledge acquisition process benefits from using the expertise of multiple domain experts. Where possible, case-based knowledge acquisition should be undertaken in a domain that has more than one source of cases. A set of cases that are generated by more than one domain expert in a domain can provide a much better insight into different approaches to solve the domain problem. This conclusion is borne out by the results obtained in this thesis using the combined knowledge of two domain experts, which are shown in Tables 6.6, 6.7, 6.8 and 6.9. When compared with the performance of rules acquired from the individual sets of cases, the rules derived from the combined sets performed better by resulting in lower average scores, and in some of the test sets, by solving more games. These results demonstrate that tapping the expertise of multiple domain experts is beneficial and desirable to the development of a knowledge base.

The second important point of the thesis is the assertion that iteration is a necessary feature of automated knowledge acquisition. With repeated applications of knowledge acquisition techniques the knowledge base can grow and improve. In case-based knowledge acquisition, as successive iterations analyze more cases, the knowledge base can be expected to continue to improve. The results of the thesis demonstrate the positive benefits of two iterations of the process that was used. Tables 6.2 through 6.5 show that, without exception, the rule bases built through incremental iteration performed better than their respective earlier counterparts. Although the results were positive, the expansion of the knowledge base cannot be expected to continue unabated over successive iterations. Indefinite growth might be restricted by limiting factors such as the finite availability of training and test cases and the methods for selecting appropriate cases. Therefore it is fair to expect that

successive iterations of knowledge acquisition would eventually reach a saturation point. These observations should not detract from the positive benefits that can be obtained from the judicious application of iterative knowledge acquisition, using some means of assessing when the process should be halted.

The third result of the thesis underlines the benefits that can accrue to the knowledge acquisition process from building a prototype expert system. The knowledge base derived for the prototype can serve as the basis for the development of a more comprehensive knowledge base for the operational expert system. From the process of developing and testing the prototype, the knowledge engineer can generate "meaningful" questions to ask the domain expert as well as guidelines for improving the knowledge base. During the development of the Blackbox prototype, questions arose about how to convert "possible ball" hypotheses to "balls" and when to declare that a game was over. When the prototype was being tested, another question that emerged was whether a "Meta-Level" rule that caused errors in two of the 120 test cases should be retained in the rule base (Section 6.1.6). The tests also highlighted the limitations of the prototype, showing cases that could not be solved and indicating which rules would be needed for their solution. A data bank of unsolved cases could be used as guidelines to the selection of cases for future iterations or as specific questions for the expert system to supply the missing rules. The experience in this thesis suggests that prototypes in other fields could also be used to generate probing questions to help further knowledge acquisition.

7.2 Future Work

I envision that the automated knowledge acquisition process as it was applied to the game of Blackbox could be expanded in future studies. An algorithm could be designed to perform additional analysis on the acquired rules and, based on that analysis, procedures could be written to perfect incomplete rules. Procedures could also be written to extrapolate from the knowledge inherent in the acquired rules and create rules to cover plausible situations that did not occur in the training cases. It is also possible that the rules that were acquired in the present experiment could

be generalized further to enable the expert system to play games on larger grids without requiring actual knowledge acquisition from the larger games.

The rule base might be improved by the design and addition of an interface to the expert system that would allow the human expert to intervene in situations where the rule base was deficient, browse the rule base, and insert or improve rules. The issue of knowledge acquisition from multiple experts could be also be studied further by increasing the number of experts contributing cases or by acquiring knowledge from cases of distributed playing where several experts cooperatively play the distributed game.

It would also be interesting to apply the ideas of automatically acquiring knowledge from cases to real life problems in domains where the previously mentioned conditions C1 through C4 apply. One such domain is the area of medical diagnosis. Patient data is often readily available, stored in computerized formats; symptoms and diagnosis, and diagnosis and treatment form natural cause-and-effect patterns. Another possible domain is that of credit approval. In this domain, too, records of previous credit applications exist along with a clear cause and effect pattern of measurements of financial stability and the granting or refusal of credit. Insurance underwriting and insurance claims adjustment are also likely candidate domains since insurance companies maintain many computerized records of existing policies that show the conditions that led to the issuing of policies, the rates that were charged, and claims settlements.

In the domains mentioned the selection of cases for knowledge acquisition could be divided along the lines of successful cases (cures, credit approvals, issuing of policies, out-of-court settlement of claims) and failures. Rule-based knowledge representation could be used to build knowledge bases for these domains. In addition, the use of frames could possibly enhance knowledge representation in the domains of medical diagnosis and insurance. However, since the extraction of frame-based knowledge is a totally different problem that was not properly addressed in this thesis, it would be inappropriate to comment on its feasibility other than to say that this too would be worthy of further study.

Bibliography

[1] J. H. Boose. Ets: A pcp-based program for building knowledge based systems. In *IEEE Western Conference on Knowledge-Based Engineering and Expert Systems*, pages 19 – 26, Amsterdam, The Netherlands, June 1986. Elsevier Science Publishers.

[2] B. G. Buchanan and T. M. Mitchell. Model-directed learning of production rules. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern Directed Inference Systems*. Academic Press, New York, 1978.

[3] C. Culbert. *CLIPS Reference Manual*. Artificial Intelligence Section, Johnson Space Center, Houston, 1989.

[4] C. De Koven and T. Radhakrishnan. An experiment in distributed group problem solving. In S. Gibbs, editor, *Multi-User Interfaces and Applications, Proceeding of the IFIP WG 8.4 Conference on Multi-User Interfaces and Application*, pages 61 – 76, September 1990.

[5] Richard O. Duda and Edward H. Shortliffe. Expert system research. *Science*, 220(4594):261 – 268, April 1983.

[6] L. Eshelman, D. Ehret, J. McDermott, and M. Tan. Mole - a tenacious knowledge-acquisition tool. *International Journal of Man-Machine Studies*, 26(1):41 - 54, January 1987.

[7] Richard Fikes and Tom Kehler. The role of frame-based representation in reasoning. *Communications of the ACM*, 28(9):904 – 925, September 1985.

[8] M. Freiling, J. Alexander, S. Messick, S. Rehfuss, and S. Shulman. Starting a knowledge engineering project: A step- by-step approach. *AI Magazine*, pages 150 - 163, Fall 1985.

[9] S. Nowlan G. Kahn and J. McDermott. More: An intelligent knowledge acquisition tool. In *Proceedings of Ninth International Joint Conference on Artificial Intelligence*, pages 581 – 583, 1985.

[10] W. A. Gale. Knowledge-based knowledge acquisition for a statistical consulting system. *Internation Journal of Man-Machine Studies*, 26(1):55 64, January 1987.

[11] C. Grossner, J. Lyons, and T. Radhakrishnan. Validation of an expert sytem intended for reasearch in distributed artificial intelligence. In *Second CLIPS Conference*, 1991.

[12] T. Gruber and P. Cohen. Principles of design for knowledge acquisition. In *Proceedings of The Third Conference on Artificial Intelligence Applications*, pages 9 - 15, February 1987.

[13] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat. *Building Expert Systems*. Addison-Wesley, Reading, MA, 1983.

[14] R. R. Hoffman. The problem of extracting the knowledge of experts from the perspective of experimental psychology. *AI Magazine*, pages 53 - 67, Summer 1987.

[15] Judith H. Lind. Downloading the expert : Efficient knowledge acquisition for expert systems. In *Proceedings of the 1986 International Conference on Systems, Man, and Cybernetics, SMC*, pages 547 - 551, 1986.

[16] J. Lyons. Blackbox game: Program written in c. Technical report, Concordia University, Montreal, Quebec, 1987.

[17] J. Lyons. Bb-ace: An expert system to solve blackbox. Technical report, Concordia University, Montreal, Quebec, 1991.

[18] S. Marcus. Taking backtracking with a grain of salt. *International Journal of Man-Machine Studies*, 26(4):383 - 398, April 1987.

[19] S. Marcus, J. McDermott, and T. Wang. Knowledge acquisition for constructive systems. In *Proceedings of Ninth International Joint Conference on Artificial Intelligence*, pages 637 - 639, 1985.

[20] R. S. Michalski and R. L. Chilausky. Learning by being told and learning by examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems*, 4(2), 1980.

[21] D. Michie, S. Muggleton, C. Riese, and S. Zubrick. Rulemaster: a second-generation knowledge engineering facility. In *IEEE Conference on Artificial Intelligence Applications*, 1984.

[22] Sanjay Mittal and Clive L. Dym. Knowledge acquisition for multiple experts. *AI Magazine*, pages 32 - 36, Summer 1985.

[23] J. G. Neal and D. J. Funke. Transfer of knowledge from domain expert to expert system: Experience gained from jamex. In *Proceedings of the 1986 International Conference on Systems, Man, and Cybernetics, SMC*, pages 536 - 540, 1986.

[24] M. A. Newstead and R. Pettipher. Knowledge acquisition for expert systems. *Electrical Communication*, 60(2):115 - 121, 1986.

[25] Kamram Parsaye and Mark Chignell. *Expert Systems For Experts*. John Wiley & Sons, Inc, New York, 1988.

[26] K. Pitula. Blackbox: Revised game program written in c. Technical report, Concordia University, Montreal, Quebec, 1988.

[27] K. Pitula, T. Radhakrishnan, and C. Grossner. Distributed blackbox: A testbed for distributed problem solving. In *Proceedings of the International Conference on Computers and Communications*, March 1990.

[28] K. Pitula, T. Wieland, and C. De Koven. Blackbox: Player's manual. Technical report, Concordia University, Montreal, Quebec, 1989.

[29] K. Pitula, T. Wieland, and C. De Koven. Distributed version of the game of blackbox. Technical report, Concordia University, Montreal, Quebec, 1989.

[30] D. S. Prerau. Knowledge acquisition in the development of a large expert system. *AI Magazine*, pages 43 - 51, Summer 1987.

[31] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221 - 234, September 1987.

[32] M.R. Railey. *Dynamic Control Structures for Cooperating Processes*. PhD thesis, University of Illinois, Urbana-Champaign, 1986.

[33] A. H. Silva and D. C. Regan. Using cognitive psychology techniques for knowledge acquisition. In *Proceedings of the 1986 International Conference on Systems, Man, and Cybernetics, SMC*, pages 530 - 535, 1986.

[34] S. M. Weiss and C. A. Kulikowski. *A Practical Guide to Designing Expert Systems*. Rowman & Allanheld, New Jersey, 1984.