INLEN: A Methodology and Integrated System for Knowledge Discovery in Databases

Kenneth A. Kaufman

MLI 97-15

INLEN: A METHODOLOGY AND INTEGRATED SYSTEM FOR KNOWLEDGE DISCOVERY IN DATABASES

A Dissertation submitted in paartial fulfillment of the requirements for
the degree of Doctor of Philosophy at George Mason University

Kenneth A. Kaufman

Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA 22030-4444

Director: Ryszard S. Michalski

PRC Chaired Professor of Computer Science,
Information Technology and Systems Engineering
School of Information Technology and Engineering, George Mason University

MLI 97-15
P97-22

November 1997

Dedication

To JJ, who stood by me.

ACKNOWLEDGMENTS

I would like to thank Professor Ryszard S. Michalski, my Dissertation Director, for his guidance and support, and for the excellent exchanges of ideas we were able to engage in during the period of this research. I would also like to thank the members of my Dissertation Committee, Professors Kenneth A. DeJong, Larry Kerschberg, and Gheorghe Tecuci, for their challenging ideas, their putting forth of their views of machine learning and knowledge discovery, and their encouragement of this work.

The implementation of the system INLEN is based on the AURORA system, developed by Ryszard S. Michalski and Bruce Katz of International Intelligent Systems, Inc. (INIS). I would like to thank INIS for providing the source code of the AURORA system.

I would like to thank the collaborators who shared in this research. Jim Ribeiro was a great help in the C-language implementation of INLEN-1; he also provided data, and helped in developing the knowledge segment structure. Dr. Eric Bloedorn, Dr. Ibrahim Imam, Dr. Mark Maloof and Dr. Janusz Wnek's interest in AQ resulted in many interesting ideas' coming to fruition.

I would like to thank Professors Tibor Vamos, Member of the Hungarian Academy of Sciences, and Tomasz Arciszewski for the data they provided and for their interest in this line of research. I would also like to thank Professor Reuven Karni for the ideas we shared and pointers to related research.

I would like to thank those people who helped to develop the software that has been integrated into INLEN. Those with whom I interacted directly during my doctoral research include Robert Bethel, John Doulamis, Joan Elliott, Jim Kelly, Gray Jones, Jim Mitchell, and Witold Szczepanik.

I would like to thank Professor Murray F. Black, Dean of the School of Information Technology and Engineering; and Professor David Rine for their support.

I would like to thank the faculty members who assisted my passage through the doctoral program, particularly, in addition to my Dissertation Commitee, Dr. Paul Lehner, who took part in the construction of my Qualifying Examination.

I would like to thank Dr. Jerzy Bala, Alan Schultz, Debbie Walko, and Tomasz Dybala for their work at various times during the period of this research in maintaining the facilities on which the work was in part performed.

I would like to thank the individuals who were comrades and collaborators at various points during this effort, including Ashraf Abdel-Wahab, Francesco Bergadano, Tom Channic, Kaihu Chen, Hugo De Garis, Kejitan Dontas, Terry Janssen, Heedong Ko, Larry Sklar, Pawel Stefanski, Bradley Utz, Haleh Vafaie, Jianping Zhang and Qi Zhang.

Last but not least, I would like to thank my family and friends, without whom this work would never have happened.

Table of Contents

List of Figures

INLEN: A METHODOLOGY AND INTEGRATED SYSTEM FOR KNOWLEDGE DISCOVERY IN DATABASES

Kenneth A. Kaufman, Ph.D.
George Mason University, 1997
Dissertation Director: Dr. Ryszard S. Michalski

Abstract

This thesis presents a methodology for multistrategy knowledge discovery from databases, and its experimental validation through the implementation and testing of the INLEN system. The presented methodology is based on the integration of diverse machine learning operators with traditional data analysis tools into a multi-operator environment. Among the issues that must be confronted in order to implement such an architecture are enabling the machine learning tools to handle the structured or numerical attribute domains they are likely to encounter, developing a sufficiently rich knowledge representation system to allow the different operators to pass necessary information to each other and to human data analysts, and the creation of a means by which a system implemented under such an architecture can function semi-autonomously, rather than requiring a domain expert to analyze the output from each step in order to determine how next to proceed.

The methodology outlined here is implemented in a system called INLEN, based on the AURORA system developed by International Intelligent Systems, Inc. INLEN integrates a database, a knowledge base, and machine learning methods within a uniform user-oriented framework. A variety of domain-independent machine learning programs are incorporated into the system to serve as high-level *knowledge generation operators* . These operators can generate diverse kinds of knowledge about the properties and regularities existing in the data. Examples of INLEN's abilities and performance are presented.

The goals and contributions of this research are to investigate the problems presented to machine learning methods by data exploration problems, to enhance the exploitation of background and discovered knowledge by machine learning programs, to develop control systems for integrating knowledge discovery technologies, and to utilize these advances by implementing a system capable of contributing knowledge and understanding in complex real-world domains.

1 PROBLEM STATEMENT

1.1 Background

The goal of this research is to advance the state of the art in knowledge discovery in databases through the development and application of an integrated machine learning approach. With the rapid growth in the amount of potentially useful information that people in all walks of life have access to, the management and processing of this information has become proportionally more important. The storage and management of this information is often automated in the form of computer databases.

With today's technology, we can store and retrieve the data with ease and we can take action based on important knowledge derived from the data, but the task of generating and extracting this useful knowledge remains as a major bottleneck in data processing. Thus the means of getting from data to knowledge has become one of the key areas of research in information technology. Software packages have been developed to automate the extraction and processing of knowledge from databases.

Traditionally, these knowledge discovery systems have followed numerical and statistical approaches that would be able to, for example, detect correlations between given factors or analyze the tendencies and variability of certain features in the data. Such systems lack the ability to perform various other types of inference that might be useful in the analysis of the data. Perhaps their biggest weakness is the inability to alter the description space from that which has been given to them as input, instead relying on their initial input, as described by a domain expert. Furthermore, these systems are not able to produce their own conceptual explanations of either the meaning of their findings or the conditions under which certain relationships may hold; a human analyst must interpret the findings to get such an explanation. Such a task is becoming more difficult as the quantity and complexity of available data increases.

Because of this, the machine learning community has taken an interest in the problem of knowledge extraction from databases. Machine learning approaches can overcome some of the limitations inherent in traditional data analysis methods. For instance, constructive induction and deduction methods can improve the data description space by analyzing the data itself, the preliminary knowledge learned by the discovery system, and/or the background knowledge provided by the domain expert. Symbolic learning methods have the advantage of representing their knowledge in such a way that it is very easy for users to understand and explain the meaning of the discovered knowledge.

Machine learning researchers have developed an assortment of domain-independent programs that can perform a narrow set of symbolic learning tasks. The weakness of these programs is the fact that they are so task-specific. While conceptual clustering programs, for example, are able to determine useful ways to group and describe sets of objects, they were not designed to generate equations governing quantitative data, create rules distinguishing between classes of objects, select representative examples from a larger database, or improve a ruleset based on new data

Recently researchers have begun exploring possibilities in multistrategy learning (e.g., Michalski, 1991a; Michalski and Tecuci, 1991; 1993; Michalski and Wnek, 1996), designing systems that can apply different machine learning tools as is appropriate to the problem at hand. For example, multistrategy task-adaptive learning (Michalski, 1991b; Tecuci and Michalski, 1991) is a paradigm by which a learning strategy is selected based on the necessary learning task as determined by the relationship between input facts and existing knowledge. Multistrategy systems can even combine radically different learning paradigms, such as symbolic methods with neural networks and statistical packages.

A difficulty faced by multistrategy systems in data exploration tasks is that they are too human-dependent. That is, they either have hard-wired tool application sequences, or they rely on their users to determine when it is appropriate for a particular tool to be employed. For large-scale tasks, this can be time-consuming, inefficient, and subject to human error.

A further difficulty for machine learning methods when applied to large databases is that they are often tuned to learning in a carefully supervised environment, in which little background knowledge is required, and in which they have access to nearly complete data that typically has little structuring within a record's features. Many real-world databases have large amounts of unavailable or inapplicable information. Furthermore, the attributes in many databases will assume a rich background knowledge of which the learning programs may not be equipped to take advantage.

1.2 Goals of this research

This research aims to develop a methodology that will overcome some of the limitations of current data exploration technology by integrating and applying modern techniques of machine learning and discovery to databases, while advancing machine learning technology in ways that make it more amenable to data analysis. Specifically, the limitations addressed by this work are as follows:

- The task-dependency of single-strategy machine learning-based discovery systems.
- The dependency of multistrategy systems on human guidance.
- Machine learning programs' inability to exploit domain-oriented background knowledge.
- Machine learning programs' difficulties in dealing with incomplete data.
- The inability of learning and discovery programs to benefit from each other's generated knowledge.

In order to meet these challenges, this work addresses the following problems:

- How to coordinate and integrate diverse learning and discovery tools into a unified data exploration environment.
- How to represent domain-specific background knowledge such that the learning and discovery tools may use it to their advantage.
- How to manipulate the data representation space in order to produce more meaningful results.

- How to store and maintain discovered knowledge such that it may be used fully by subsequent applications of different discovery tools.
- How to reduce the human's burden in the knowledge discovery task by providing a means for the multistrategy system to select what operators to invoke next.

In order that the proposed methodology may be validated through application to data-analysis problems, it is being implemented in a large-scale system, called INLEN, for conceptually analyzing databases and discovering regularities and anomalies in them.

The architecture of INLEN integrates a relational database and a knowledge base with a variety of machine learning programs, implemented in the form of *knowledge generation operators*, that create new "nuggets" of data and/or knowledge from existing data and knowledge. The system is designed to incorporate a wide spectrum of knowledge generation operators, such as those for creating conceptual descriptions of sets of facts, identifying logical regularities and similarities among facts or groups of facts, inventing conceptual classifications of data, generating new attributes to better describe data, selecting relevant examples or attributes, formulating equations governing quantitative data as well as the conditions of their applicability, as well as operators representing known numerical and statistical techniques. These operators employ all basic forms of inference—deduction, induction or analogy. They allow a user to make discoveries in the data using a variety of strategies, and therefore INLEN can be viewed as a multistrategy data exploration system that extends the frontiers set by its predecessor systems QUIN [Michalski, Baskin and Spackman, 1982; Michalski and Baskin, 1983; Spackman, 1983], ADVISE [Michalski and Baskin, 1983; Michalski et al, 1987; Baskin and Michalski, 1989] and the AURORA system developed by Michalski and Katz [INIS, 1988]. In particular, AURORA provided INLEN with its "look-and-feel," and with preliminary versions of its rule learning, optimization and testing, and advisory operators.

This research involves the analysis of the issues, problems and requirements for the development of a system such as INLEN. Because of the nature of its architecture, this development involves both the adaptation of existing knowledge discovery tools to a common environment and the creation and implementation of new tools and techniques. INLEN also serves as a basis for experimentation by application to "real-world" problems and the generation of new knowledge and understanding in the domains of application. In doing so, this research will confirm the hypothesis that integrated machine learning systems are able to contribute to knowledge discovery in databases above and beyond traditional manual or statistical methods.

1.3 Organization of this dissertation

Chapter 2 discusses related research, both direct predecessors of this work and parallel efforts to attack the knowledge discovery problem. Chapter 3 categorizes the knowledge discovery tasks addressable by the INLEN architecture. In Chapter 4 the INLEN methodology is presented and framed within the research problems presented above. Chapter 5 discusses the implementation of this methodology, and Chapter 6 provides examples of the results of its application. Chapter 7 illustrates the KGL language with examples of code for generating and processing knowledge. Chapter 8 summarizes the results and contributions of this work

and describes possible directions for the continuation of this line of exploration. The Appendix contains technical descriptions of the INLEN-3 package, and is intended to serve as a guide for future INLEN programmers.

2 RELATED WORK

The idea of a multi-operator approach to knowledge discovery was formulated by Michalski in the 1980s. The first such effort, from which INLEN derived much of its conceptual architecture, was the QUIN system (**Qu**ery and **In**ference), a combined database management and data analysis environment [Michalski, Baskin and Spackman, 1982; Michalski and Baskin, 1983; Spackman, 1983]. QUIN was designed both as a stand-alone system, and as a subsystem of ADVISE, a large-scale inference system for designing expert systems [Michalski and Baskin, 1983; Michalski et al, 1987; Baskin and Michalski, 1989].

The INLEN architecture expands on QUIN's design of multiple operators by incorporating a number of new learning and inference operators. Additionally, it maintains a more complex knowledge base than QUIN's, which was designed primarily with expert system rule bases in mind.

The user interface for the early versions of INLEN grew directly out of the AGASSISTANT [Katz, Fermanian and Michalski, 1987] and AURORA [INIS, 1988] systems. AGASSISTANT was a shell for developing agricultural expert systems capable of acquiring knowledge through a machine learning program acting in the role of knowledge engineer and creating a knowledge base from classified examples. AURORA expanded upon AGASSISTANT by introducing new learning techniques (program AQ15 [Michalski et al, 1986]) and expanding the capabilities of the incorporated expert system shell. The result was a general-purpose PC-based expert system shell with learning and discovery capabilities.

In recent years, new tools have been developed – in particular, more advanced inductive learning systems, e.g., AQ15c [Wnek et al, 1995], a C-language reimplementation of AQ15; AQ17 [Bloedorn, Wnek and Michalski, 1993], a version of the AQ learning program capable of multistrategy constructive induction; and ABACUS-2 [Greene, 1988], a program for integrated quantitative and qualitative discovery.

Other work that has influenced the development of INLEN includes EMERALD [Kaufman, Michalski and Schultz, 1993; Kaufman, and Michalski, 1993], a user-oriented multi-program environment for education and research in machine learning, and the work in the area of expert database systems [Kerschberg, 1986, 1987, 1988].

The integration-of-operators architecture upon which INLEN's has been based may be found in independent work in application domains other than knowledge discovery. For instance, CONDOR (Strat, 1992) follows such a philosophy in the domain of computer vision and object recognition. In CONDOR, different operators (e.g., edge detection) are invoked by a heuristic-based control engine in an attempt to apply the optimal tools toward recognizing different viewed objects.

There has been a variety of other research relating to various aspects of the knowledge discovery in databases (KDD) problem, for example [Piatetsky-Shapiro and Frawley, 1991; Piatetsky-Shapiro, 1991 and 1993; Simoudis, Han and Fayyad, 1996]. The focus of much of this work is application-oriented, in which existing systems are brought in for assessment of applicability to specific knowledge discovery tasks, with a low priority for general applicability. These systems are typically limited in their available learning strategies, which in turn are directed toward a specific learning goal. For example, Alexander, Bonissone and

Rau (1993) applied C4.5 [Quinlan, 1990] and statistical methods to a sales database in order to improve marketing strategies.

Other KDD research investigates low-level knowledge discovery strategies that can then be incorporated into various knowledge discovery systems rather than serve as stand-alone modules or systems. One such example is the work by Piatetsky-Shapiro and Matheus (1993), in which they have developed a numerical method for estimating the strengths of dependencies among the attribute values in a database.

Unlike the work mentioned above, INLEN's goals focus on both the exploration of theoretical and methodological issues in knowledge discovery and the applicability to practical problems. Hence, the architecture is designed to be domain-independent, while being implemented in a specific knowledge discovery system with a broad spectrum of novel knowledge discovery operators.

Other KDD research not limited to a specific application, nor to low-level methodological issues, has typically explored paths other than INLEN's multistrategy machine learning-oriented approach. Many developed systems use statistically or numerically based methods. For example, GLS (Zhong and Ohsuga, 1994) uses a multi-agent architecture similar to INLEN's in that different operators may be called upon for different discovery tasks. In GLS, these operators are statistical and numerical in nature, and they can be applied to a database in parallel. As such, they work under the limitations of statistical and numerical operators mentioned above.

Among the non-statistical approaches, IMACS (Brachman et al, 1993) is an interface that allows a user's knowledge to guide the discovery process in a database system. Its operators allow the user to define criteria for concept definition and detection of change in the data over time. This method leaves most of the discovery process up to the user (as specified by the input requests); it instead performs chiefly as an advanced query processor and definer of views of the data and as a system able to monitor the database for changes in trends tagged by the user. Another multistrategy approach, RECON (Simoudis et al, 1994) combines single inductive and deductive reasoning into a general-purpose knowledge discovery environment. A third approach, RDT/DB (Morik and Brockhausen, 1996), combines inductive logic programming with dependency-detection and value agglomeration operators, resulting in a knowledge discovery tool that works closely with the user.

In comparison with these approaches, the INLEN methodology focuses more on symbolic learning operators, and is designed for the incorporation of a large number of tools, rather than just one or two for inductive reasoning. By necessity, this leads to an increase in the complexity of the system's infrastructure and its knowledge base.

Recently, integrated systems using modular architectures similar to INLEN's have been developed. KEPLER (Wrobel et al, 1996) facilitates the plugging in of new knowledge discovery tools, and DBMiner (Han et al, 1996) combines query-accessible symbolic and statistical knowledge discovery operators.

A major difference between INLEN and the approaches used by these systems is that INLEN does not simply make a set of tools available to the user; one of its focuses is to incorporate a knowledge base built in such a way as to facilitate the propagation of knowledge from one operator to another, and to organize this knowledge in such a way that it can be presented to a

user or, additionally, to INLEN itself through the KGL knowledge processing language, and used for task selection based on the nature of the knowledge.

One of the fundamental concerns of INLEN is the ability to integrate diverse learning and discovery techniques. Some of the work in the multistrategy learning community approaches such problems. For example, MORPH (Gould and Levinson, 1991) combined inductive (through finding commonalities in graphical representations of similar results), deductive (through explanation-based generalization) and genetic operators (to search for patterns that best describe positions) in order to improve its chess-playing ability.

CKRL (Morik, Causse and Boswell, 1991) is an approach to integrate different learning strategies by developing a canonical representation language to serve as a universal interface for a multistrategy toolbox. Users may define rules, concepts and functions and have them translated into forms usable by available tools. In contrast, the KGL language developed as part of this work serves more as a method for integrating and orchestrating different operators, as opposed to a server of knowledge to and from the various operators.

In summary, the contributions of INLEN over its predecessors and independently developed systems include:

- Being the first large-scale architecture for symbolic-oriented integrated discovery.
- Expanded use of domain-oriented background knowledge, particularly *structured attribute* representation.
- The concept of a *knowledge segment*, in which different forms of knowledge are linked to one another and to exemplary data.
- The creation and implementation of a meta-level language for semi-autonomous selection and execution of data exploration tasks.

3 A RANGE OF POTENTIAL DATA MINING AND KNOWLEDGE DISCOVERY PROBLEMS

This chapter explores the different types of tasks that may be necessary in order that an analyst's data exploration goals may be met. Specifically, the analyst may be faced with an organized set of information that may be symbolic, numeric, or a combination of the two. The information may be complete or incomplete, and it may or may not include incorrect data. The records in the database may refer to individual objects or events, or they may include timestamped sequences of facts about a given entity.

It is assumed that the relational data model is being used—tables of rows and columns in which each row represents a record in the database and each column represents a data attribute. While such a condition is not required for the discussion below, it does serve to simplify the presentation.

The first concern of a data analyst is to ensure that the data set is appropriate for the pending discovery task. Otherwise, operators should be called upon to improve it. If the number of records is too large for an upcoming task, a subset must be chosen that will be suitable for it. Records may be chosen randomly, or according to some task-oriented criterion. For example, the "method of outstanding representatives" [Michalski and Larson, 1978] prepares a data set for the learning of classification rules by selecting as training examples the records that tend to be representatives of their class near the borders of their class in the event space. In this way, the learning agent will have a better picture of the relevant factors near borders between the classes in the representation space.

Similarly, if there are too many attributes in the representation space or it is believed that some are not relevant to the upcoming discovery task, operators can be called upon to retain only the attributes that are most likely to be useful to the task. The selection criteria may differ depending on the nature of the problem. For example, the attribute set most suitable for generating classification rules for one decision attribute may differ from the one most suitable for classification based on another decision attribute. Similarly, the attribute set most suitable for learning a set of decision rules may differ from the one most suitable for learning a decision structure from the same data (Section 4.3.4).

Sometimes none of the provided attributes generate particularly satisfactory results for a given discovery task. In this case, it may be possible to generate new attributes from the existing ones that are better suited for the task. One method is through abstraction; numerical attributes may be discretized into intervals, and nominal attributes may have their values structured into generalization hierarchies. Another method involves applying constructive induction operators to create new attributes whose values are functions of the values of one or more of the existing attributes. If time-dependent data are present, operators may be used to create new attributes from existing ones that are in whole or in part time-based and that capture the time-dependent patterns in the data (Davis, 1981).

When the user is not focusing on a particular class of concepts (i.e., no attribute has been selected as a target or decision attribute), there are a number of available discovery strategies,

known as *interdependence methods*, due to their lack of a single dependent attribute. A conceptual or numerical clustering method may be used to create groupings of the data and, in doing so, recommend a classification schema. A statistical analysis of the data may be performed in order to identify some numerical relationships among the data. Other numerical relationships may be found through an equation-generating operator which attempts to find mathematical regularities among some or all of the examples in the database. Another option when not focusing on a particular decision class is to use a concept learning operator in a "grand tour" of the data by assuming in turn that every possible decision attribute is the one of chief concern, and applying the discovery operator appropriately.

When the analyst's goal has been more explicitly specified, more goal-oriented operators can be applied. Given a classified data table, tools for learning from examples may be used to formulate knowledge for characterizing a class of examples or for differentiating between classes. The resulting knowledge may have declarative form (decision rules) or procedural form (decision structures, decision lists). The actual operator used for such learning will depend on the nature of the data. If the data is time-dependent, operators for determining patterns in sequences may be used. If the data is known to be noisy, the discovery operators may be tuned to filter out some of the noise.

If the data analyst wishes to test the validity of some assumed or previously generated knowledge, a knowledge testing operator may be invoked the checks the validity of the knowledge against a new dataset. If the knowledge is found to be insufficiently accurate, or if the analyst wishes to do so based on a new influx of data, the analyst may invoke an incremental learning operator to fine-tune the knowledge base based on the newly available information.

Knowledge may be believed to soundly represent the application domain and the user's goals, but it may still be desired that the form of the knowledge be improved. Operators for doing so include rule optimization operators, knowledge abstraction/concretion operators, and operators to generate task-oriented procedural knowledge from declarative knowledge.

Once a suitable knowledge base has been generated, users will want to be able to apply the knowledge to such tasks as filling in missing data, analyzing an unknown situation, and getting advice on what actions to take. Tools for such tasks include expert system shells, procedural knowledge application engines, and knowledge testing programs when given unclassified input.

Finally, the data analyst may need to have the data or knowledge presented in a more easily understood way than through a simple display of the data records, decision rules, etc. Hence, there exists a need for data and knowledge visualization operatiors that can depict selected portions of the data knowledge in a way most understandable and useful to the analyst.

In summary, it is very likely that during the course of data exploration, an analyst will be faced with a very wide variety of problems. There exist tools to take on each of the listed classes of problems; however, no single one of them comes close to covering the entire presented spectrum. Thus, it is important that the next generation of data mining tools is able to meet these complex needs by integrating diverse methods into a uniform data exploration environment.

4 METHODOLOGICAL ISSUES

The motivating idea behind this research is to develop a methodology for integrating database, knowledge base, and machine learning technologies in order to provide a user with a powerful system of tools for manipulating both data and knowledge, and for extracting from that data and/or knowledge new or better knowledge. The approach is to build an "intelligent assistant" that can improve the effectiveness of a data analysis expert, and obtain important discoveries and conclusions partially on its own. Such a system can serve as an assistant for the conceptual analysis, discovery and explanation of patterns in databases.

This approach employs a synergistic system that allows a human expert and a computer tool to perform the tasks that each party is better suited for. When a database is large, it may be difficult for an analyst to find patterns due to the sheer volume of data. A learning program can be especially helpful in such situations, and can help to avoid the possible human error of overlooking something of note in the data. A system that can process and filter data faster than a human analyst, with an equal or lower error rate, will be very useful in domains where large amounts of data analysis are necessary. The user, in turn, can help to guide the machine's search, and separate relevant from irrelevant.

The role for the intelligent assistant in this model is to search for all kinds of qualitative and/or quantitative patterns, and to notify an analyst about the patterns viewed as important. These patterns can be formulated by applying methods of symbolic concept learning, or by the use of traditional statistical methods when appropriate. Experiments with some existing machine learning programs have shown that these programs can find unexpectedly simple patterns that are difficult for people to recognize (e.g., Falkenhainer and Michalski, 1990), or discover regularities that would be hard to formulate without an aid of a program (e.g., Michalski, 1983).

Systems able to extract useful knowledge from large databases can be usefully applied to many fields involving complex decision making, such as resource allocation and management, business transactions, medicine, chemistry, physics, economics, demographics, global change, scheduling, planning, etc.

4.1 Architectural design

In order to meet its knowledge discovery goals, the underlying data and knowledge representation in INLEN is the *knowledge segment*, which links relational tables with rules, equations and/or hierarchies. The knowledge segment is a flexible structure for storing background or discovered knowledge about the facts in the database. Its format is designed to facilitate interaction with other data and/or knowledge, and to facilitate the user's understanding of the concepts stored within. Knowledge segments are presented in more detail in Section 4.3.3.

The general design of INLEN is shown in Figure 1, based upon the initial design presented in [Kaufman, Michalski and Kerschberg, 1991a] and updated in [Kaufman, Michalski and Kerschberg, 1991b; 1991c; Michalski et al, 1992]. As is depicted in Figure 1, the INLEN system consists of a relational database, a knowledge base, and a set of operators that may be applied to them. The database stores known facts about a domain, and the knowledge base

contains rules, constraints, hierarchies, decision structures, equations accompanied with preconditions, and enabling conditions for performing various actions on the database and/or knowledge base. This knowledge base can contain not only knowledge about the contents of the database, but also metaknowledge for the dynamic updating of the knowledge base itself.

The purpose for integrating the various learning and discovery capabilities is to provide a user with a set of advanced tools to search for and extract useful knowledge from a database, to organize that knowledge from different viewpoints, to test this knowledge on a set of facts, and to facilitate its integration within the original knowledge base.

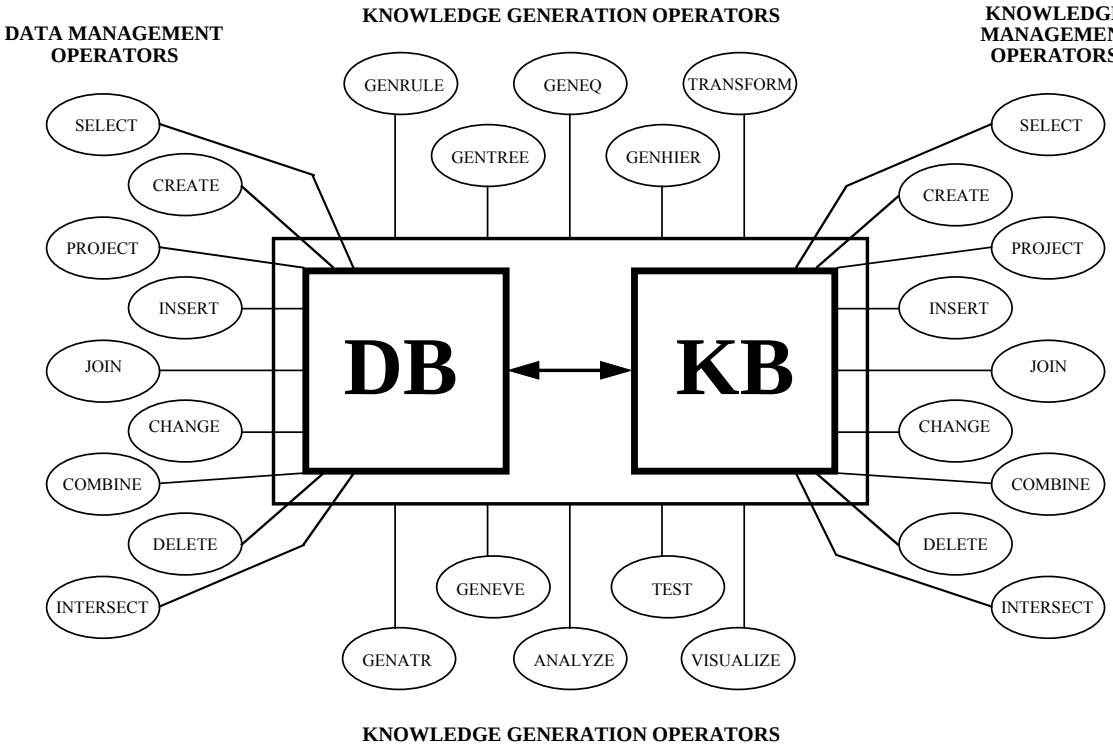

Figure 1. Top-level functional architecture of INLEN

These tools are designed to complement one another, and to be capable of performing many types of learning. For example, different operators can be employed to learn a set of rules from examples (empirical induction), generalize a descriptor or a set of objects (constructive deduction or induction), hypothesize explanations for events in the data based on rules in the knowledge base (abduction), speculate on unknown attribute values of an object based on known values of similar objects (analogical reasoning), and suggest unknown attribute values by employing rules or formulas in the knowledge base (deduction).

INLEN employs three basic sets of operators: *data management operators* (DMOs), *knowledge management operators* (KMOs), and *knowledge generation operators* (KGOs). The data management operators are standard operators for accessing, retrieving and manually altering the information in the database. Thus, they operate on the relational tables in the system. The knowledge management operators perform analogous tasks on the knowledge base, in situations in which manual input, access or adjustments are required. The knowledge generation operators, on the other hand, interact with both the data and knowledge bases.

They take input from both of them and invoke various machine learning programs to perform tasks such as developing general rules from facts, determining differences between groups of facts, creating conceptual classifications of data, selecting the most relevant attributes, determining the most representative examples, and discovering equations governing numeric variables. The results of KGOs are stored as knowledge segments. Figure 1 shows the ten major classes of KGOs [Michalski et al, 1992b]. These classes and their individual members will be described briefly below.

The knowledge generation operators perform inferences on data and knowledge segments in order to create new or better knowledge. As part of their function, the KGOs also include implicit primitives to handle the retrieval of inputs and the placement of their results into the data and/or knowledge base. In general, a KGO takes one table from the database and one or more knowledge segments from the knowledge base, and generates one or more new tables and/or knowledge segments. This knowledge discovery may use either an incremental or a batch mode; in the former case, the emphasis is on improving, refining, or adjusting the strength of existing knowledge, while in the latter case, wholly new knowledge is being discovered from facts and/or other knowledge.

Cataloguing the problems for which each operator is useful brings to light the justification for integrating such a large set of operators, namely that each of these operators is used for a different kind of common data exploration task. Along with the descriptions of the operators, their uses are listed.

The KGOs have been grouped by the types of output they generate (see Figure 1.) Those whose primary output consists of rules are separated from those that generate sets of tuples, for example. Many of these operators are similar to or based on existing programs; the research on which these methods are based is cited below. Some of these operators have yet to be implemented in INLEN; details of the current implementation status may be found in Chapter 5. Section 4.2 contains a complexity analysis of the implemented operators.

GENRULE: Generate Rules

Operators in the GENRULE class take some form of data and/or knowledge as an input, and return a set of decision rules induced from the input examples. Among the GENRULE KGOs are the CHARSET and DIFFSET operators. CHARSET (Characterize Set) determines a description characterizing a class of entities that describes all of the examples in the input group in as much detail as possible. DIFFSET (Differentiate Set) induces general rules that encapsulate the differences between the primary set and the other classes. These operators are applicable in situations in which a classified data table is available and the user wants to find the defining characteristics of a class (in the former case) or to generate simple rules to distinguish between classes (in the latter case). In the initial implementation of INLEN, AQ15 [Michalski et al, 1986] was used to perform these functions. The incorporation of AQ15 in INLEN led to the development of AQ15c [Wnek et al, 1995], a version of which resides on INLEN.

GENTREE: Generate Decision Trees

The GENTREE operators output knowledge in the form of decision trees or, more generally, decision structures. EVENTREE (Event to Tree) uses events in a relational table as input, and generates a tree for classifying the input events. RULETREE (Rule to Tree) organizes a

set of decision rules into one or more trees. EVENTREE is based on ideas implemented in the C4.5 program for generating decision trees from examples [Quinlan, 1990] and ASSISTANT [Cestnik, Kokonenko and Bratko, 1987]. The RULETREE operator utilizes the AQDT methodology described in [Imam and Michalski, 1993; Imam, 1995; Michalski and Imam, 1997]. These operators provide a mechanism for determining general-purpose or task-oriented specifications of procedures to determine the class of an object. As of this writing, work has begun on incorporating GENTREE operators into INLEN.

GENHIER: Generate Hierarchies

The GENHIER operators conceptually classify an input set of tuples, rules, equations, etc. One of these, the CLUSTER operator creates a logical division of the input objects into two or more groups. In addition to the generated hierarchies, it determines a set of rules characterizing the created groups. CLUSTER uses the conceptual clustering algorithm from the CLUSTER/2 program described in [Michalski, Stepp and Diday, 1981; Michalski and Stepp, 1983; Stepp, 1983, 1984]. This operator is applicable when the goal is to find useful ways to classify a set of objects to which no classifications have yet been applied.

TRANSFORM: Transform Knowledge Segments

The TRANSFORM operators perform basic inferential transformations on knowledge segments, hence both the primary inputs and outputs are knowledge segments of the same type, typically decision rules. There are two pairs of inverse operators: ABSTRACT and CONCRETIZE, and GENERALIZE and SPECIALIZE, in addition to IMPROVE, an operator that improves knowledge by giving it new examples to learn from. The first four of these operators are based on the inferential theory of learning as described by Michalski [Michalski, 1990, 1993]. The IMPROVE operator applies incremental learning to improving knowledge. One example of implemented incremental learning is as a component of the AQ15 program [Michalski et al, 1986]. The purpose of each of these operators is to improve existing knowledge; in the case of the IMPROVE operator, knowledge is reconciled with newly available information, and in the case of the other operators, knowledge is manipulated into a form of higher utility for the user. The IMPROVE operator is implemented in the current version of INLEN, as are abstraction/concretion in the form of structured attributes.

GENATR: Generate Attributes

The purpose of the GENATR operators is to modify the representation space in such a way that it will facilitate the generation of high-quality results by other operators. These operators map relational tables to relational tables whose rows are the same but whose columns have been changed, either by the addition of new attributes or by the removal of old ones. SELATR (Select Attribute) determines the attributes in a relational table that are most relevant for differentiating between various classes of objects, and produces a reduced table that retains only those variables chosen by the operator, thereby increasing the efficiency of other learning operators. CONATR (Construct Attribute) applies mathematical or logical operators specified in its arguments in order to combine variables into useful composites. These operators are based on existing programs. SELATR employs a modified version of the PROMISE algorithm [Baim, 1982]; the algorithm and abilities of the operator are described in Section 4.3.4. CONATR incorporates the capabilities of the AQ17 program [Bloedorn, Wnek and Michalski, 1993]. Data-driven constructive induction [Bloedorn and Michalski,

1991] appears to be an especially suitable method for the attribute construction tasks described here. The INSIGHT methodology [Ribeiro, Kaufman and Kerschberg, 1995] is used for constructing useful knowledge bases from distributed relational tables.

GENEVE: Generate Events

The GENEVE class covers a wide variety of operators that generate a new set of tuples, selected according to some criteria of desirability. SELEVE (Select Event) determines the examples (objects) that are the most representative of the examples contained in input relational tables or the most desirable according to some selection criteria provided by the user in order to generate a data set better suited for subsequent discovery tasks in terms of speed and quality of output. PREDVAL (Predict Value) speculates on likely values for unknown attributes of incomplete or hypothetical data elements, using existing knowledge to reason about the incomplete tuples, plausible reasoning to learn from examples of similarly behaving events, and sequence characterization techniques to extrapolate from changes in a linear domain. SELEVE uses the ESEL methodology, described in [Michalski and Larson, 1978; Cramm, 1983]. PREDVAL is the subject of research to build upon the foundations set by the performance elements of programs such as APPLAUSE [Dontas, 1988], AQDT [Imam and Michalski, 1993; Imam, 1995], and AURORA's advisory module.

ANALYZE: Analyze Data

The ANALYZE family of operators return knowledge in the form of numerical weights that describe the elements in the database. These numbers can represent logical or statistical relationships. The operator RELKS (Relate Knowledge Segments) discovers relationships such as inclusion, disjointedness, correlation, generalization and abstraction within a set of knowledge segments. GENSTAT (Generate Statistics) performs a statistical analysis of the data in order to determine its various statistical properties. The incorporation of RELKS into INLEN is a subject of current research, with partial implementation through INSIGHT [Ribeiro, Kaufman and Kerschberg, 1995], while the implemented GENSTAT operator incorporates existing statistical methods to generate its output. These operators are applicable when the current discovery goal is to find underlying relationships between the data attributes and between individual pieces of discovered knowledge.

TEST: Test Knowledge

The TEST operator determines the performance of a ruleset on a set of examples by testing the input knowledge segments' predictive accuracy with regard to a set of input examples (specified in a relational table). TEST uses the ATEST methodology [Reinke, 1984] for analyzing consistency and completeness in rules, and generating confusion matrices.

VISUALIZE: Visualize Knowledge

The VISUALIZE operator presents data and/or knowledge to the user in a convenient, easy-to-understand format. VISUALIZE uses the DIAV methodology [Wnek, 1995] for diagrammatic visualization.

In summary, each of these operators is specialized for a unique data analysis task. By integrating such operators, one can create a powerful system that can cope with a wide array of data exploration problems. Hence a major research issue is the development of a means of

integrating these components into a synergistic package. The ways in which INLEN addresses this problem are addressed in the following section.

4.2 Applicability and scalability of the knowledge generation operators

Given the data size constraints enforced by the prototype versions of INLEN, an important feasibility question regards its application to large databases. This section discusses the computational complexity of the knowledge generation operators already implemented in order to demonstrate whether such programs can realistically be applied to larger databases.

4.2.1 Learning rules from examples

Many of the implemented operators—learning of characteristic or discriminant descriptions, rule optimization, incremental learning, and rule testing—are performed using a version of the AQ15c program [Wnek et al, 1995], a C-based reimplementation of the original Pascal language AQ15 program [Michalski et al, 1986]. The worst-case computational complexity of the original AQ15's algorithm for learning one conjunctive decision rule was analyzed by Paliouras (1993) and found to be $O(|M| \, |A|^2 \, |V_{max}| \, |E_-|^2)$, where M is the user-defined width of the beam search used during the generation of rules, |A| is the number of input attributes, $|V_{max}|$ is the maximum domain size for any attribute, and $|E_-|$ is the number of training examples in the negative class. It can be assumed that in a very large database, $|E_-|$ will dwarf |A|, $|V_{max}|$ may be kept small through the abstraction of the attribute domain, and |M| (whose default value is 1 in INLEN-3, sufficient for most problems) can be set to as low a value as necessary. Hence, the main factor in the complexity of learning a conjunctive decision rulein large databases will be the number of training examples applied to the learning session (which may be controlled through example selection); the processing time will vary with the square of this number.

This complexity value presented by Paliouras is overly pessimistic. It assumes that (1) the extension-against procedure will only remove one negative example from a cover at a time, and perhaps more importantly, (2) One of the most expensive rule evaluation criteria (coverage ratio of positive to negative examples, coverage ratio of newly covered positive examples to negative examples, or information gain ratio -- those that require the testing of examples of all classes for coverage) will be included in the rule evaluation function selected by the user. In reality, these evaluation criteria are rarely used. None of them are part of the default AQ15c Lexicographic Evaluation Function, nor are they even made available to INLEN users.

Another change in the complexity level calculated by Paliouras was caused by the conversion of AQ15 from Pascal to C. In the Pascal version, attributes and values were stored as Pascal set structures, possibly but not necessarily sharing a common set structure. As a result, operations such as checking whether a rule covered an event would require calls to Pascal set functions for each attribute, often bringing an |A| element into complexity calculations.

In the C implementation of AQ15, attributes and values are stored in a single dynamically allocated bit string structure. Coverage checks can thus be performed by performing set intersection and comparison operations, whose complexity will be dependent on the length of the bit string, which is based on the total number of legal values of all the independent attributes.

Hence, in the case of INLEN, the worst case complexity of the rule evaluation step is not $O(|STAR|\ |E_-|\ |A|)$ as stated in Paliouras' worst-case analysis of AQ15-Pascal, but rather $O(|STAR|\ |E_+|\ |V_{tot}|)$, where $|E_+|$ is the number of positive examples of the class and $|V_{tot}|$ is the total number of attribute values. Given that |STAR|, the size of a partial star, has an upper bound of |M| |A|, the complexity of this stage in INLEN is bounded by $O(|A|\ |E_+|\ |V_{tot}|)$.

Following Paliouras' reasoning, the complexity of finding the best rule for a seed example in AQ in INLEN is thus $O(|E_-|\ |A|\ (|A|^2\ |V_{max}| + |E_+|\ |V_{tot}|))$, which will reduce to:

$$O(|E_-|\ |A|\ |E_+|\ |V_{tot}|).$$

The differences between the AQ15 and AQ15c algorithms should not cause major changes in this result. One of the new features of AQ15c was the implementation of strict integrity constraints in the application of the extend-against operator with regard to structured attributes. Three sets of checks are made to ensure that: (1) for the value of the structured attribute in the negative example being extended against, the final extension will not include any value that is an ancestor of this value in the generalization hierarchy; (2) for the value in the negative example being extended against, the final extension will not include any value that is a descendant of this value in the generalization hierarchy; (3) for the value in the positive seed example, the final extension will include any value that is a descendant of this value in the generalization hierarchy. To illustrate these principles, consider a hierarchy of animals. Given the positive example of a concept *x may be any cat*, and the negative example, *x may not be any terrier*, it is clear that (1) the concept generalization *x may be any dog* is inconsistent; (2) the concept generalization *x may be any fox terrier* is inconsistent; (3) the concept generalization *x may be any siamese cat* is consistent. Each of these three checks involves examining internal nodes of the hierarchy repeatedly until there is no further change; this may require any number of iterations from 1 to the maximum depth of the hierarchy. Since the maximum depth of the hierarchy, the number of internal nodes, and the set operations involved are each bounded by the number of values of the attribute, the complexity of the preprocessing has a worst case performance of $O(|V_{max}|^3)$. Hence, given that $|A_s|$ is the number of structured independent attributes, the complexity of the extension against operator increases from the $O(|A|\ |V_{tot}|)$ for nominal attributes to $O(|A|\ |V_{tot}| + |A_s|\ |V_{max}|^3)$ in AQ15c, and similarly, the overall AQ complexity increases from $O(|E_-||A|\ |E_+|\ |V_{tot}|)$ to $O(|E_-|\ |A|\ |E_+|\ |V_{tot}| + |E|\ |A_s|\ |V_{max}|^3)$.

Given that |E| will be far greater than $|V_{max}|$ in large databases, the use of structured attributes should not have a significant effect on the processing time.

4.2.2 Optimizing rules according to some criterion

INLEN's rule optimization operator uses rules instead of examples as the input; the goal of the operator is to improve the rules by tuning them based on user-defined criteria. It uses the same methodology as rule learning from examples in AQ15c, and as such has a complexity of:

$$O(|R_-|\ |A|\ |R_+|\ |V_{tot}|)$$

where R is the number of input rules.

4.2.3 Improving rules through incremental learning

Unlike rule learning and optimization, rule improvement through incremental learning originally used a "zero-memory" algorithm (one that did not make any use of prior training examples) that involved a preprocessing step when learning rules for each class that adds to the complexity of the entire operator. In this step, the input rules were compared with the new training examples in order to ensure that inconsistencies were not passed to the rule generation module. If an inconsistency was found, the cover genration procedure would be used to fragment the rule into consistent components.

Unfortunately, this procedure was performed without any MAXSTAR constraint. By doing so, the resulting fragments would be guaranteed to be complete and consistent. However, for each rule in the positive class, all inconsistent rules might be fragmented (by the extend-against function) into a number of hypotheses up to the number of attributes in the domain). Hence, by the time of the final iteration, the number of rules in consideration could approach $O(|R_0| \, |A|^{|R+|})$, where $|R_0|$ is the initial size of the rule base, $|A|$ is the number of attributes, and $|R+|$ is the number of positive input hypotheses for the class under consideration. The only thing holding this number down was the *absorb* procedure, which removed from the intermediate cover any potential rules that were subsumed by others. This is an extremely expensive step, requiring $O(|\text{Intermediate Cover}|^2 \, |A|)$ time in the worst case [Paliouras, 1993]. Substituting in the worst case rule base size determined above, the size of this step could approach:

$$O(|R_0|^2 \, |A|)$$

It was thus concluded that in any complex domain with many attributes and more than a few rules, the incremental learning facility provided by AQ15c was not a viable option.

One way to avoid this problem would be to ensure that the rule sets for different classes do not intersect. However, this is not a default option for rule learning, and may not be desired by the user (the generated rule will tend to be more complex, less intuitive, and dependent on the ordering of the attributes in the dataset).

As a result, INLEN was modified to maintain a full memory of prior training examples in its knowledge base, so that it might employ the full-memory incremental learning algorithm described in [Reinke, 1984]. In the worst case, the intermediate cover would consist of as many rules as there were prior training examples. As a result, the preprocessing step to modify an input rule would be reduced to a complexity of:

$$O(|E_0| \, |E_{-c}| \, |A| \, |V_{tot}|)$$

where $|E_0|$ is the number of prior examples that satisfied the input rule (this serves as a MAXSTAR bound, since all retained specializations of the rule will cover at least one such example, and at least one uniquely), and $|E_{-c}|$ is the number of new negative examples of the class that satisfy it. After this preprocessing, the incremental learning process may be slightly simpler than batch learning with the same set of examples, since examples covered by preprocessed rules for their class will not require any further learning to generate a classification rule that covers them.

4.2.4 Testing consistency and completeness of rules

The rule testing procedure in AQ15 in INLEN works as follows: For each testing example, for each rule, test for strict coverage. If so, determine degree of match. If there was no strict coverage, test each rule for flexible match. Testing either for strict or flexible match requires $O(|A| \ |V_{max}|)$ time, where $|A|$ is the number of attributes and $|V_{max}|$ is the size of the largest attribute domain. Hence, the total complexity of this operator is:

$$O(|E| \ |R| \ |A| \ |V_{max}|)$$

Thus, increases in the size of the database or the number of examples tested should not slow this operator unacceptably.

4.2.5 Selecting the most useful attributes

The attribute selection algorithms used in INLEN work as follows: For each attribute, evaluate its promise level by (1) projecting the attribute's value for each training example; (2) for each value, compute the number of examples of each class having that value. Step (1) has a complexity based on the number of classes multiplied by the square of the number of examples in the class being examined (the examples are ordered by a simple insertion sort). Step (2) has a complexity based on the number of classes multiplied by the number of values of the attribute multiplied by the number of examples in the class being examined. Thus, if $|E_{max}|$ is the number of examples in the most prolific class, $|V_{max}|$ is the maximum number of values for any attribute, $|C|$ is the number of classes, and $|A|$ is the number of attributes, the worst-case complexity of this operator is $O(|A| \ |C| \ (|V_{max}|^2 + |V_{max}| \ |E_{max}|))$ which for a large number of examples, approaches:

$$O(|A| \ |C| \ |V_{max}| \ |E_{max}|)$$

4.2.6 Using the advisory module to predict values

When INLEN's advisory module is used to predict values for missing data, the inference process involves querying the value of an attribute, updating the degree of match for each rule in consideration, and updating each class's degree of confidence based on these values. The second of those steps may involve a detailed search of the value-rule cross-reference tables, which may have a complexity based on the number of references to the attribute in the rule base (bounded by the product of the number of rules and the number of possible values of the attribute) multiplied by the number of rules. Hence the worst-case complexity of this process is:

$$O(|A| \ |R|^2 \ |V_{max}| \ |C|)$$

4.2.7 Generating a statistical report

The statistical report generator under development produces four tables: means and modes of all attributes for all decision classes, variances of all numeric attributes for all decision classes, covariances between all numeric attributes, and correlations between all numeric attributes. Mode calculation for non-numeric attributes requires $O(|C| \ |E| \ |V_{max}|)$ time. Mean and variance calculation for numeric attributes requires $O(|C| \ |E|)$ time. The calculation of the covariance between two numeric attributes requires $O(|E|^2)$ time, and given the covariance and variance values, the calculation of correlations can be done in constant time. Thus, the generation of these tables requires $O(|A_{nom}| \ |C| \ |E| \ |V_{max}| + |A_{num}| \ |C| \ |E| + |A_{num}|^2 \ |E|^2 + |A_{num}|^2)$

time, where $|A_{nom}|$ is the number of non-numeric attributes and $|A_{num}|$ is the number of numeric ones. In large databases, this reduces to:

$$O(|A_{nom}|\,|C|\,|E|\,|V_{max}| + |A_{num}|^2\,|E|^2)$$

4.2.8 Scalability summary

Most of the operators implemented in INLEN should retain satisfactory functionality as database size is scaled up. The performance of AQ-based rule learning may degrade, but that can be worked around in several ways. One way is to learn from a carefully selected representative example subset. An operator for performing that task is planned for implementation in future versions of INLEN.

Another option is to learn through incremental learning. The zero-memory algorithm showed an unacceptable level of degradation, so it was replaced by a full-memory method whose computational complexity is similar to that of batch learning, with some storage overhead in retaining prior examples. By using such a facility, a knowledge base can be generated in learning episodes of acceptable size; as the total amount of training data increases, the knowledge will approach a true representation of the dataset, and the incremental learning episodes will become shorter (since much of the new data will already be described by the accumulated knowledge. This method also has the advantage of filtering out noise, as the incremental preprocessing step removes rules that appear spurious.

4.3 Problems in knowledge representation and integration

This research addresses a range of theoretical and methodological issues that affect symbolic-oriented multistrategy knowledge discovery. These issues include:

- The evolution of knowledge representations that best serve both the user and the automated discovery engines, including representations of hierarchical data and quantization techniques for continuous data. In complex hierarchies, it is often the case that the learning goal will involve intermediate levels of generalization, unlike learning tasks in less structured domains. The choice of a quantization schema is as vital as any feature selection scheme for making the patterns in the data available for capture.
- The development of a method for the integration of symbolic and statistical learning techniques with a database system and a knowledge base to create a synergistic system for supervised and unsupervised learning.
- The creation of a new knowledge representation schema that allows the system to operate on components of knowledge and data as single, linked units. This representation must support a great variety of knowledge generation operators that differ in the types of input they require and the types of knowledge they generate. The links between knowledge and data may include information about the applicability and typicality of the knowledge, the historical relationships between the knowledge and the linked data, etc.
- A method for selecting a feature set that will be useful for a particular learning task. Not only does this task involve selecting the features most relevant to the concepts being learned, but one must also consider the nature of the discovered knowledge and how it will likely be used. This also biases the value of a given feature set.

In the following subsections, these problems are discussed.

4.3.1 Quantization of continuous data

Traditionally, machine learning programs have been designed for the laboratory environment. Symbolic rule learning systems are built to work very well when classifying based on simple nominal or linear data types. Databases, on the other hand, often contain many unquantized numeric attributes. Learning programs which are designed to perform on attribute domains with a relatively small number of values must reconcile the raw data with their capabilities in some way, ideally using intelligent quantization.

The quantization problem consists of trying to find the view of continuous data best suited for a particular learning task. There are different ways to group the values into discrete ranges, and unless definitive background knowledge is available, the optimal organization for a particular discovery problem may not be apparent. Even when numeric values have been quantized into ranges by a domain expert, the expert-generated abstraction schema may not be optimal for the learning task (e.g., Karni and Loksh, 1996). Furthermore, a particular set of ranges may be useful for one learning task, but irrelevant to another.

Several endeavors in machine learning approach the problem of creating the different values to be used for an attribute. Numerical and conceptual clustering (e.g., Sokal and Sneath, 1973, Michalski and Stepp, 1983; Fisher, 1987) are essentially approaches to the problem of creating an output (classification) variable for some domain. Karni and Loksh (1996) are investigating automated creation of classification hierarchies, i.e., the clustering of given values for nominal non-decision variables. Clearly, the discretization of numeric data is a related effort. Wrobel (1991) investigated the development of a methodology for discretizing sensory inputs into meaningful attribute ranges.

INLEN performs automatic discretization of numeric data through aggressive use of the ChiMerge algorithm [Kerber, 1992]. With this method, neighboring distinct values of the attribute found in the data are merged into single ranges based on a χ^2 analysis of the classification of the values. When the classification patterns of adjacent ranges are statistically dependent on one another, the ranges are combined into one.

One important consequence of this algorithm is that the grouping of values into intervals will depend on the classification of the input data. Specifically, given a new learning problem from the same data set, the training data can be grouped much differently into classes, and thus the set of ranges of a given attribute most likely to generate concise and useful knowledge may be much different. For example, given an auto insurance customer database, there is likely to be little correlation between the set of accident-prone clients and the set of customers who are likely be interested in purchasing some other service offered by the company. It is quite possible that the ranges into which the driver's ages are divided that are most useful for the first learning task are not so appropriate for the second one. To combat this problem, ChiMerge is called upon to recompute the ranges for each numeric variable whenever new sets of data are added or a new discovery problem is specified.

A limitation of the ChiMerge methodology is that it focuses on one attribute at a time, and may therefore overlook instances when a decision is dependent on some combination of attributes, while having little correlation with any one of them. A constructive induction

approach may expose utility among combinations of numeric attributes; combining ChiMerge with constructive induction remains an area for further research.

Another limitation is that the methodology is dependent on the classification of input training examples, and that it therefore can not be used to quantize an output variable (since that would depend on its already having been divided into classes). There are several ways that an output numeric variable can be discretized into a hierarchy of ranges. A domain expert can suggest ranges, ranges can be determined based on some other output variable, conceptual clustering can be used, etc.

4.3.2 Structured variable representation

Along with nominal and linearly ordered attribute domains, INLEN supports learning with complex structured data types. There are many ways of structuring data to convey information about their interrelationships (Jonassen, Beisner and Yacci, 1993). In INLEN a *structured attribute* is structured by organizing its values into a generalization hierarchy. A leaf node in the structure represents a member of a subclass of the classes represented by each of the nodes above it in the hierarchy.

An INLEN user can not only define a data attribute's values to follow some generalization hierarchy, one can also define nodes within the hierarchy as *anchor nodes* - important foci for learning, generalization and specialization [Kaufman, 1996; Kaufman and Michalski, 1996b]. The justification behind the selection of such nodes is that humans tend to assign different levels of significance to nodes in a generalization hierarchy.

Cognitive scientists speak of *basic* level nodes within a generalization hierarchy whose children share many sensorially recognizable commonalities, an idea first put forth by Rosch et al. (1976). Other factors that help to characterize a node's utility compared to those at higher or lower levels of abstraction are concept typicality (how common are the features of this concept among its sibling concepts), and the context in which the concept is being used [Klimesch, 1988; Kubat, Bratko and Michalski, 1997]. Each of these factors affects the usage of a particular concept level in making generalizations. For instance, a red delicious is an apple, which is a type of fruit, which is a kind of food. In everyday usage, when we encounter a red delicious, we are likely to think of it as an apple or a piece of fruit, but we are less likely to think of it as a red delicious or simply as a piece of food.

By encoding the significance of the nodes into the knowledge representation of a discovery system, the system can present discovered knowledge that focuses on these levels of abstraction when possible. We would typically prefer to see the classification rules "A living thing belongs to Class 1 if it is a dog" or "A living thing belongs to Class 1 if it is an animal" instead of "A living thing belongs to Class 1 if it is a vertebrate" or "A living thing belongs to Class 1 if it is a member of the carnivore order."

INLEN's approach is to allow users to define "substructures" as part of an attribute's generalization hierarchy. The root of a substructure tree is a node to be regarded as an anchor node in the attribute's structure. A substructure is automatically "grafted" into the parent structures; since the root node of the substructure also appears as a node in the main structure, its descendants can be added into the hierarchy as a whole. This procedure is recursive -- a node within a substructure can be the root of another substructure.

For instance, if an the generalization hierarchy for an attribute *type-of-furniture* has an internal node **chair**, we could convert **chair** into a leaf, then create a substructure called **chair**, consisting of the hierarchy of types of **chair**s. We can thus view the substructure nodes (**chair** and all its descendants) as a lower part of the furniture structure. Most importantly, INLEN includes integrity constraints so that the values for the subvariable **chair** only have meaning if the substructure is activated by setting the value of *type-of-furniture* to **chair**.

To utilize substructures, a knowledge discovery operator combines all of the segments of the structure into a larger, single unit before attempting generalization, yet information about the substructures is maintained, and their roots are taken to be "anchor nodes" in the overall structure. In the case of the AQ rule learning operator, for instance, rule generation is done in two phases – the discovery and ranking of hypotheses consistent with the input data, followed by a postprocessing phase, in which the level of generality of the best discovered hypothesis is adjusted according to user preferences. In structured domains in which anchor nodes are present, the first phase operates identically to the case of nominal or simple-structured domains. The postprocessing is, however, handled differently. Some generalization or specialization may now occur, even if the user specified maximally general or specific rules, in order that when possible the anchor nodes appear in the output, rather than more general or specific values. For example, an output characteristic (i.e., the user requested maximal specification) output rule in the furniture domain would be "x = **chair**" rather than "x = **recliner**" if a training example of the target class was a **recliner**, and none of the examples of other classes were **chair**s. Similarly, a discriminant (maximally general) rule would be "x = **chair**" rather than "x = **sitting-furniture**", even if the training data were consistent with the more general hypothesis.

Another important feature in INLEN's structured data representation is its ability to work with multiple views of the data [Kaufman, 1996], through an architecture similar to overlapping semantic maps [Jonassen, Beisner and Yacci, 1993]. Consider an application in which a user is trying to classify personal computer systems, and one of the attributes is the model of the monitor being used. Computer monitors themselves can be classified based on various features, such as their manufacturers and their size. We may not know prior to a learning task which classification view will provide the most useful knowledge. Furthermore, more than one view may generate the best decision rules. A decision rule, for example, may include the condition that the system should have either a 17" monitor or one manufactured by NEC. INLEN's generalization engine automatically selects the optimal (according to user criteria) representation of the knowledge.

As is the case with generalizing to anchor nodes, the choice of view in which to present the output data is made during the postprocessing stage. Different generalizations of the characteristic rule are attempted in a search for consistent ones which cover many of the training examples. The most useful generalizations, regardless of the views of the attribute that they are based on, become part of the output rule.

4.3.3 Knowledge representation

A knowledge base ideally serves many purposes. It contains declarative or procedural knowledge on which decision-making can be based. It contains equations for reference and deductive reasoning. It contains organizational background knowledge to define

relationships among different concepts. And it contains references to examples of concepts, whether as representative, exceptional, or the basis for the generation of some other knowledge.

INLEN's knowledge base and the programs that make up INLEN's knowledge discovery operators all have their own special requirements with respect to input and output information. What is vital for one module may not be needed by another. Hence a goal of this research has been to design a knowledge base that can conveniently hold the information needed by all of these components and to engineer an internal interface that will facilitate communication among the various modules.

As such, the design of the knowledge segment is critical. The knowledge base must be able to integrate knowledge represented as rules, equations, procedural specifications, representative example sets, and combinations of these components. Structures need to be designed that can represent the information stored by each of these knowledge types and support interactions between them.

At the same time, the knowledge must be stored in such a fashion that the various operators can extract what they need from it. Furthermore, it should be clearly presentable to the user. A goal of this research is to design knowledge representations that will facilitate the knowledge discovery process.

The contents of a knowledge base can be divided into subunits, each describable by a set of pertinent statistics [Ribeiro, Kaufman and Kerschberg, 1995]. The application of an operator (an individual learning session) can be associated with an input data set, the time at which it was run, etc. as well as a set of output knowledge units describing different classes of events (e.g., rulesets, portions of decision trees, families of equations.) Each of these output units can be associated with the class of objects it describes, statistics on the training sets associated with it, and the individual pieces of knowledge that comprise the unit (e.g., rules, individual decision procedures, single equations). The knowledge units can be associated with the training examples that satisfied them and aggregate numbers thereof, and also sets of the subunits (e.g., individual conditions) that comprise them. The knowledge corresponding to these subunits can be associated with the statement of the condition and various informational weights such as the number of positive and negative examples of the class in question that satisfy the condition.

The concept of a *knowledge segment* has evolved from this representation. A knowledge segment in INLEN is a unit of knowledge, or a cohesive combination of such units; in the rule-based example above, each condition and its associated weights comprise a knowledge segment; these can be combined into a larger knowledge segment representing a rule and its associated statistics and examples. Similarly, larger knowledge segments can represent an entire cover (i.e., a complete set of rules with a shared consequent).

Guided by the idea of a knowledge segment, a rule in INLEN can be represented by the following syntax [Kaufman and Michalski, 1997a] based on the VL1 variable-valued logic knowledge representation [Michalski, 1973]:

Referent:	An attribute, or several attributes linked by (internal) conjunction or disjunction

Relation Symbol (Rel):	=, <>, >, <, >=, or <=
Reference:	A legal value of the attribute(s) comprising the associated Referent, an (internal) disjunction, or a range of such values
Condition:	[<Referent> <Rel> <Value>]
Condition Weights:	Numerical information indicating the number of datapoints (training examples) of the positive and negative classes satisfying a given Condition
Weighted Condition:	<Condition> <Condition Weights>
(Weighted) Description:	A conjunction of (Weighted) Conditions
Rule Weights:	Numerical information characterizing a Description, e.g., number of training examples satisfying the Description
Example Keys:	Identifiers of individual training examples satisfying an associated Description
Rule:	An implication <Description> => <Condition> annotated by Rule Weights and Example Keys
Cover:	A set of Rules with the same consequent Condition
Ruleset:	A set of Covers whose consequent Conditions share the same Referent attribute
Session:	A learning session, defined by a timestamp, input dataset, and output Ruleset

Figure 2 shows an example of a knowledge segment structure for a set of discovered decision rules. The Session knowledge segment includes the timestamp and the names of the decision attributes (concepts) from which knowledge was derived. From this segment emanate links to the different Cover knowledge segments for each decision attribute. These contain the names of the classes and links to the rules that describe them, as well as the number of training examples in the entire class. The rule knowledge structures are associated with the database keys of the examples that satisfy them, the number of training examples that satisfy them, and links to the individual conditions.

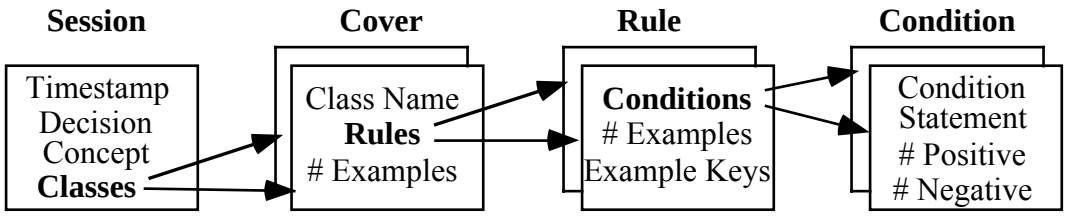

Figure 2. Knowledge segment organization for a decision rule set

Figure 3 shows an instantiation of this architecture - a knowledge segment hierarchy derived from the 1993 World Factbook [CIA, 1993]. The countries of the world were divided into classes based on their fertility rate, with the lowest class describing the 42 countries with fertility rates less than 2. One of the discovered rules differentiating this class from countries with higher fertility rates described 16 countries, including Andorra and Austria. It consisted of several conditions including one, "Birth Rate is between 10 and 20," that described all 42 low-fertility countries and only 20 others.

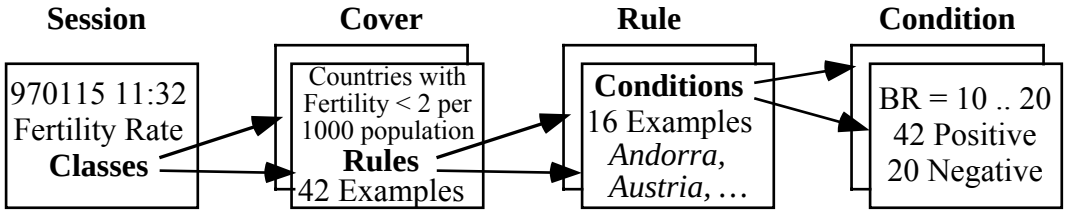

Figure 3. Knowledge segment structure for the decision class Fertility < 2

4.3.4 Selection of a useful attribute set

From within a set of data, some of the data attributes will be more useful to a given learning task than others, although the set of relevant features will change depending on the learning task. A useful capability of a knowledge discovery system is the ability to separate out the attributes most useful to a particular learning task. This will often speed up the performance of that task greatly, with little loss or a gain in output quality.

One effort to achieve such a goal, PROMISE [Baim, 1982], serves as the cornerstone for feature selection in INLEN. PROMISE uses the following algorithm to determine an attribute's likely informativeness:

> Let P = 0. Then for each distinct value of the attribute in the training dataset:
> Let S be the set of examples that have this value.
> While S contains examples from more than one class:
> Find C, the largest class with the smallest (>0) number of examples within S.
> Remove all examples of that class from S and increment P by the number of examples removed from S divided by the total number of examples in class C.

An attribute's utility in discriminating between classes is thus maximized when P = 0, while a feature without any discriminatory value will have P = 1 - (1 / number_of_classes). A simple inversion and normalization operation on P outputs a Promise value of 0 to 1 with higher values implying better discrimination.

An analysis of PROMISE led to the conclusion that within the algorithm described above, all class-value sets would be counted in P except for the smallest class with the largest number of examples within each S. Hence the implementation of PROMISE 2 undertaken during the course of this research streamlined the algorithm:

> Let P = 0. Then for each distinct value of the attribute in the training dataset:
> Let S be the set of examples that have this value.
> Find C, the smallest class with the largest number of examples within S.
> Increment P by the number of examples in S in this class divided by the total number of examples in C.

An attribute's utility in discriminating between classes is thus maximized when P = number_of_values, while a feature without any discriminatory value will have P = number_of_values / number_of_classes. A simple normalization operation on P outputs a Promise value of 0 to 1 with higher values implying better discrimination.

Both implementations base their evaluations of an attribute's discriminatory ability on its overall performance for all values. They are based on expected performance regardless of the input. As such, these value estimates are especially useful for decision-tree-oriented knowledge bases, in which we want to first investigate the features most likely to bring us to a speedy decision. Such attributes may not be optimal in a rule-based representation, in which exceptionally high discrimination for certain values with mediocre performance for others may be far more useful.

As an example, consider the task of distinguishing between upper-case letters of the English alphabet based on properties such as whether parts of them are closed, whether they contain any curved lines, etc. In a rule-based (declarative) representation, the letter Q can be distinguished from the rest of the alphabet by a simple and concise property:

> If the letter has a tail, it is a Q.

By such reasoning, the attribute *has-tail* is extremely useful for rule-based learning, and should therefore be retained in any reduced attribute set. But if the knowledge base is decision-tree-oriented (procedural), the utility of such an attribute is far lower. Presumably, most of the letters encountered will *not* be Qs. Hence, testing the letter for a tail will be a wasted step that only serves to eliminate the possibility of it being a Q, without making any progress in distinguishing between the other 25 letters. It is better to pare down the set of hypotheses more rapidly, and only check for a tail as a last step in cases such as when the set of possible letters has been reduced to O and Q.

INLEN supports this method of attribute evaluation as well as that of PROMISE. In this mode, the evaluation step is modified as follows:

> Let P = 0. Then for each distinct value of the attribute in the training dataset:
> Let S be the set of examples that have this value.
> Find the smallest decision class (in terms of number of training examples) with the largest number of examples within S. Increment P by the number of examples in S that belong to this class divided by the total number of examples *with that value*.

In cases in which a value gives perfect discrimination, preference is given to attributes whose discriminating values are found in more training examples.

4.4 A language for semi-autonomous learning: KGL-1

4.4.1 Why create a knowledge generation language?

Machine learning researchers have moved toward multistrategy learning in response to the limitations of task-specific systems. These systems are designed to call upon diverse learning methods depending on the task at hand (e.g., Michalski, 1991b). Such systems still present some difficulties for a data analyst. One is that we have yet to design or build a control

system that can automatically and accurately select discovery tasks to perform based upon the user's goals. Nor can these programs determine which newly discovered facts will be regarded as useful.

At this stage, the development task is simply to get the discovery system to the point at which it can be a useful assistant for a data analyst, who will remain in charge and direct the search. Anything more is an issue for future research. But many of the tasks in an INLEN-like knowledge discovery toolkit are NP-complete in nature. Even polynomial-time approximations of optimal performance may result in long processes when applied to databases of reasonable size. To require the user to be present both to initiate a learning task and to assist in the post-processing is inconvenient for the human and inefficient for the machine.

There may exist rules of control suitably general that they can be applied to a large proportion of knowledge discovery domains, and hence embedded within a discovery system. Others may be more domain- and task-dependent. Individual users may have their own collection of necessary tasks to perform on the data.

Furthermore, as data analysis tools become more sophisticated, it is becoming increasingly important that tools be developed for the analysis of the discovered knowledge. Imielinski and Mannila (1996) speak of "rule query languages," through which a user can access objects in a knowledge base, as being a critical element in the future evolution of data mining systems.

It is therefore important that a user can articulate desires and expertise to the knowledge discovery system, beyond serial invocations of the basic operators, in such a way that the machine can perform independently for longer sequences of actions. One approach is the creation of a high-level language, readable by the program, that allows the user to create a plan for guiding the system through various contingencies. Such a language, if versatile enough, could allow the user to write macros and/or complex programs to be executed once, periodically or on the occurrence of some event such as an infusion of new information into the database.

These factors have motivated the development of KGL-1, a prototype high-level language for multistrategy learning that has been implemented in INLEN 3.0. Similarly, in an attempt to bridge the gap between data and knowledge querying, an extension to SQL called M-SQL [Imielinski, Virmani and Abdulghani, 1996] has been developed, with the capability of querying for certain types of rules and invoking a rule-generating operator. KGL differs from M-SQL in that it is able to call upon many different types of knowledge generation operators, and also in that it is designed to be less tightly coupled with SQL (although an SQL interface for invoking data management operators is planned) and more closely resembles a programming language than a query language.

This research attempts to identify metaknowledge for operator control that can take a form such as "If a certain **condition** is satisfied by a dataset or the knowledge currently associated with that dataset, apply the following sequence of **operators** with the following **parameters**." For example, in a domain in which the database is continuously being updated, a rule may be used such as, "If more than 3% of the records in the database have been modified or added since the last application of this rule, test the knowledge base for

continued applicability to the new data. If its degree of match falls below a 90% threshold, apply incremental learning to improve the knowledge, and analyze the records that do not match the original rules to see if they share any interesting common bonds."

In general, the requirements of such a data analysis language are:

- Invoking different types of programs as single operators for learning and knowledge discovery.
- Looping and branching abilities similar to those found in programming languages.
- Discrimination among the different attributes in the database. The user should be able to classify the attributes into groups based on importance, type, number of legal values, etc. With such a feature, the user may specify "partial grand tours" of the database, such as "Use the Set Differentiation operator to generate a full set of classification rules using all nominal attributes as decision variables."
- Discrimination among the different rules, rulesets, decision structures, etc. that make up a domain's knowledge base. The user should be able to select rules based on complexity, strength, typicality, events they cover, etc.
- Data-driven control strategies, triggered by changes to a database beyond a given threshold level. Among the patterns that must be detectable are missing values and conflicts with the existing knowledge base.
- Knowledge-driven control strategies, triggered, for example, by the discovery of especially strong patterns or exceptions. The program must be able to examine a piece of discovered knowledge and identify some of the attributes of the knowledge itself.

4.4.2 Specification of KGL-1

The primary goal in designing the knowledge generation language KGL is the ability to interact with INLEN data and knowledge bases to a degree at least equivalent to INLEN's capabilities in interactive mode. Simplicity of implementation was often chosen over elegance of the language. As such, KGL-1 is a command-based language whose constructs take on the form <verb> <object> <parameters>.

The concepts upon which the language is based include the condition-rule knowledge structure described in Section 4.3.3, as well as the data table organization and knowledge generation operators supported by INLEN. The following commands are supported in KGL-1:

assign: Assignment operator - can take either of two forms: *assign* <*variable*> <*expression*> or <*variable*> = <*expression*>. In either form it assigns to the variable the result of the evaluation of the specified expression. The expression may be a string, a constant, the name of a variable, a function (possibly based on the contents of one or more knowledge segments), or an expression combining any of those forms with numerical, relational, logical, or functional operators.

begin/end: Delineators of a block of commands

display: Taking the form *display <expression list>*, this command writes the values specified by the expression list upon the screen.

do: Taking the form *do <operator>(<argument list>)*, this command invokes a knowledge processing operator, using the given set of arguments.

for: Perform a block of commands a certain number of times. It takes the form: *for <variable> = <min> to <max>* followed by the block to be repeated.

forall: Perform a block of commands for all given objects meeting a certain criteria. It takes the form: *forall <object-type>(<criteria>)* followed by the block to be repeated. For instance, to operate on all rules for decision class Age that cover at least 50 examples, the command would be: forall rules(Age, num_covd >= 50).

if: Tests the condition following the if for veracity. If it is true, perform the next block. Optionally, that block may be followed by an *else* command to indicate a block to be performed if the condition is not true.

open: Selects the active data table.

print: Identical to the display command, except output is written to an output file.

while: Tests the condition following the while for veracity, and repeatedly performs the next block as long as the condition remains true.

The knowledge generation operators that may be currently be invoked by KGL calls are listed below. It should be noted that the descriptions of the commands presented here are functional ones; the KGL syntax differs slightly. For example, in the operator CHAR(Datatable, Class, Params), Datatable does not need to be specified; KGL automatically uses the one currently in use. In the cases below, "Params" represents a parameter set associated with the operator. Many of these parameters are optional; if they are not specified, the system will use default values.

CHAR(Datatable, Class, Params): Characterize a set of entities of a given class in the given datatable (i.e., build a characteristic description), using the given parameters to guide the characterization process.

DIFF(Datatable, Class, Negative Examples, Params): Differentiate entities of the given class in the given datatable from the set of negative examples (typically but not necessarily all examples of other classes in the datatable), using the given parameters to guide the differentiation process.

SELECT(Entities, Datatable, Params): Select components, specified by *entities* (an expression defining the entities to be selected, such as examples (rows), attributes (columns), or both), of the given datatable to be selected according to the criteria specified by Params, and thus form a smaller datatable that is a subset of the original one. The parameters may be deterministic (e.g., "select the first 50 rows") or based on the result of some data analysis (e.g., "select the 5 columns corresponding to the attributes that will be most useful in differentiating between the values of a given dependent attribute).

TEST(Datatable, Ruleset, Params): Test the knowledge contained in the given ruleset against a set of testing examples provided in the given datatable, using the given testing parameters. Report on the ruleset's consistency and completeness.

GENSTAT(Datatable, Params): Generate a statistical report on a classified datatable that provides the user with means, modes and variances of attributes (both over the entire data set and for individual classes), and covariances and correlations between numeric attributes.

PREDICT(Ruleset, Example, Params): Predict the class of the given example based on the given rules and inference parameters.

Currently under development for KGL are the following operators for generating statistical reports, conceptual clusters, and decision structures:

STRUCT(Input, Params): Generate a decision structure for classification based on the input, which may be a classified datatable or a ruleset, and a set of parameters to guide the operator's search.

CLUSTER(Datatable, Params): Conceptually cluster the rows of the datatable into classes, guided by the parameter set. Return the classified datatable and characterizations of the created classes.

One of KGL's most powerful capabilities is its ability to access and query a knowledge base, and make decisions based on the knowledge. The functions #rules, #conditions, and #references, and the analogous constructs forall(rules, …) and forall(conditions, …) all take one or two arguments; the first one names the dependent attribute for which the rules were generated (it can also be the position number of the variable or a KGL variable which evaluates to the name of an attribute), and the second states the conditions under which the knowledge component is to be counted. If there is no second argument, the functions return the total number of rules, conditions that make up all the rules, or individual values referenced in the conditions respectively. If the second argument is provided (only applicable to #rules and #conditions), the functions return the number of rules/conditions that meet the requirements specified by the second argument.

The second argument to these functions may contain conjunctions, disjunctions, comparisons, etc., as are supported in the **if** and **while** commands. They may also contain special attributes and functions derived from the knowledge itself.

For the #conditions function and the forall(conditions) statement, the following constructs are available:

pos: total number of training examples of the target class that satisfy the condition

neg total number of training events for decision classes other than the target class that satisfy the condition

supp level of support the condition alone gives to the target hypothesis, defined as **(pos / pos + neg)**, and expressed as a percentage

comm commonality level of the condition, defined as the number of training examples of the target class that satisfy the condition divided by the total number of training examples of that class, and expressed as a percentage

For the #rules function and the forall(rules) statement, the following constructs are available:

num_covd: total number of training examples that satisfy the rule

%covd: percentage of training events for the class that satisfy the rule

covers(key): true if the rule is satisfied by the training example with the given key value

num_conds(…): total number of conditions that make up the rule that satisfy the characteristics specified in its argument. The characteristics can be any characteristics that may be specified in the **#conditions** call

For both the #rules and the #conditions functions, as well as the associated forall statements, the following constructs are available:

class the class of the action portion of the rule (or the rule that contains the condition) in question, returned in the form of a quoted string

rno the position of the rule in its class's ruleset. The first rule for the class has rno = 1, the second has rno = 2, etc.

has_attr(…) true if the rule or condition references at least one of the attributes given as arguments

has_val(…) true if the rule or condition references at least one of the values given as arguments. The syntax of the arguments is as follows:
The first argument is the name of the attribute whose values are to be checked. Subsequent arguments are the individual values. Ranges may be specified by including the end points in parentheses, separated by a space.

New capabilities are being added to KGL in a format similar to those described above, as experimentation demonstrates the need.

5 STATUS OF IMPLEMENTATION

This chapter describes the implementation of the INLEN system. As described above, the complete system integrates many modules that often serve as powerful stand-alone programs. In order to implement such a large-scale system with so many capabilities, this development has been undertaken in stages. Each stage represents a version, with an increasing set of capabilities, which can be independently tested and applied. Each stage also represents and advance in the integration of all of the implemented pieces.

5.1 Development history

The first stage of development (version INLEN-1) came from a C-language version of AURORA. It included a knowledge base of simple decision rules, a prototype relational database, and an extensive user-oriented menu-based graphical interface. The knowledge generation operators (KGOs) consisted of a subset of the full set of operators. Like AURORA, INLEN-1 was implemented on an IBM-compatible computer. It was limited to small data sets due to the DOS 640K barrier (approximately 13 attributes and 180 records in a data set).

The second major phase of INLEN development (INLEN-2) expanded the size of the datasets that could be learned from, and incorporated new features including advanced processing of structured domains, automatic quantization of numeric domains, structured decision variables, and two modes of attribute selection (see Chapter 4 for details of these techniques). The size limitations of INLEN-2 were as follows:

Maximum Number of Knowledge Systems allowed at one time:	30
Maximum Number of Application Domains:	20
Maximum Number of Variables per Database:	95
Maximum Number of Values per Variable:	200
Maximum Number of Examples to be learned from:	650
Maximum Depth for Simple Structured Variable:	5
Range of Simple Integer Type Variable:	0-50
Maximum Value of *Maxstar* Learning Parameter:	30
Maximum Number of Values Displayed by Advisory Module:	50
Cutoff Value for Rule Confirmation:	70

The latest version, INLEN-3, which continues to evolve, includes the creation of a distributed knowledge base and the development and incorporation of a language and interpreter for knowledge discovery tasks (see Sections 4.3.3 and 4.4 respectively).

5.2 INLEN-3 in detail

In this section, the implemented modules of INLEN-3 are described from a user's point of view (a technical description of the program is found in Appendix A.)

The main menu offers the user five options: creating or developing a knowledge system, entering the learning and discovery module, applying an advisory system to a problem, browsing a knowledge system, and entering a tutorial on using INLEN.

5.2.1 Developing or browsing a knowledge system

These two menu options are described together, because they follow closely the same course. While the Develop option allows modification of the components of a knowledge system (through create/access, copy (to another knowledge system), and delete options), the Browse option only allows the user to view or print them.

In either case, the user is presented with a three-column menu. In the first column are the names of the knowledge systems known to INLEN, headed by <New System>, for cases in which the user wishes to define a new one. The second column lists the available operations on the systems, as listed above. The third column lists the components of the knowledge system: System Description, Rules, Variables, Examples, Testing Events, Learning Parameters, Inference Parameters, and Whole System. By choosing one item from each column, the user may select options such as accessing the testing events of the PEOPLE knowledge system.

The **system description** consists of four components: application domain (which is used for sorting the knowledge system list in the Develop/Browse system menus), the name by which the knowledge system will be known, a short (up to 55 characters) description of the system, and the name of a file (if any) that contains a more detailed description of the knowledge system.

When **rules** are selected, the options available to the user are closely related to the Learning and Discovery Module. The program is routed into that module. The various functions were be discussed in depth in Section 5.2.2. It suffices to say here that the user is presented with options to learn rules from examples, manually enter or alter rules, optimize an existing ruleset according to criteria specified by the learning parameters, test rules against a testing event set, or combine rulesets.

When **variables** are selected, the user is presented with a spreadsheet-like table, in which the attributes for the knowledge system can be defined. The top row consists of the names of the attributes (with the dependent/decision attribute in the leftmost column). The next row contains either the attribute types (nominal, linear, structured, integer (0-50) or numeric (specified by the range of values, e.g., 2-16)) or the attribute costs (0-100). Subsequent rows list the attributes' legal values. If the attribute type is nominal, the order of the values is irrelevant. If the type is linear, the values are ordered from lowest to highest. If the attribute type is structured, the values are ordered in a depth-first traversal of the hierarchy; by typing a Tab after the value name, the user can add a level of depth (indicated by one level of indentation on the screen). A value is thus the direct descendant of the first value above it with one fewer level of indentation, and the direct parent of all values below it with exactly one more level of indentation that are above the next value with the same level of indentation (or less). If the type is integer, no values need be given. If the type is numeric, interval ranges may be specified in the form [low-high] (ordered as if the attribute was linear), or they may be generated automatically through dynamic discretization.

A user can provide additional structure information by dividing the structure definition into several adjoining columns and giving columns after the first one the attribute types *substructure* or *sview*. A substructure represents an addition to the structure, and as such its attribute name must be identical to a node defined in a previous column of the structure

specification. This node then becomes an *anchor node* of the structure (Section 4.3.2). An sview represents an overlapping portion of the structure; through this feature, multiple views of the structure may be specified, allowing a node to be the direct descendant of two or more nodes. INLEN's generalization operators will automatically choose the best way to generalize a node, given a context of other nodes.

The user may also access "annotations" within the variables table, which provide additional information on the attributes and their values. Each variable can be assigned an importance level from 1-10 that will be used by the Advisory Module (Section 5.2.3), and a question for the advisory system to ask when seeking the value of that attribute (otherwise the default "<attribute-name> is" is used. Values of decision variables can be assigned names of files to display and/or files to execute should they be chosen by the advisory module. Values of other attributes can be assigned names of help files to be displayed should the user need further information about them.

Within the Variables Table editor, the user has access to a number of functions such as duplicating, moving or swapping values or columns. The user may also call upon the SELVAR (select variables operator). The user will be prompted for the parameters by which the selection is to be made: whether to select attributes based on overall or maximum discrimination ability (see Section 4.3.4), whether to base the cutoff between selected and rejected attributes on number of attributes to be selected or on a threshold Promise level, and what the threshold value should be for selection/rejection of attributes.

Once the attribute selection module provides its advice, the user may accept, reject or modify the new attribute set either by adding the next best attribute to the set of retained attributes or by removing the worst retained attribute from the list of attributes to be retained.

The **examples** and **testing events** editors appear very similar to the variables editor, but their rows constitute individual training/testing examples, with blank cells representing unknown values. INLEN checks and requires that entered values be legal ones as defined in the Variables table (although new values can be added as an afterthought). The examples tables have an additional column, called "key", that may be displayed at the user's discretion. This column allows for the specification of a unique identifier for the example. If one is not provided, a sequence number will automatically be generated.

Examples may be entered in several ways in INLEN-3. In the examples editor, a user may either type in attribute values by hand or use the available key to scroll through legal attribute values. A user may also import the file from elsewhere, either from a DBase file (.dbf extension), an INLEN-formatted file (.dta extension, 12-character fixed length fields beginning with the key, one example per row), or a free-form file (any other extension, the first row lists the attribute names, separated by commas, subsequent rows list the examples, attributes in the specified order, values separated by commas).

When **learning parameters** is selected, the user may access some of the parameters used by the AQ learning program. The user may select a value for *maxstar* (the desired scope of the beam search), choose between intersecting and disjoint rulesets, and select a rule preference criterion, either one of the four predefined ones (characteristic, discriminant, minimal complexity, minimum cost) or a user-defined set of criteria.

Similarly, the **inference parameters** editor allows the user to fine-tune parameters for use by the advisory module. These parameters serve to specify the updating of hypotheses' degrees of confidence, and to guide the operation and termination of the advisory session. For details, see (Kaufman and Michalski, 1997b).

When **whole system** is selected in access mode, the user receives a listing of the component files of the system -- which have been built and which are not yet present.

5.2.2 The learning and discovery module

The learning and discovery module is the setting for many of INLEN's knowledge generation operators. The user is presented with one of three menus depending on the means of entry -- from the main menu, from the "create rules" option of the develop system menu, or from the "access rules" option of the develop system menu. Among the options available to the user are rule learning, optimization or improvement through the AQ15c program, rule testing using the ATEST program, rule compilation, generation of a statistical analysis of a data table, and execution of a KGL program. Currently under development to be added to this section of INLEN is an operator for generating task-orienteddecision structures from decision rules.

In **rule learning**, the operator generalizes a set of examples based on the provided *learning parameters*. In **rule optimization**, the same procedure is followed, but the input is a ruleset rather than a set of examples. By this means, a discriminant ruleset may be generated from characteristic descriptions, that will often be of higher quality than had the ruleset been learned directly from examples (Cuneo, 1975). In **rule improvement**, both an example set and a ruleset serve as inputs; the rules are refined so as to effect consistency with the new examples.

In **rule testing**, the testing events are matched against a ruleset to test the rules' consistency and completeness. For a given testing event, it is first determined whether the example completely satisfies any rule for any decision class. If one or more rules are satisfied, the classes are assigned degrees of match as follows:

- If no rules for the class are satisfied by the example, degree of match = 0.
- If one or more rules for the class are satisfied, the degree of match is the probabilistic sum of the degrees of match for each satisfied rule (probabilistic sum(A, B) = (A + B) / AB), where the degree of match for a satisfied rule is the ratio of the number of testing examples of the rule's class satisfied by the rule to the number of testing examples of the rule's class satisfied by any rule for that class.

If no rules are satisfied exactly by the testing example, the example's degree of match for the class is the probabilistic sum of its degree of match for all rules in the class, each of which is defined by the percentage of conditions in the rule that are satisfied by the example.

The testing examples are assigned to the class to which they have the greatest degree of match. If this class is the one intended for the testing example, it is counted as a correct classification; otherwise, it is counted as incorrect.

The testing operator returns a confusion matrix showing for each testing event whether it matched a rule exactly, its degree of match for each class, and the class to which it was

assigned. The output is summarized by counts of the numbers of correctly and incorrectly classified examples, and the accuracy percentage of the test.

If the decision attribute was of structured type, these numbers will be returned for each node in the structure at which examples were tested. The methodology is as follows: If testing an example of a class that is not at the top level of the structure, test the example in the context of all top level nodes. If the testing operator classifies the example as a non-ancestor of its intended node, score the test as incorrect at the top level and stop. Otherwise, score the test as correct at the top level, and test the example with regard to the nodes that are children of the one to which the top-level test assigned it. If the result of the test is incorrect at this level, score it accordingly; if it is the proper class, score it as correct at this level, and if it is an ancestor of the proper class, score it as correct at this level and repeat at the next level.

The **rule compilation** process involves all rulesets in the knowledge system, rather than simply the active decision class. The rules are checked for syntactic accuracy, and if all goes well, cross-index tables of the attributes, values, rules and conditions are put into a file for the use of the advisory module.

The **statistical report generator** creates up to four tables providing a statistical snapshot of the data. The first table contains means of numeric non-decision attributes and modes of non-numeric ones, both for individual decision classes and for the data set as a whole. The second table contains variances of numeric non-decision attributes, both for individual decision classes and for the data set as a whole. The third and fourth table respectively contain covariances and correlation coefficients between all pairs of numeric attributes. Currently, these tables are displayed to the screen and written to files in the knowledge base; the user will be able to links between the statistical and declarative knowledge using tools under development in the KGL language.

If the user chooses to execute a **KGL program**, the system will prompt the user for the name of the file containing the program. The program will then run without prompting the user, save for updates on the status of operator calls and any screen displays that the program may have been instructed to give. The output from the program is written to file KGL.OUT, in addition to any changes that may be made to the knowledge base.

5.2.3 The advisory module

A user can enter the Advisory Module in one of two ways – either by selecting it through the Main Menu or, if the sole purpose of the INLEN session is to get advice from a particular application system, by typing INLEN followed by the name of the application system at the DOS prompt. The latter method will bypass the screens involved in the selection of an advisory system.

When the module is entered from the Main Menu, the user will be able to select from a list of advisory systems. Once the advisory system has been selected, if rule bases exist for more than one decision attribute, the user will be prompted for the decision attribute to be determined during the advisory session.

An advisory session is a three stage process, consisting of *reduction*, *discrimination* and *confirmation*. During reduction, the user is asked for the values of variables whose importance values have been defined as 10. The purpose of this phase is to remove

immediately from consideration any hypotheses that contradict the user's input. (As an example, consider the process of choosing a restaurant at which to eat. If a group decides on Chinese food, they won't even consider a nearby Mexican restaurant, no matter how well it meets their needs in terms of price, convenience, etc.) If more than one hypothesis remains after reduction, INLEN continues to the discrimination phase, in which the effort is to single out one outstanding candidate hypothesis from the set of hypotheses still under consideration. The purpose of the final phase (confirmation) is to confirm or disconfirm that particular hypothesis; if it is disconfirmed, a new leading candidate hypothesis is selected, and confirmation continues. During all three phases, INLEN keeps the user notified of its progress and its confidences in the leading hypotheses.

The confirmed candidate hypothesis, or (if none was confirmed) the one with the highest confidence value when all questions were exhausted, will be the one INLEN suggests to the user. If a decision file was specified in the decision value's annotations, its contents will be displayed on the screen. Otherwise, a simple statement of the decision will appear. If it is desired, the user may then change one or more answers and see how that affects the ultimate decision.

The inference parameters and the process of calculating degrees of confidence are described in detail in (Kaufman and Michalski, 1997b).

6 EXPERIMENTS

This chapter presents some of the results of testing different features of this methodology using different sets of training data and different learning goals. Many of these experiments used economic and demographic data from two sources – a database containing facts and figures for different countries for each year from 1965 to 1990 gathered by the World Bank, and a database using the information contained in the 1993 World Factbook [CIA, 1993].

The World Bank database contains records on 95 statistics for each year in which the information was available. Examples of the available data for a given country and year are statistics on the population including size, growth rate, density, sex and urban/rural ratios; birth and death rates; land features (e.g. forests, agriculture); GNP and allocations thereof; educational statistics; etc.

One experiment using this database aimed to find differences between the development patterns of countries in Eastern Europe and the Far East [Kaufman, 1994]. Conceptual clustering was performed using program CLUSTER/2 [Stepp, 1983] in order to characterize the different countries into groups. One clustering consisted of four groups, two of which exclusively contained Eastern European countries and two of which contained only Asian countries. Two of the groups (one for each region) contained only two countries apiece. Furthermore, looking at the way each region's countries had been divided into groups, the partitions were based, among other thing, on the change in a country's labor force participation (CLFP) during the 1980s. In the case of Eastern Europe, the two countries put into a separate group (Albania and Romania) had a higher CLFP than the rest of the region's countries, but in the case of the Far Eastern countries, the two nations singled out (Laos and Cambodia) had a lower CLFP than the rest of the nations in the region. This clustering is graphically represented in Figure 4.

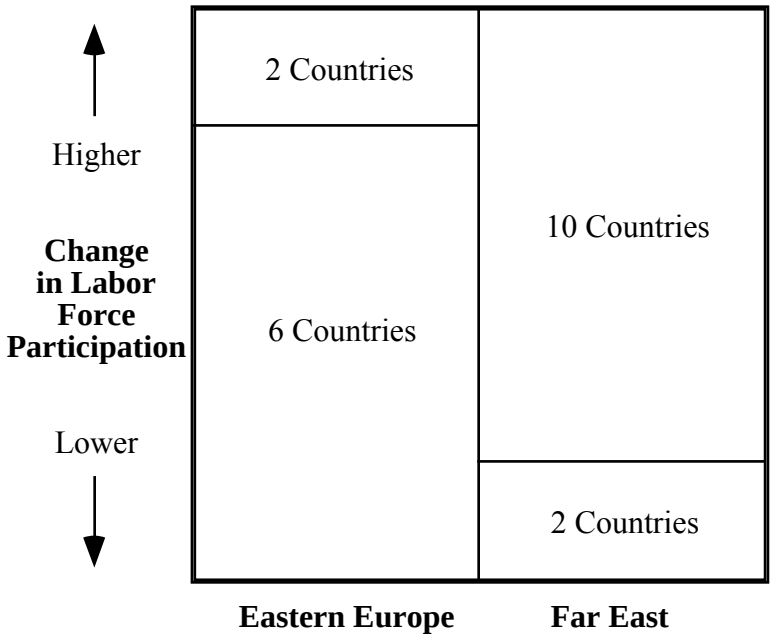

Figure 4. Conceptual Clustering of Eastern European and Far Eastern countries

This grouping suggested that Eastern European countries tended to have had low CLFPs for their regions, while Far Eastern countries tended to have high ones. As such, it could be surmised that countries with low CLFPs for their region were following an Eastern Europe-style pattern of economic development, while the Far Eastern-style pattern of development had high ones. Class characterization and knowledge transformation through rule optimization were then performed sequentially using the AQ15c program (Wnek et al, 1995) to distinguish between countries that fit into the Eastern European model and those that fit into the Far Eastern one. Factors found to be useful in distinguishing between the two patterns of development were life expectancy, percentage of the population who are of working age, and percentage of the labor force in industry, in addition to the CLFP.

This experiment demonstrated one of the cornerstone features of the methodology - the integration of different learning and discovery strategies in such a way that knowledge can be passed from one operator to another in a seamless thread of discovery, leading to conclusions unreachable by any one individual program.

A follow-up experiment sought out distinctions and exceptions among more of the world's regions, based on attributes that describe quality of life, labor structure, consumption, education, etc., by using characteristic rule learning and testing operators. Core countries deemed typical of each region studied were used to build the characterizations, and the resulting rules were then tested against other countries in the region using the ATEST operator [Reinke, 1984] in order to determine which countries followed their regional patterns and which did not.

Some patterns appeared very clearly in the characterizations of the regions. For example, the core countries of Mediterranean Europe tended to have lower percentages of their gross national products allocated to education than most areas. Southeastern Europe showed a rural society, heavily devoted to agriculture. The developed countries of the Far East had some of the world's lowest death rates.

When non-core countries were tested against the characterizations of their regions, some countries were found to behave as if they were from another part of the world. Not surprisingly, China bore a closer resemblance to the then Communist countries of Southeastern Europe than to its East Asian neighbors. And Italy shared more in common with Western Europe than with its fellow Mediterranean countries.

One interesting finding was that island countries were more likely to deviate from the norms for their region than mainland countries. For example, while the economic patterns for geographically and politically unique Switzerland were right in step with the pattern of the Western European core countries, Ireland showed a noticeably different pattern (a 40% degree of match, as opposed to Switzerland's 80%).

This experiment illustrates INLEN's ability to point out trends in subgroups of the data and detect exceptions, both expected and unexpected. In another experiment using the World Factbook PEOPLE database, not only was an exception detected, but an explanation was offered to the user. While the subgroups in a demographic domain may indicate that member countries or regions have something in common, one type of exception may occur when one of the members of these constructed subsets shows a marked dissimilarity to the rest of the

group. Understanding the nature of these exceptions may prove in turn to be a springboard for additional discovery.

INLEN discovered several rules characterizing the countries with low (less than 1% per year) population growth [Kaufman and Michalski, 1996a]. One of the rules had three conditions that together were sufficient to distinguish 19 low growth countries from all of the countries with higher population growth rates. The rule is shown here with three weights attached to each condition: *Pos* represents the number of positive examples (countries with population growth rates below 1%) satisfying the condition; *Neg* represents the number of negative examples (countries with population growth rates above 1%) satisfying the condition, and *Supp*, defined as Pos / (Pos + Neg) in percent, represents an approximate measure of the degree of confidence that the condition alone provides toward the conclusion that a country will have a population growth rate below 1%.

Conditions characterizing Population Growth Rates below 1%:

		Pos	**Neg**	**Supp**
1	Birth Rate = 10 to 20 or Birth Rate ≥ 50	46	20	69
2	Predominant Religion is Orthodox or Protestant or Hindu or Shinto	40	68	37
3	Net Migration Rate ≤ +20	32	104	23

The first and strongest condition states that the country must have a low (under 20 per 1000 population) or very high (over 50) birth rate. The presence of a very high birth rate is extremely counterintuitive; using the links in the knowledge base, one may examine the 19 countries involved. Such an inspection points out that 18 have birth rates below 20, while only one, Malawi, has the high birth rate. INLEN had thereby identified an exception to normal patterns. When further learning was focused on Malawi, the explanation was clear. A massive outward net migration rate had been discovered, by far the most extreme migration rate in the world.

The rule shown in the previous example contained an attribute for predominant religion. In the initial dataset, this attribute was unstructured. An area for exploration is how the structuring of data affects the discovered knowledge. In order to approach this question, INLEN was used on identical data sets with and without the Religion attribute being structured [Kaufman and Michalski, 1996b]. A portion of the assigned structure is shown in Figure 5.

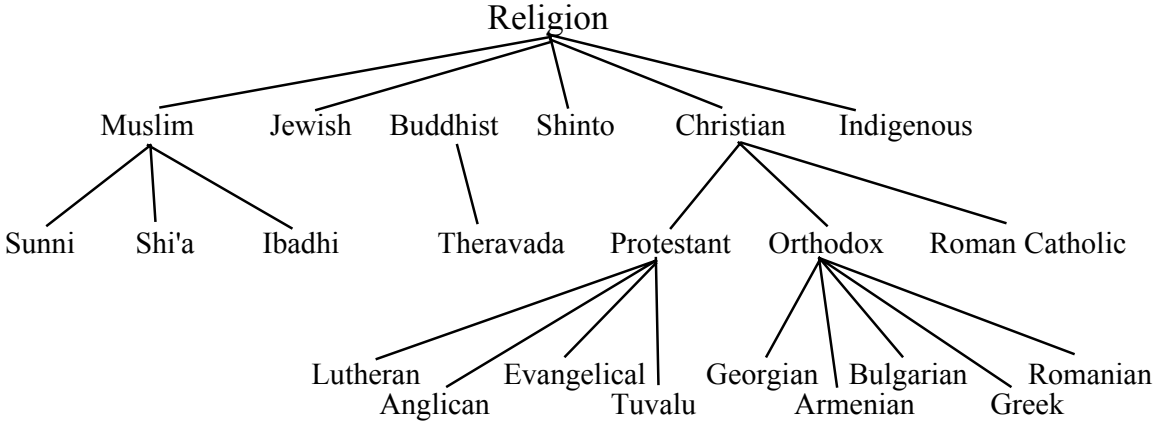

Figure 5. Part of the structure of the PEOPLE database's **Religion** attribute

One strong argument for structuring is as follows: If the Religion attribute were set up in an unstructured manner, the statement "Religion is Lutheran" would be regarded as being as antithetical to "Religion is Christian" as the statement "Religion is Buddhist," leading to the possibility that some contradictions (such as "Religion is Lutheran, but not Christian") might be discovered.

Experiments using INLEN have indicated very interesting findings regarding the usage of structured and non-structured attributes. Among the findings regarding their use as independent (non-decision) variables were that structuring attributes led to simpler rules than when the attributes were not structured, and that certain patterns were only found when the domains were structured. These findings are illustrated by the results below:

When INLEN-2 learned rules to distinguish the 55 countries with low (less than 1 per 1000 people) population growth rates from other countries, in a version of the PEOPLE database in which the attribute "Religion" was not structured, one of the rules it found was as follows:

Population Growth Rate < 1 if: (20 countries)
Literacy = 95% to 99%,
Life Expectancy is 70 to 80 years,
Religion is Roman Catholic or Orthodox or Romanian or Lutheran or Evangelical or Anglican or Shinto,
Net Migration Rate ≤ +20 per 1000 people.

As is indicated in its top line, this rule was satisfied by 20 of the 55 countries that had low growth rates.

When the same experiment was run with "Religion" structured, a similar rule was discovered:

Population Growth Rate < 1 if: (14 examples)
Literacy = 95% to 99%,
Life Expectancy is 70 to 80 years,
Religion is Roman Catholic or Orthodox or Shinto,
Net Migration Rate ≤ +10 per 1000 people.

This rule was satisfied by 14 of the 55 low-growth countries. The removal of some of the Protestant subclasses from the Religion condition and the tightening of the Net Migration Rate condition caused six countries which had been covered by the first rule, not to satisfy this rule. At a slight cost in consistency, simpler, more encompassing rules than either of the above were found:

Population Growth Rate < 1 if: (20 examples, 1 exception)
Literacy = 95% to 99%,
Life Expectancy is 70 to 80 years,
Religion is Christian,
Net Migration Rate ≤ +10 per 1000 people.

Even when not relaxing the consistency constraint, a concise rule not generated in the unstructured dataset was discovered that described 20 countries including the six described by the unstructured rule, but omitted in the structured rule:

Population Growth Rate < 1 if: (20 examples)
Birth Rate = 10 to 20 per 1000 people,
Death Rate is 10 to 15 per 1000 people.

Similar differences were obtained when structuring the output variable. By classifying training examples at different levels of generality, rules can be learned to classify both in general and in specific terms. This will tend to reduce the complexity and increase the meaningfulness of the rules at all levels of generalization.

In the PEOPLE database it is difficult to learn a set of rules for predicting a country's likely predominant religion given other demographic attributes without using the hierarchies. There are 30 religious categories listed for the 170 countries. The conditions making up the rules will likely have very low support levels (informational significance, defined as number of positive examples satisfying the condition divided by total number of examples satisfying the condition) because a single condition that exists in most of the countries with a certain predominant religion will typically be found, at least occasionally, elsewhere.

When learning rules for distinguishing between the religions in an unstructured format, the highest support level found in the entire rule base was 37% for the awkward condition "Literacy is 70 to 90% or 95 to 99%", which described 26 of the 50 Roman Catholic countries, and 43 of the remainder of the world's countries. In contrast, structuring the classification of religions led to several top-level conditions with higher support levels. For example, one condition, with a 63% support level in its ability to distinguish the 88 Christian countries was "Population Growth Rate ≤ 2".

More drastic effects were seen at the lower levels of the hierarchy. In the unstructured dataset, five rules, each with two to five conditions, were required to define the 11 Sunni Muslim countries. The only one to describe more than two of the 11 countries was this set of fragmented conditions:

Religion is Sunni_Muslim if: (4 examples)
Literacy = 100% or less than 30%,
Infant Mortality Rate is 25 to 40 or greater than 55 per 1000 people,
Fertility Rate is 1 to 2 or 4 to 5 or 6 to 7,
Population Growth Rate is 1 to 3 or greater than 4 per 1000 people.

The ranges in each of the four conditions are divided into multiple segments, suggesting that this is not at all a strong pattern. In contrast, the structured dataset produced two rules, each with one condition, to distinguish Sunni Muslim countries from other predominantly Islamic nations. The first, "Religion is Sunni_Muslim if Infant Mortality Rate ≥ 40", described all but one of the positive examples (Jordan was the exception), and only one predominantly Islamic, but non-Sunni, nation. The second, "Religion is Sunni_Muslim if Birth Rate is 30 to 40" alone discriminated Algeria, Egypt, Jordan and Tajikistan from the rest of the Muslim world.

Experiments also indicate the practical utility of problem-oriented quantization of continuous variables. Advantages include simpler rules that will often have higher accuracy than with a fixed discretization schema. When characterizing different regions of the world based on an economic database, INLEN-2 was able set thresholds that would generate knowledge with high support levels. For example, the allocation of a country's GNP to agriculture of greater than 28% (with that number generated by ChiMerge) was a useful indicator for distinguishing Eastern African countries from other regions of the world.

An experiment using the World Bank 1990 database showed the effect of intelligent attribute selection on the discovery process. Discriminant rules were generated to differentiate the various regions of the world in three ways: using the entire data set (34 attributes), using the ten attributes deemed to have the best average discrimination capability as measured by PROMISE, and using the ten attributes with the best maximum discrimination capability according to PROMISE. Learning on the full data set took over an hour, while runs could be made on the smaller data sets in just a few minutes.

Using the full data set, each region required, on the average, 1.5 rules to describe its member countries. These rules, on the average, had roughly 5 conditions apiece. From the smaller data sets, more, but simpler rules were generated. Using average discrimination as a criterion for attribute selection, 2.4 rules were needed per class, and 3.4 conditions per rule. With the maximum discrimination criterion, 2.4 rules were also needed, but these averaged 4 conditions apiece. One difference was found in the rulesets that supported the hypothesis that the maximum discrimination criterion is more useful for building a rule-based (declarative) knowledge base. When using this criterion, more conditions with high support levels were found, suggesting that the knowledge was more succinct and not simply fit to match the training example set.

One unexpected result from this experiment was an insight into the handling of sparse data. In this database, many of the countries had values for most of the data attributes; however, little information was available regarding a few countries, for example Bermuda. When selecting attributes based on the overall discrimination criterion, no attributes were retained that provided any information about Bermuda, and therefore no rules were generated to encompass it. With the maximum discrimination criterion, one of the attributes for which information about Bermuda was available, amount of cereal imported, was retained. Hence a rule could be learned with a condition based on this attribute that Bermuda could be said to satisfy (at least to some degree, since for all of the other conditions, Bermuda's status was unknown).

There is no intrinsic reason why one of the attribute selection criteria retained an attribute whose value was known for Bermuda and the other did not; it was a random occurrence based on the data. Nonetheless this puts forth the issue, should feature selection methodologies be biased in the case of sparse data so that all records in the dataset are represented by at least one known value in the data.

7 EXAMPLES DEMONSTRATING THE USE OF KGL

We examine a problem is concerned with searching a database of countries for patterns, regularities and exceptions. The dataset was extracted from the World Factbook's PEOPLE data table. This data table consists of characteristics of 190 countries in terms of nine attribute-value pairs per country. The attributes represent the country's population growth rate (PGR), birth rate (BR), death rate (DR), infant mortality rate (IMR), net migration rate (NMR), fertility rate (Fert), life expectancy (LE), literacy percentage (Lit), and predominant religion (Rel). Each attribute has between 5 and 7 values, with the exception of predominant religion, which has 31.

The following samples of KGL code illustrate some of the language's capabilities. The code describes a plan for determining various relationships between the different attributes. Specifically, it searches for strong relationships in the characterizations of the different concepts, tries to improve knowledge that does not meet desired quality standards, analyzes the makeup of a ruleset, and sees if the perceived similarity between two countries is a tight or a superficial one.

We begin by learning a set of characteristic descriptions of all the attribute classes in the data table. The parameters for this operator are extracted from the file people1.lrn, except for the scope parameter, which is automatically set to 2.

```
open PEOPLE
do CHAR(decision=all, pfile=people1.lrn, scope=2)
```

From these characterizations, the program counts the number of "strong" descriptions of Population Growth Rate classes in three fashions: those that describe at least 60% of the training examples of their class, those that describe at least 25 training examples, and those that have at least two conditions with support levels (number of positive examples covered / total number of examples covered) greater than 50%. The three counts are output and the average of these three numbers and the number of "strongest rules"—those that qualify in all three strong categories are displayed on the screen.

```
strongrules1 = #rules(PGR, %covd >= 60)
strongrules2 = #rules(PGR, num_covd >= 25)
strongrules3 = #rules(PGR, num_conds(supp > 50) >= 2)
strongestrules = #rules(PGR, %covd >= 60 and num_covd >= 25 and
      num_conds(supp > 50) >= 2)
print "strong rules of all types: ", strongrules1,
      strongrules2, strongrules3
display "Avg: ", avg(strongrules1, strongrules2, strongrules3),
      "strongest: ", strongestrules
```

The program found one strong rule of each of the first two types, and seven of the third type. No rules qualified as "strongest rules."

The next section of KGL code examines the Infant Mortality Rate ruleset. If too many spurious (covering less than 10% of the class) rules are found, relearn the ruleset in discriminant mode with a higher scope of search and compare the total number of references in the ruleset.

```
spuriousIMrules = #rules(IMR, %covd < 10)
if spuriousIMrules > 6
      begin
```

```
        display “Relearning IMR in Discriminant mode)
        print “Old number of IMR references: “, #references(IMR)
        do DISC(decision=IMR, pfile=people1.lrn, scope=5)
        print “New number of IMR references: “, #references(IMR)
        end
else
        display “IMR rules were judged as adequate”
```

When applied to the PEOPLE data table, the program discovered that there were too many spurious Infant Mortality Rate rules. When discriminant descriptions were learned, the number of references in the ruleset was reduced from 591 to 326.

The next line provides an example of different counts in KGL as it analyzes the IMR ruleset. It generates counts of the number of rules that reference Literacy, the number of rules that reference either Literacy or Religion, and the number of conditions that reference Literacy or Religion. The last two numbers may not be the same, since a rule may have a condition that references Literacy and one that references Religion.

```
print #rules(IMR, has_attr(Lit)), #rules(IMR, has_attr(Lit,
      Rel)), #conditions(IMR, has_attr(Lit, Rel))
```

KGL found 28 rules that referenced Literacy, 38 rules that referenced Literacy or Religion, and 58 conditions that referenced Literacy or Religion.

We continue analyzing the ruleset as we discover the number of Infant Mortality rules with different numbers of conditions and the number of Fertility conditions that achieve various positive-negative coverage ratios.

```
for i = 1 to 7
      begin
      print “IMR rules with “, i, “ conditions: “, #rules(IMR,
            num_conds() = i)
      print “Fert conditions with P/N ratio of “, i, “:1: “,
            #conditions(Fert, supp = 100 or pos / neg >= i)
      end
```

The program found 0, 1, 4, 17, 16, 3, 0 Infant Mortality rules of sizes 1 to 7 respectively; most discriminant descriptions in this domain were 3 or 4 conditions long. The number of Fertility Rate conditions with positive/negative ratios exceeding 1:1 to 7:1 were 43, 15, 7, 2, 2, 2, and 2 respectively. Hence, many conditions had support levels of at least 50%, then a sizable drop-off at 66%, and it would appear that two conditions covered no negative examples at all.

One of the findings of the experiment described in the previous chapter in which rules were generated for “core” countries of a region and then other countries were examined as to how well they fit their region’s pattern, was that Canada tended to follow some of the patterns of the Far East more closely than it did those of the United States. For instance, Canada and Japan were in the same class for six of the nine decision attributes in the PEOPLE database. One indicator of whether such a relationship might be significant or superficial is whether the two countries are described by the same characterizations, or whether different characterizations within the same class are required to describe them. The following KGL code conducts this experiment by checking the ruleset for each decision attribute:

```
num_match = 0
for i = 0 to 8
      begin
```

```
        forall rules(i, covers("Canada"))
              if covers("Japan")
                    print var_name(i), " groups them together"
              else
                    print var_name(i), " groups them separately"
        num_match = num_match + #rules(i, covers("Canada") and
           covers("Japan")
        end
print "Rules covering both: ", num_match
```

This experiment discovered that the only occasion in which Canada and Japan were grouped together in a rule was in the ruleset describing Birth Rate, thus supporting the hypothesis that the perceived similarities between the two countries may be superficial.

Finally, two more operators are called, both of which automatically output a summary of their results. In the first, the knowledge base is used to predict a country's likely Net Migration Rate based on some other known data -- Population Growth Rate, Death Rate, Life Expectancy and Fertility Rate values. In the second, the four best attributes (based on a decision rule representation) for determining a country's likely predominant religion are selected, and the data set is projected on these attributes into the data table PEOPLE2.

```
do PREDVAL(decision=NMR, PGR=2..3, DR=20..25, LE=70..80,
      Fert=3..4)
do SELVAR(decision=REL, out=PEOPLE2, thresh=4, criterion=max)
```

The prediction operator returns an estimate of an outward Net Migration Rate of less than 10 per 1000 people with 44% confidence. Other hypotheses it found plausible were an inward migration rate of less than 10 per 1000 people (27% confidence) and an inward migration rate of between 10 and 20 per 1000 people (24% confidence).

The feature selection operator determined that the four attributes best suited to the creation of decision rules for Predominant Religion were Death Rate, Life Expectancy, Birth Rate and Fertility Rate.

The above samples of KGL code are intended to give the reader a sample of the kind of tasks INLEN can be instructed to perform through the use of a high-level language. While all of the capabilities demonstrated above have been implemented, the language is still under development, and new features and abilities continue to be added.

8 CONCLUSIONS

8.1 Summary

This thesis presents a methodology for knowledge discovery based on the integration of symbolic and statistical learning and discovery techniques. Because of the increasing reliance on large amounts of data, intelligent data analysis has become increasingly vital. The tools offered by machine learning provide steps toward an answer.

Given the state of today's technology, integration of diverse learning and discovery strategies is the most likely architecture for a discovery system that will be able to give answers to the different kinds of questions a data analyst may ask.

The development of INLEN has encountered the problems associated with integrating different discovery tools as operators into a single system. The knowledge segment architecture and the KGL language have evolved to facilitate the free exchange of information between operators, and between machine and human.

Other major research issues have arisen in response to the application of symbolic learning to knowledge discovery in a large database environment. These include the exploitation of the rich background knowledge available in many databases, coping with sparse data, selecting subsets of the data that will make for a tractable discovery process at a small loss in predictive accuracy, and the development of a control system to allow for partial automation of the discovery process.

During the course of this research, development has progressed on an implementation of the described methodology. A prototype has been built that incorporates previously existing, adapted, and new tools and techniques as its knowledge generation operators. Among the adaptations and additions implemented in response to the problems mentioned above were modifications to the AQ15 learning engine to effect consistent processing of structured input and output data in an environment of increased domain knowledge, and a new version of PROMISE, with an additional mode that biases attribute selection toward use in a rule-based environment.

The INLEN prototypes have been applied to data in engineering, document analysis and economic/demographic domains. Through these applications the capabilities of the methodology have been explored. It supports incremental discovery of patterns, trends, exceptions, anomalies and groupings through the application of different operators in sequence. It can provide explanations for its discoveries. It can generate subgroups from a set of events and find patterns in them. It can quantize continuous data and exploit data structuring in order to generate simpler, more useful discoveries.

8.2 Contributions

This work has explored some of the theoretical and practical issues in developing an integrated, symbolic-oriented approach to knowledge discovery in databases including:

- Development, implementation, and demonstration of a methodology for integrated knowledge discovery based primarily on the application of

symbolic learning operators. Inherent in its architecture is a central knowledge base sufficiently rich in semantic content for the operators to work independently, yet share information as the discoveries of one are utilized by another. The knowledge base can also maintain information on the discovery process itself, such as the training examples on which the knowledge is based, degrees of confidence, etc.

- Issues in implementing and exploiting structured data representations, resulting in the implementation of anchor nodes and generalization within a multiple hierarchy environment. Anchor nodes allow the user to define nodes in a generalization hierarchy as being at a greater or lesser level of utility (for the purposes of the problem at hand) than their ancestors and descendants. A learning program can be instructed to focus generalization and specialization on the nodes that are most desirable. The generalization algorithms necessary to effect this feature have been implemented and tested in INLEN.
- A multiple hierarchy representation allows a user to define an independent attribute based on more than one generalization hierarchy. A program can characterize a set of cases based on the attribute, and only then determine how best to represent the attribute. Given the attribute "aircraft", 747, 727 and 757 would generalize to "Boeing", while 747, L-1011 and DC-10 would generalize to "wide-body".
- A methodology for dynamic automatic discretization of continuous attributes. Currently, the discretization is performed using a Chi-square analysis of an individual variable's relationship to the dependent variable. Continuing research aims to improve upon this method through looking at combinations of attributes (perhaps integrating this method with constructive induction) and to develop a method for discretizing a continuous dependent variable.
- A means for selection of the attributes best suited toward a discovery problem based on the use of a rule-based knowledge base. Given a set of stand-alone decision rules, attributes that discriminate well for only one reasonably common value will lead to a succinct rule base.
- A framework, requirements specification, initial implementation, and demonstration of a high-level knowledge discovery language in which data and knowledge characterizations, sequences of operators and conditional instructions can be specified.

In summary, this work has explored problems of data exploration, and in response provided methodologies for operator integration, knowledge sharing, the better use of background knowledge, problem-dependent representation space optimization, and semi-autonomous discovery through the user's specification of a procedure to be followed.

A major advantage of the presented INLEN methodology is that it is not constrained by the task-specificity of single-strategy discovery systems. INLEN's architecture is based on modularity; new or upgraded operators may be incorporated nearly intact, with the only major changes coming in the input/output and the user interface.

8.3 Current limitations

8.3.1 Methodological limitations

One of the largest methodological limitations of INLEN is due to the fact that an architecture that integrates many different tools that have been used in stand-alone applications is by nature very massive. The infrastructure needed to integrate all these operators becomes increasingly complex as the power of the underlying system grows. Such components of the architecture as the ability to reason with structured data, a knowledge base consisting of distributed knowledge segments and the ability to access operators either interactively or through KGL programs have added greatly to the complexity of INLEN, and adding other novel components to the architecture will involve dealing with some or all of these complex features.

The design of the knowledge base does not yet allow for the storage of "commonsense knowledge". In the experiment (Chapter 6) in which the anomalous behavior of Malawi was detected, the key to the discovery was the expectation of a certain relationship between birth rate and population growth rate that was violated in Malawi in 1993. A store of domain-specific background knowledge of the type "Higher birth rate is usually linked to higher population growth" would allow a discovery system to flag data that behave anomalously.

The method for the dynamic discretization of continuous attributes is limited by the discretization algorithm used. The implemented ChiMerge algorithm only exposes relationships between the values of a single attribute and the decision class; relationships based on multiple attributes are not detected (although constructive induction could conceivably be used to generate new attributes from those interdependent ones). Furthermore, a satisfactory way of discretizing a continuous decision variable into useful decision classes has yet to be developed.

The presented means for reasoning with structured attributes is only applicable to structuring unordered values. Creating hierarchies of ordered attributes could be especially useful when dealing with integer data, for example.

The KGL language is brand new, and still evolving, primarily in response to detected shortcomings. As of this writing, there is no means by which individual records in the database can be accessed in reference to elements of the knowledge base (e.g., a query for records that satisfy all or most of the conditions of a certain rule). Currently, the syntactic and semantic requirements of the language are quite rigorous. For example, to reference a condition in the context of a rule, the rule must be referenced first ("For each rule that satisfies certain criteria, check certain conditions in the rule" would be a legal type of request, while "For each condition that satisfies certain criteria, check the rule in which it was found" would not). In experiments up to this point, KGL has been powerful enough to reflect the concepts we wanted to present to it, but there is no full assurance that this will hold when more complex queries to the knowledge base are made.

8.3.2 Implementational limitations

In some senses, the biggest implementational limitation of INLEN is its scope. By nature, integrating a large number of operators is a demanding task, even when most of the technology has been built in stand-alone modules. What has been implemented is still

essentially a prototype, with only a portion of the projected operators incorporated into the system, and the system yet to be made portable to a non-DOS platform. Nonetheless, what is present may serve as a proof-of-concept, showing some of the capabilities of this methodology.

As of this writing, the knowledge base supports only rule-based knowledge and links to supporting examples. Procedural and equational knowledge forms have not been implemented in INLEN.

Among the knowledge generation operators that have yet to be integrated into the system are operators for analysis of time-dependent data, generation of decision structures, equations and hierarchies, example selection, creation of new attributes, and data and knowledge visualization. While an operator was recently implemented for generating statistical reports, a method for linking such knowledge to symbolic knowledge has yet to be implemented.

While the creation of a distributed knowledge base alleviated some of the difficulties of older versions of INLEN (e.g., only one ruleset could exist at a time for any knowledge system, meaning that if you changed decision variables you would have to relearn or reload rules), some limitations still exist. Each ruleset (for a decision attribute) remains in a single file, so that if rules are relearned for one new class, the entire file will be overwritten and lost. Breaking the knowledge base files into individual covers and even rules seems to be an eventual necessity.

One limitation of the system's current implementation is its often high memory usage. Such operators as AQ15 by nature require a large amount of overhead for sizable problems. The variables and examples editors load in the entire data tables for possible manipulation. Hence, the size of the datasets available to INLEN will be limited by machine capacity until another approach is implemented, such as handling these functions through automatic DBMS queries.

Another limitation lies in the user interface itself. Versions up to INLEN-3 have relied on a keyboard-only interface with DOS graphics. Work has begun on making INLEN more portable and modern through a reimplementation in Java.

8.4 Future work

8.4.1 Implementational plans

Future development of the INLEN system includes a version portable to a Sun workstation with a portable graphical interface. Knowledge generation operators under development include statistical data analysis, decision structure generation, and selection of representative examples. The distribution of the knowledge base implemented in INLEN-3 is a further step toward INLEN's integration with a commercial-grade DBMS. The KGL language will be expanded, possibly with an SQL data interface.

8.4.2 Future research

While this research has resulted in approaches to some of the methodological issues involved in knowledge discovery, others remain to be addressed.

As mentioned above, INLEN's knowledge segments have, to date, only represented rule-based and example-based knowledge. As operators to generate and work with other forms of knowledge (e.g., equations or decision structures) are incorporated, experimentation will determine what representations offer the integration of and communication between these different knowledge types. One question to be addressed will involve the search for a basic representation that encompasses the components of each of these forms and may even allow for conversion between forms when appropriate.

INLEN's approach to structured data representation provides a powerful tool for the generation of understandable discoveries. While the implementation of anchor nodes contributes substantially to this ability, nodes within structures can currently be designated only as anchor or non-anchor. One area for future development is to develop a schema for assigning different priorities and/or thresholds to nodes in a structure, so that the level of generalization can also depend in part on the distribution of the descendants of a node in the training example set.

A related topic involves the setting of consistency thresholds for generalization. At times, all that prevents the closing of an interval or the climbing of a hierarchy is a small number of negative examples. It must be determined when it is preferable to generalize in spite of the negative evidence. This question ties in with the idea of the two-tiered concept representation of base concept and refinements (Bergadano et al., 1992).

Another issue for future research approaches the problem of discretization of continuous variables. The ChiMerge method employed by INLEN performs intelligent discretization, but it has several limitations. It can only examine one attribute at a time, and is thereby unable to view relationships between combinations of attributes and decisions when the relationship is not evident by viewing any one of those attributes. One possible approach is to integrate constructive induction with the quantization process. Another discretization problem that needs to be addressed occurs because Chi-Merge operates by correlating attributes with decisions, and as such is unable to quantize a numerical decision variable. Methods for grouping decisions, perhaps using conceptual clustering, need to be explored.

One possible approach to the discretization problem would utilize a hill-climbing algorithm implemented by Zhang (1997), in which each continuous attribute domain is divided into a number of equal-length intervals (the number being determined by iterations of training and testing and selecting the one with the best performance). Such an approach may be time-consuming, as every step of the search requires evaluation of the proposed discretization schema, and it may be limited through its equal interval requirement. However, such an approach could prove to be effective for both dependent and independent attributes.

This thesis presented a novel method of predicting the utility of an attribute for a given learning problem based on a rule-based representation. Nonetheless, exploration must be made of boundary and threshold conditions, for example, which attribute is more useful – one with a value with higher predictive accuracy, or one with a value with slightly less predictive accuracy, but which appeared more often in the training data set, implying that a rule involving this value might be used more frequently?

This continuing research aims at the resolution of a range of theoretical and implementational issues involved in the development of a methodology and a multistrategy system for the

intelligent analysis of large amounts of data. The theoretical issues are focused on the representation and integration of data, knowledge, and knowledge discovery tools for this task, allowing for the development of a domain-independent learning and discovery system that is not limited to a narrow scope of tasks, but is capable of assisting database analysts in diverse fields. This work thus aims to contribute both to extending the frontiers of automated knowledge discovery and to advance our knowledge in various application domains in which knowledge is actively being sought from available data. The advances in the methodological domains discussed here will effect enhanced knowledge discovery capabilities.

REFERENCES

Alexander, W.P., Bonissone, P.P. and Rau, L.F., "Preliminary Investigations into Knowledge Discovery for Quick Market Intelligence," *Proceedings of AAAI-93 Workshop on Knowledge Discovery in Databases*, Washington, DC, July 1993, pp. 52-60.

Baskin, A.B. and Michalski, R.S., "An Integrated Approach to the Construction of Knowledge-Based Systems: Experiences with ADVISE and Related Programs," in *Topics in Expert System Design*, Guida, G. and Tasso, C. (eds.), Elsevier Science Publishers B. V., Amsterdam, 1989, pp. 111-143.

Baim, P.W., "The PROMISE Method for Selecting Most Relevant Attributes for Inductive Learning Systems," Report No. UIUCDCS-F-82-898, Department of Computer Science, University of Illinois, Urbana IL, Sept. 1982.

Bergadano, F., Matwin, S., Michalski, R.S. and Zhang, J., "Learning Two-Tiered Descriptions of Flexible Concepts: The POSEIDON System," *Machine Learning*, 8, 1992, pp. 5-43.

Bloedorn, E. and Michalski, R.S., "Data Driven Constructive Induction in AQ17-PRE: A Method and Experiments," *Proceedings of the Third International Conference on Tools for AI*, San Jose, November 1991.

Bloedorn, E., Wnek, J., and Michalski, R.S., "Multistrategy Constructive Induction: AQ17-MCI," *Proceedings of the Second International Workshop on Multistrategy Learning*, Harpers Ferry, WV, May 1993, pp. 188-203.

Brachman, R.J., Selfridge, P.G., Terveen, L.G., Altman, B., Borgida, A., Halper, F., Kirk, T., Lazar, A., McGuinness, D.L. and Resnick, L.A., "Integrated Support for Data Archaeology," *Proceedings of the AAAI-93 Workshop on Knowledge Discovery in Databases*, Washington, DC, July 1993, pp. 197-211.

Central Intelligence Agency, *1993 World Factbook*, 1993.

Cestnik, B. Kokonenko, I. and Bratko, I., "ASSISTANT 86: A Knowledge Elicitation Tool for Sophisticated Users," in *Proceedings of the Second European Working Session on Learning*, Bratko, I. and Lavrac, N. (Eds.), Bled, Yugoslavia, 1987.

Collins, A. and Michalski, R.S., The Logic of Plausible Reasoning: A Core Theory," *Cognitive Science*, Vol. 13, No. 1, 1989, pp. 1-49.

Cramm, S.A., ESEL/2: "A Program for Selecting the Most Representative Training Events for Inductive Learning," Report No. UIUCDCS-F-83-901, Department of Computer Science, University of Illinois, Urbana, IL, Jan. 1983.

Cuneo, R.P., "Selected Problems of Minimization of Variable-Valued Logic Formulas," Masters Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1975.

Dietterich, T. and Michalski, R.S., "Learning to Predict Sequences," in *Machine Learning: An Artificial Intelligence Approach Vol. II*, Michalski, R.S. Carbonell, J.G. and Mitchell, T. (Eds.), Morgan Kaufmann Publishers, Los Altos, CA, 1986, pp. 63-106.

Dontas, K., "Applause: An Implementation of the Collins-Michalski Theory of Plausible Reasoning," Master's Thesis, University of Tennessee, Knoxville, TN, August 1988.

Falkenhainer, B. and Michalski, R.S., "Integrating Quantitative and Qualitative Discovery in the ABACUS System," in *Machine Learning: An Artificial Intelligence Approach, Volume III*, Kodratoff, Y. and Michalski, R.S. (Eds.), Morgan Kaufmann, San Mateo, CA, 1990.

Fisher, D., Knowledge Acquisition via Incremental Conceptual Clustering, *Machine Learning, 2*, 1987, pp. 139-172.

Gould, J. and Levinson, R., "Method Integration for Experience-Based Learning," *Proceedings of the First International Workshop on Multistrategy Learning*, Harpers Ferry, WV, November 1991, pp. 378-393.

Greene, G., "Quantitative Discovery: Using Dependencies to Discover Non-Linear Terms," Master's Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1988.

Han, J., Fu, Y., Wang, W., Chiang, J., Gong, W., Koperski, K., Li, D., Lu, Y., Rajan, A., Stefanovic, N., Xia, B. and Zaiane, O.R., "DBMiner: A System for Mining Knowledge in Large Relational Databases," *2nd International Conference on Knowledge Discovery and Data Mining*, Portland, OR, 1996, pp. 250-255.

Hong, J., Mozetic, I. and Michalski, R.S., "AQ15: Incremental Learning of Attribute-Based Descriptions from Examples, the Method and User's Guide," Report No. UIUCDCS-F-86-949, Department of Computer Science, University of Illinois, Urbana IL, 1986.

Imam, I. and Michalski, R.S., "Learning Decision Trees From Decision Rules: A Method and Initial Results From a Comparative Study," *Journal of Intelligent Information Systems*, Vol. 2, No. 3, pp. 279-304, 1993.

Imam, I., "Deriving Task-Oriented Decision Structures from Decision Rules," Ph.D. Thesis, George Mason University, 1995.

Imielinski, T. and Mannila, H., "A Database Perspective on Knowledge Discovery," *Communications of the ACM*, 39:11, pp. 58-64, 1996.

Imielinski, T., Virmani, A. and Abdulghani, A., "DataMine: Application Programming Interface and Query Language for Database Mining," *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, OR, pp. 256-261, 1996.

International Intelligent Systems, Inc., "User's Guide to AURORA 2.0: A Discovery System," Fairfax, VA, International Intelligent Systems, Inc., 1988.

Jonassen, D.H., Beisner, K. and Yacci, M., *Structural Knowledge: Techniques for Representing, Conveying, and Acquiring Structural Knowledge*, Hillsdale, NJ, Lawrence Erlbaum Associates, Inc., 1993.

Karni, R. and Loksh, S, "Structured Attribute Discovery as a Tool within Total Quality Management," 1996.

Katz, B., Fermanian, T.W. and Michalski, R.S., "AgAssistant: An Experimental Expert System Builder for Agricultural Applications," Report No. UIUCDCS-F-87-978, Department of Computer Science, University of Illinois, Urbana, IL, Oct. 1987.

Kaufman, K., "Comparing International Development Patterns Using Multi-Operator Learning and Discovery Tools," *Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases*, Seattle WA, 1994, pp. 431-440.

Kaufman, K, "Addressing Knowledge Discovery Problems in a Multistrategy Framework," *Proceedings of the Third International Workshop on Multistrategy Learning*, Harpers Ferry WV, 1996, pp. 305-312.

Kaufman, K. and Michalski, R.S., "EMERALD 2: An Integrated System of Machine Learning and Discovery Programs to support Education and Experimental Research," *Reports of Machine Learning and Inference Laboratory*, MLI 93-10, Center for Artificial Intelligence, George Mason University, Fairfax, VA, 1993.

Kaufman, K. and Michalski, R.S., "A Multistrategy Conceptual Analysis of Economic Data," Ein-Dor, P. (ed.), *Artificial Intelligence in Economics and Management: An Edited Proceedings on the Fourth International Workshop*, Boston, Kluwer Academic Publishers, 1996a, pp. 193-203.

Kaufman, K. and Michalski, R.S., "A Method for Reasoning with Structured and Continuous Attributes in the INLEN-2 Knowledge Discovery System," *2nd International Conference on Knowledge Discovery and Data Mining*, Portland, OR, 1996b, pp. 232-237.

Kaufman, K. and Michalski, R.S., "KGL: A Language for Learning," *Reports of the Machine Learning and Inference Laboratory*, MLI 97-2, George Mason University, Fairfax, VA, 1997a.

Kaufman, K., "INLEN-3: A Multistrategy System for Learning, Inference and Knowledge Discovery in Databases: Overview, Implementation, Experiments and User's Guide," *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA, 1997b (to appear).

Kaufman, K., Michalski, R.S. and Kerschberg, L., "Mining For Knowledge in Data: Goals and General Description of the INLEN System," in Piatetsky-Shapiro, G. and Frawley, W.J. (Eds.), *Knowledge Discovery in Databases*, AAAI Press, Menlo Park, CA, 1991a, pp. 449-462.

Kaufman, K., Michalski, R.S. and Kerschberg, L., "An Architecture for Integrating Machine Learning and Discovery Programs into a Data Analysis System," *Proceedings of the AAAI-91 Workshop on Knowledge Discovery in Databases*, Anaheim CA ,1991b.

Kaufman, K., Michalski, R.S. and Kerschberg, L., "Knowledge Extraction from Databases: Design Principles of the INLEN System," *Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems, ISMIS-91*, Charlotte NC, 1991c, pp. 152-161.

Kaufman, K., Michalski, R.S., and Schultz, A., "EMERALD 2: An Integrated System of Machine Learning and Discovery Programs for Education and Research, User's Guide," *Reports of Machine Learning and Inference Laboratory*, MLI 93-8, George Mason University, Fairfax, VA, 1993.

Kerber, R., "ChiMerge: Discretization of Numeric Attributes," *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, San Jose, CA, 1992, pp. 123-127.

Kerschberg, L. (ed.), *Expert Database Systems: Proceedings from the First International Workshop*, Benjamin/Cummings Publishing Company, Menlo Park, CA, 1986.

Kerschberg, L. (ed.), *Expert Database Systems: Proceedings from the First International Conference*, Benjamin/Cummings Publishing Company, Menlo Park, CA, 1987.

Kerschberg, L. (ed.), Expert Database Systems: *Proceedings from the Second International Conference*, Benjamin/Cummings Publishing Company, Menlo Park, CA, 1988.

Klimesch, W, *Struktur und Aktivierung des Gedaechtnisses. Das Vernetzungsmodell: Grundlagen und Elemente einer uebergreifenden Theorie*. Verlag Hans Huber, Bern, 1988.

Kubat, M., Bratko, I. and Michalski, R.S., "A Review of Machine Learning Techniques," Chapter in *Machine Learning and Data Mining: Methods and Applications*, London, John Wiley & Sons, 1997 (to appear).

Michalski, R.S., "Discovering Classification Rules Using Variable-Valued Logic System VL1," *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford, CA, 1973, pp. 162-172.

Michalski, R.S., "A Theory and Methodology of Inductive Learning," in *Machine Learning: An Artificial Intelligence Approach*, Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (Eds.), Morgan Kaufmann, San Mateo, CA, 1983.

Michalski, R.S., "Inferential Theory of Learning as a Basis for Multistrategy Task-adaptive Learning," *Reports of the Machine Learning and Inference Laboratory*, MLI 90-1, Center for Artificial Intelligence, George Mason University, Fairfax, VA, 1990.

Michalski, R.S., "Searching for Knowledge in a World Flooded with Facts," *Proceedings of the Fifth International Symposium on Applied Stochastic Models and Data Analysis*, Granada, Spain, April, 1991.

Michalski, R.S., "Inferential Learning Theory as a Conceptual Basis for Multistrategy Learning," *Proceedings of the First International Workshop on Multistrategy Learning*, Harpers Ferry, WV, November 1991, pp. 3-18.

Michalski, R.S. and Baskin, A.B., "Integrating Multiple Knowledge Representations and Learning Capabilities in an Expert System: The ADVISE System," *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, 1983, pp. 256-258.

Michalski, R.S., Baskin, A.B. and Spackman, K.A., "A Logic-based Approach to Conceptual Database Analysis," Sixth Annual Symposium on Computer Applications in Medical Care (SCAMC-6), George Washington University Medical Center, Washington, DC, 1982, pp. 792-796.

Michalski, R.S., Baskin, A.B., Uhrik, C. and Channic, T., "The ADVISE.1 Meta-Expert System: The General Design and a Technical Description", Report No. UIUCDCS-F-87-962, Department of Computer Science, University of Illinois, Urbana, IL, 1987.

Michalski, R.S. and Imam, I.F., "On Learning Decision Structures," *Fundamenta Matematicae*, 31, Polish Academy of Sciences, 1997, pp. 49-64.

Michalski, R.S., Kerschberg, L., Kaufman, K. and Ribeiro, J., "Searching for Knowledge in Large Databases," *Proceedings of the First International Conference on Expert Systems and Development*, Cairo, Egypt, April 1992.

Michalski, R.S., Kerschberg, L., Kaufman, K. and Ribeiro, J., "Mining for Knowledge in Databases: The INLEN Architecture, Initial Implementation and First Results," *Journal of Intelligent Information Systems: Integrating AI and Database Technologies*, 1, August 1992, pp. 85-113.

Michalski R.S., and Larson, J.B., "Selection of Most Representative Training Examples and Incremental Generation of VL_1 Hypotheses: the underlying methodology and the description of programs ESEL and AQ11," Report No. 867, Department of Computer Science, University of Illinois, Urbana, IL, 1978.

Michalski, R.S. and Larson, J.B., rev. by Chen, K., "Incremental Generation of VL1 Hypotheses: The Underlying Methodology and the Description of the Program AQ11," Report No. UIUCDCS-F-83-905, Department of Computer Science, University of Illinois, Urbana, IL, 1983.

Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N., "The AQ15 Inductive Learning System: An Overview and Experiments," Report No. UIUCDCS-R-86-1260, Department of Computer Science, University of Illinois, Urbana, IL, 1986.

Michalski R.S., and Stepp, R.E., "Automated Construction of Classifications: Conceptual Clustering Versus Numerical Taxonomy," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 5, No. 4, 1983, pp. 396-410.

Michalski, R.S., Stepp, R.E. and Diday, E., "A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts," in *Progress in Pattern Recognition*, Vol. 1, L. N. Kanall and A. Rosenfeld (Eds.), New York: North-Holland, 1981, pp. 33-56.

Michalski, R.S. and Tecuci, G. (Eds.), *Proceedings of the First International Workshop on Multistrategy Learning*, Harpers Ferry, WV, November 1991.

Michalski, R.S. and Tecuci, G. (Eds.), *Proceedings of the Second International Workshop on Multistrategy Learning*, Harpers Ferry, WV, May 1993.

Michalski, R.S. and Wnek, J. (Eds.), *Proceedings of the Third International Workshop on Multistrategy Learning*, Harpers Ferry, WV, May 1996.

Morik, K. and Brockhausen, P., "A Multistrategy Approach to Relational Knowledge Discovery in Databases," *Proceedings of the Third International Workshop on Multistrategy Learning*, Harpers Ferry, WV, May 1996, pp. 17-27.

Morik, K., Causse, K. and Boswell, R., "A Common Knowledge Representation Integrating Learning Tools," *Proceedings of the First International Workshop on Multistrategy Learning*, Harpers Ferry, WV, November 1991, pp. 81-96.

Paliouras, G., "Scalability of Machine Learning Algorithms," Master's Thesis, Department of Computer Science, University of Manchester, 1993.

Piatetsky-Shapiro, G. (Ed.), *Proceedings of the AAAI-91 Workshop on Knowledge Discovery in Databases*, Anaheim, CA, July 1991.

Piatetsky-Shapiro, G. (Ed.), *Proceedings of the AAAI-93 Workshop on Knowledge Discovery in Databases*, Washington, DC, July 1993.

Piatetsky-Shapiro, G. and Frawley, W.J. (Eds.), *Knowledge Discovery in Databases*, AAAI Press, Menlo Park, CA, 1991.

Piatetsky-Shapiro, G. and Matheus, C.J., Measuring Data Dependencies in Large Databases, *Proceedings of the AAAI-93 Workshop on Knowledge Discovery in Databases*, Washington, DC, July 1993, pp. 162-173.

Quinlan, J.R., "Probabilistic Decision Trees," in *Machine Learning: An Artificial Intelligence Approach, Volume III*, Y. Kodratoff and R.S. Michalski, (Eds.), Morgan Kaufmann, San Mateo, CA, 1990, pp. 140-152.

Reinke, R.E., "Knowledge Acquisition and Refinement Tools for the ADVISE Meta-Expert System," Master's Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1984.

Ribeiro, J., Kaufman, K. and Kerschberg, L., "Knowledge Discovery from Multiple Databases," *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Montreal PQ, 1995, pp. 240-245.

Rosch, E., Mervis, C., Gray, W., Johnson, D. and Boyes-Braem, P., "Basic Objects in Natural Categories," *Cognitive Psychology*, Vol. 8, 1976, pp. 382-439.

Simoudis, E., Han, J. and Fayyad, U. (eds.), *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, Portland, OR, 1996.

Simoudis, E., Livezey, B., and Kerber, R, "Integrating Inductive and Deductive Reasoning for Database Mining," *Proceedings of AAAI-94 Workshop on Knowledge Discovery in Databases*, Seattle, WA, August, 1994, pp. 37-48.

Sokal, R.R. and Sneath, P.H., *Principles of Numerical Taxonomy*, W.H. Freeman, San Francisco, 1973.

Spackman, K.A., "QUIN: Integration of Inferential Operators within a Relational Database," ISG 83-13, UIUCDCS-F-83-917, M. S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1983.

Stepp, R.E., "A Description and User's Guide for CLUSTER/2, a Program for Conceptual Clustering," Department of Computer Science, University of Illinois, Urbana, IL, 1983.

Stepp, R.E., "Conjunctive Conceptual Clustering: A Methodology and Experimentation," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1984.

Strat, T.M., *Natural Object Recognition*, New York, Springer Verlag, 1992.

Tecuci, G. and Michalski, R.S., "A Method for Multistrategy Task-Adaptive Learning Based on Plausible Justifications," in Birnbaum, L. and Collins, G. (Eds.), Machine Learning: Proceedings of the Eighth International Workshop, Chicago IL, June 1991, Morgan Kaufmann, 1991.

Wnek, J., "DIAV 2.0 User Manual, Specification and Guide through the Diagrammatic Visualization System," *Reports of the Machine Learning and Inference Laboratory*, MLI 95-5, George Mason University, Fairfax, VA, 1995.

Wnek, J., Kaufman, K., Bloedorn, E. and Michalski, R.S., "Inductive Learning System AQ15c: The Method and User's Guide," *Reports of the Machine Learning and Inference Laboratory*, MLI 95-4, George Mason University, Fairfax VA, 1995.

Wnek, J. and Michalski, R.S., "Hypothesis-driven Constructive Induction in AQ17: A Method and Experiments," in *IJCAI-91 Workshop on Evaluating and Changing Representations in Machine Learning*, Sydney, Australia, 1991.

Wrobel, S., "Toward a Model of Grounded Concept Formation," *Proceedings of the 12th international Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991, pp. 712-717.

Wrobel, S., Wettschereck, D., Sommer, E. and Emde, W., "Extensibility in Data Mining Systems," *2nd International Conference on Knowledge Discovery and Data Mining*, Portland, OR, 1996, pp. 214-219.

APPENDIX A TECHNICAL SPECIFICATION OF INLEN-3

At present, INLEN-3 is composed of 19 executable modules, which range from collections of knowledge generation operators and the menus to run them to introductory and tutorial screen displays. Another important component is the set of support files that provide primitives for the user interface. We will first examine these components, and then the individual modules of INLEN.

A.1 Support files

The files named **globals.c**, **exp_glob.c**, and **mkb_glob.c** handle the initialization and passing between modules of global or semi-global information. Items referenced in globals.c may be used in any or all of the modules. Items referenced in exp_glob may be used in any of the five modules (data, gem_comp, kdl, varb, varsel) that manipulate the entire variables and/or examples tables. Items referenced in mkb_glob are intended for the three-column menu displays in the Develop and Browse System modules.

Each of these three files performs input and output functions upon a variety of files that are used for transmitting global information between the executables. One such set of information, for example, is the *chn* array which maintains, for many of the executables, a record of where they were entered from. In this way, a module can bring up a menu relevant to where the user came from. If the user selected "Create rules" from the Develop System menu, it makes no sense to give the option of testing a ruleset; the rules do not yet exist.

The file **tptcmac.c** reflects INLEN's lineage from the Pascal language AURORA system. The file, which has shrunk over the years to just one function, contains an implementation of the Pascal IN construct, by defining the function *inset*. Its companion file **tptcmac.h** (and shadow file **tptcmac1.h**) contain a few remaining Pascal-based definitions, such as defining a Boolean data type.

Most of the other support files contain general purpose screen and file input/output functions. With the exception of drawaur and intro, each executable is linked with one of **aqscr.c**, **data_scr.c**, **gp_utils.c**, **gp2.c**, **mkb_scrn.c**, **screen2.c**, or **screen3.c**. These files are very similar to one another, perhaps containing subtle differences for the modules that call them. The file gp2.c, called by the *inlen* executable, is the smallest and simplest. Most of the others contain extensive string, menu, help, screen, and window functions.

A.2 The 19 executable modules

A.2.1 Introductory screens: drawaur and intro

Realistically, the **drawaur** and **intro** executables are not part of INLEN proper. If the user enters the program by typing "inlen demo" rather than "inlen" at the DOS prompt, these screens will be shown before the system reaches its main menu. Once there, the screens can not be reached by backtracking.

Drawaur contains a colorful BGI graphic representation of a sunrise superimposed by the program name and version number.

Intro allows the user to scroll between three screens that outline the basic ideas of INLEN before moving on to the main menu.

A.2.2. Routing control: inlen and aurmain

To a large extent, these two executables control the flow of the program. **Inlen.exe** is the nerve center. When a user first types "inlen", this is the module that runs. It initializes all global variables, and then enters a loop of sending spawn commands to the other executables. The values returned by the spawned programs (using defined constants with self-explanatory names such as GOTO_AURMAIN, GOTO_VARSEL, etc.) tell INLEN what to spawn next, or whether to exit. Thus, if from the *mkb* module, we wish to enter the varb module, *mkb* returns to *inlen* the value GOTO_VARB, and *inlen* then spawns varb. Meanwhile, *mkb* has also set *varb*'s allocated element of the chn array (see Appendix A.1), so that varb will recognize that it was called from mkb.

While **aurmain**, like most of the other executables, is spawned directly from *inlen*, and returns the program's next destination to *inlen*, rather than sending the flow there directly, from the user's point of view this module is a simple one-screen menu, namely the main menu. Aurmain.c itself is a short file, relying primarily on calls to the menu and support functions in gp_utils.c.

A.2.3 On-line tutorial: tut

When the user selects the "Learn how to use INLEN" option from the main menu, an on-line tutorial is extracted from three files. *Inlen.hlp* contains the main menu for the tutorial and information on the advisory module. *Dev.hlp* contains information on developing a knowledge system. *Ldmenu.hlp* presents many of the options made available through the learning and discovery menu.

Tut loads in the entire tutorial at the start through function *chain*. Chain begins reading the information in inlen.hlp, including the screens and menus to be presented. Included in the menu information are details of which lines to display and/or files to chain to for each given menu selection. If a new file (for example, dev.hlp) is to be chained to, chain is then recursively called to read in that file. In effect, a tree of sets of screens (leaves) and menus (branches) is built, and referenced by a set of two-dimensional arrays (the first dimension represents the file being read currently, and the second represents the page or option number within that file.

The organization of the help files is as follows. Each help file is divided into sections by lines consisting of asterisks. The first section consists of menu information, the second contains text that appears in conjunction with the menu generated from the first section, and every subsequent section contains a page of text to be shown given the appropriate menu selections.

The first section of the file contains four lines per menu entry. The first line contains two numbers -- the start and end pages for the selected items (or a pair of 2s if the help pages are in another file). The second line contains the line appearing in the menu presented to the user. The third line contains the text to be appended to the menu path at the top of the screen should the item be selected. The fourth contains the name of another help file to be chained to should the item be selected, or an exclamation point if the help screens are in the current

file. For example, should the user select "What is an advisory system?" and the associated screens be located in sections 4-6 of the current file, line 1 will contain the numbers 4 and 6, and line 4 will consist of an exclamation point.

A line showing start and end pages 0 and 0 tells the reader that it has finished reading menu items.

In the listings of the help screen text, lines beginning with tildes are header lines, and will be shown in red, rather than white type.

Tut.c allows up to 6 help files to be part of the on-line tutorial sequence. There can be up to 6 entries on a menu, and up to 50 pages referred to from a file.

A.2.4 One from column A: mkb and rev

These two modules share the common bond of a three-column selection format in which the first column contains the names of the knowledge system, the second contains the different operations, and the third contains the different system components.

In **mkb**, the available operations (create, access, copy, delete) allow the user to modify the components of a knowledge system. The copy and delete functions are handled in mkb.c, as are the operations on the system descriptions. Others are sent out to the appropriate modules (*aq15* for rules, *varb* for variables, *data* for examples and testing events, *param* for learning parameters, *infpar* for inference parameters, and *comp_dis* for "accessing" a whole system).

In **rev**, the available operations (view, print) allow the user to explore the contents of a knowledge system without altering anything. All print functions are handled in rev.c, largely through functions *do_chosen* and *prnt*. Rules and system descriptions are viewed in rev.c; others are sent out to the appropriate programs, as is done in mkb.

In mkb, the functions *do_access*, *do_create*, *do_copy* and *do_delete* sort out the tasks to be performed based on the second column selection. Each contains a case statement based on the selected item in the third column.

Both executables use the functions in get.c to read in information on the available knowledge systems and their assigned domains from file *systems.aur*, and mkb.c rewrites the file (in function *rewrite_sys*) should a system description be added, changed or removed. Each entry in the file consists of three lines -- the name of the knowledge system or domain, the short description of the system (or a line of asterisks if it is a domain), and the name of a file containing a longer description of the system (or another line of asterisks in the case of the domain). Knowledge systems assigned to a domain are ordered alphabetically by name directly after their domain entry. The domains themselves are also ordered by name. Domain names are maintained in the array *domain*, and their positions among the system names are maintained in the *dompos* array. System information is maintained in the arrays *sysname*, *sysshrt* and *syslong*. The array *page_params* is generated after this information is read in; it holds the starting domain and system numbers for a given page.

A.2.5 Taking inventory: comp_dis

If "access whole system" is chosen from within mkb, the **comp_dis** executable is called upon to display the completion status of the various portions of the selected knowledge system.

The display is in a window created by this module and destroyed when the user leaves it. On the checklist it displays are system description (always present), rules (for at least one decision attribute in the knowledge system), compiled rules (for at least one decision attribute), learning event file (.gem), training examples, testing examples, and variables. It also indicates whether learning and inference parameter files are present, or if the defaults are being used.

A.2.6 Data table maintenance: varb, data and varsel

The common bond of these three executables is their use of the full variable and example tables, as defined in exp_glob.c. Because of their size and the allocation limitations of Borland C++, the larger structures are allocated piecemeal at run time. As a result, the fifth row of the third column of the examples table is addressed as (*dat_array[5])[3].

Examples and training examples are normally stored in (*dat_array[row])[column] and (*dat_a2[row])[column], but their assignments are reversed in the examples editor when the testing examples are the ones being actively edited (this is determined through the *chn* variable). The array exst[column][row] (note that the subscripts are in standard (x,y) order instead of the (example, column) order) holds Booleans indicating whether a given value has been assigned. Similarly, val_exst[column][row] applies to value entries in the variables table.

The variables table information is more distributed. Arrays var_name, Var_type (note capitalization), var_cost and var_quest maintain the names, types, costs, and questions associated with the variables respectively. The array (*var_array[row])[column] contains structures with fields representing the name of the value (val_name), a help file associated with the value (val_help_file), the structure level of the value (val_struct), and display and execution files should this value be selected by the advisory module (val_disp_file, val_prog_file).

The **varb** executable allows the user to view or change the variables table, and most of the functions therein focus on the display of the screen, the user's moving through it, and basic integrity checking (such as making sure a variable is assigned a legal type). Variables that assist in this capacity are row and col (screen coordinates), rw (row in the table at which the cursor is positioned: -2 is the variable name row, -1 is the variable cost or type row, and nonnegative values represent the n+1st example row), cl (column in the table at which the cursor is positioned), tpbnd, btbnd, ltbound and rtbound (topmost and bottommost rows; leftmost and rightmost columns currently appearing on the screen).

The examples editor (**data** executable) works in a similar fashion, with similar variables. The difference is that the table the user is reading or changing consists of examples, rather than attribute definitions. There is an additional column that the user may choose to view (at the cost of one regular column) or not to view that contains keys for the examples -- (supposedly) unique identifiers that are not used in the learning process itself. *Data* performs additional integrity checking; it makes sure that values entered are legal for the variables, although simple omissions may be corrected on the spot, or the user may move directly into the variables editor to make corrections.

The **varsel** executable performs the duties of the attribute selection operator. The attributes are selected based on their decision-making utilities, rather than by the user. The user

specifies the parameters by scrolling through a menu of choices in function *select_par*, and then function *VARSEL* performs the attribute rating calculations.

Varsel employs some data structures unique to its needs. The *attribute* structure maintains information about an attribute including its name, its position in the variables table, and its raw and normalized promise scores. The *aclass* structure maintains information on a decision class, such as the number of examples of the class and the number of examples of the class that have known and unknown values for a given independent attribute.

After attribute utilities are calculated, the variables are sorted in descending order of promise. The user can then view, accept, reject or modify the suggestions of the operator through an interface similar to that in the variables and examples editors.

These three executables (as well as kdl, see Appendix A.2.13) rely on two support files for writing the variables and examples tables to INLEN files. The file **mk_range.c** contains the software for applying the Chi-merge algorithm for discretizing a numeric variable based on its classified examples. One data structure used to this end is a doubly linked list of *INTVREC* structures. At any point, this list represents the different intervals into which an attribute has been discretized. Each one includes upper and lower bounds, the interval's chi value, and a histogram of the classes whose examples have a value of the attribute under examination falling into the given interval.

The file **make_gem.c** creates a .gem file (which lists the examples in AQ input table format) whenever variables or examples are saved. The code appears complex because the program must pass through the examples, class by class, taking into account the possibility of a structured decision attribute, for both training and testing examples.

A.2.7 Learning parameter editing: param

The **param** executable is a stand-alone module for editing or viewing a knowledge system's learning parameters (as opposed to the nearly identical **aqpar.c**, that operates from within the aq15 module, see Appendix A.2.9). Several factors make this straightforward module more complex than it might otherwise be. First, input is taken from the .lrn file if it exists, and otherwise from the default file learn.aur. More importantly, INLEN's parameters files are stored in AQ parameters table format, making the reading and writing of such files more difficult. Finally, the lower portions of the parameters screen (level of specialization and Lexical Evaluation Function criteria) are automatically dependent on the learning mode if the mode is not "user-defined". In the case of user-defined learning, the user may scroll through and change those items.

A.2.8 Inference parameter editing: infpar

Unlike param, **infpar** is maximally straightforward and simple. The .inf files are flat text files; one parameter is represented by a single letter, all others by numbers. Defaults are not read in from a file, but rather defined within the program.

A.2.9 Rule learning and testing: aq15

The source files that comprise the **aq15** executable can be divided easily into two groups: the files unique to INLEN that make up the menus, selection screens and algorithms for converting data to AQ input format, and INLEN's implementation of AQ15c. Because

AQ15c serves as a stand-alone program and is documented elsewhere, the latter files will be discussed in less detail than a full description would provide; instead their descriptions will focus on the differences between the two versions, as well as describing some of the most critical structures.

The source files unique to INLEN include *aq15.c* (menus encountered upon entering the module), *aqscreen.c* (detailed user specification of the problem after the task has been selected in aq15.c), *aqscr.c* (screen, menu and support functions), *combo.c* (combining domains from two knowledge systems), *aqpar.c* (a close copy of param.c to modify the learning parameters), and *aqscrn1.c* (reading INLEN files and generating an AQ input file). Of these, only aqscrn1's duties are not straightforward, and mirrored in other parts of INLEN.

Aqscrn1 has three base functions: *learn_read* (used when learning or improving rules from examples), *optimize_read* (used when optimizing a ruleset according to a user-provided set of criteria), and *test_read* (used when a user is testing rules). Each of these calls several functions to build the appropriate input file. *Read_var_file_into_gem* reads a .var file into an internal data structure, while taking into account structured variables spread over multiple columns. *Domain_types* writes the domaintypes table, and *variables_table* generates the variables table. *Var_names* generates the individual names and structure tables. As a side effect, the variable *size_of_complex_in_bits* is updated to hold the aggregate number of values in all of the attributes. The function *create_bytes* in aqset.c is then called to allocate and initialize aq15's *bits* variables. *Insert_params* loads the parameters and criteria tables, and then the event tables are added directly from the .gem file that was created when the examples or variables were saved. Finally, if any input hypotheses are to be used (incremental learning, rule optimization, or rule testing) the function *convert_rules* parses the appropriate .crl file and loads the rules into the AQ input file.

The key data structures in AQ15c are the *BITS*, *COMPLEX* and *NODE* structures. BITS is simply a string of bits, for representing a set; the ith bit represents the ith element of the set, with 1 indicating presence and 0 indicating absence. Most occurrences of BITS have a length of *size_of_complex_in_bits*, so that they can indicate any arbitrary description in the defined attribute space. Occasionally, a BITS structure will be of size *nvarsgbl* (the number of attributes in the domain); in these cases, they are maintaining information on which attributes have a given property (e.g., which attributes are referred to in a given extended selector). These structures are accessed and maintained chiefly in the file aqset.c, which includes functions for such operations as initialization, testing for the presence of a given element or set of elements, union, intersection, set difference, setting a given bit or set of bits, operations on individual selectors (attributes), etc.

Another key data structure is the *COMPLEX*, which can represent a rule, a condition, or an example. Each has a *BITS* structure representing its description. Additionally, rules and conditions have information on examples covered and a rule's utility according to the user-specified evaluation criterion. Examples have information on the rules that cover them, as well as their identifying keys.

The *NODE* structure represents a decision class. Various fields of this structure relate to a the training and testing events and input and output rules associated with a class, the name of the class, and any children classes (in the case of a structured decision attribute).

INLEN files *util1.c* and *util.c* (cover generation), *inptools.c* and *coverpro.c* (processing of background rules), *aqcore.c* (main rule generation) and *aqlearnu.c* (initialization and invocation) correspond to the AQ15c file aq.c. INLEN files *init1.c* and *init2.c* correspond to the AQ15c file setup.c. Beyond those, there is a 1-1 correspondence between INLEN and AQ15c files: *aq15util.c*, *aqlist.c*, *aqset.c*, *atest.c* and *outptool.c* with aqutil.c, list.c, aqs.c, atest.c and postpro.c, respectively.

The major differences between INLEN's and stand-alone AQ15c versions are the processing of structured attributes' anchor nodes in INLEN's setup and postprocessing (function trimcover) stages, some sacrifice of efficiency for readability of code, the fact that ATEST only operates in one evaluation mode, and the differences in required output. INLEN must return all of the related pieces of the knowledge segment to be created (e.g., various weights). Another difference lies in the fact that the uniclass and TRUNC/NL features have not yet been implemented in INLEN.

A.2.10 After the learning: gem_comp

The **gem_comp** executable is chained to from aq15 after successful rule learning or optimization. Its screens allow the user to view rules and the examples they were generated from (through a smaller version of the example editor). The user can then choose whether or not to save the rules. If so, they are moved from the temporary rule file *last.rle* into the knowledge base. The knowledge base's Session structure is also updated with the timestamp of the new ruleset.

Because of the capability of viewing training examples, the data structures defined in *exp_glob* (Appendix A.1, A.2.6) are used to hold the variable names and the example table. They are read in similarly to their input in data.c.

Key functions In gem_comp.c are *display_rule* for viewing a rule, *view_examples* and *move_it* for viewing and scrolling among the examples, and write_out_rules for updating the session file and creating the .rle (rule) and .rlp (rules in a form for a reader) files. In the latter function, each line of the session file is read into a *SESS* data structure. If a line's decision variable field matches the one just learned, that line is updated. Then the information is written back into the session file. Following the revision of the session file, lines from last.rle are converted one by one into the format of the two rule files.

A.2.11 Rule compilation: create

The rule compiler, **create**, may be the most complex of INLEN's 18 executables. From the information in a knowledge system's .rle files, it generates .crl files (similar in format, but with recalculated distinctiveness values), .cin files (index files for the .crl files), and .adv files (advisory system guide files).

The .adv file is divided into several sections. The first section consists of cross-reference tables for every value of every independent attribute that appears in a rule. For each such value, there are sets of five numbers for each time the value is cited in a rule. The first three numbers represent the class, rule and condition numbers of a given citation. The next two numbers represent the distinctiveness (a representation of how unique the value is to rules for the given decision class -- it is the fraction of classes whose rules cite the value at least once)

and commonality (fraction of the training examples of the decision class that have the value), both listed in three digits, with 999 representing 100%.

The next portion of the .adv file lists, for each decision class, the variable numbers of the attributes that appear in the ruleset for that class. The third section lists in order the calculated informativeness levels for each attribute, based on their values' distinctiveness values. The final part of the .adv file is devoted to the definition of arithmetic expressions to be used by the advisory module; this feature is not currently implemented in INLEN.

The creation of the .adv is done in three major steps. The first, centered around functions *parse_rule* and *parse_condition*, reads in rules from the .rle file, one condition at a time, and loads the conditions into "selector" data structures, which maintain the information on each condition -- the variable, relation, values, and weights.

The second step, centered around function *save_rule*, dissects the relations and value sets in the selector structures. In other words, when a condition is written "$x \leq y$", it will change the value listing to an explicit list of all values up to y. Similarly, a condition written "$x \neq y$ or w to z" will have a value list generated of all values other than y that are outside of the w-z range. Once the values are sorted out, each value is put onto a list associated with the corresponding attribute, along with its class, rule and condition numbers and its distinctiveness (if provided by a user) and commonality weights.

The third major step, centered around function *write_system*, builds the .adv file. The first part of the file, the variable/value listing, is generated from the structures generated in save_rule. If distinctiveness values were not provided by a user, they are calculated dynamically at this point for each attribute value. The informativeness levels of the attributes (listed in the third section of the .adv file) are based on the sum of their values' distinctiveness levels.

Two separate source files are also part of the create executable. *Exparse.c* contains code for parsing arithmetic expressions (not currently implemented in INLEN). *Wrtcrl.c* generates output rule (.crl) and index (.cin) files.

A.2.12 The advisory module: advv

The **advv** executable is second only to aq15 in size; it performs all of the tasks of the advisory module. Its source files are apportioned as follows:

Advv.c is a small file primarily for starting the advisory process. It performs much of the initial dynamic data allocation required by the module. It also invokes the calls to functions *get_system* (in which the user selects a knowledge base), *read_tables* (in which the knowledge base is read in), and *Control* (which performs the actual advisory session).

Padvmod.c contains the code for selecting a knowledge base (both the knowledge system and the decision variable) to use, and for reading in the inference parameters to be used. *Advvutil.c* contains some utility functions for the modules, including functions for handling the advisory system's window scheme, and functions for reading in attribute information from the .var file.

Progmod.c contains the core of the advisory system, including the three-stage inference process, code for calculating degrees of confidence, and functions for many of the end-of-

session options including answer changing, session recording, and advice execution. *Upplan.c* holds the functions for updating the advisory system's "plan" on the screen as the session proceeds.

Get_ans.c is used to read an answer from the user. If the attribute is of integer or numeric type, the user types in the value rather than scrolling through a menu of choices. The functionality for prompting and reading an answer of one of these types is contained in file *do_numbe.c*.

Vrule.c contains the code for displaying a histogram of decisions under consideration and their confidence levels. *Drule.c* is similar to viewrle.c (in module rev) and the display_rule function within the gem_comp module in that it shows user-selected rules one page at a time. However, drule also contains a highlighting ability so the user can view easily which values have satisfied which conditions.

An advisory session is a three stage process, consisting of *reduction*, *discrimination* and *confirmation*. During reduction, the user is asked for the values of variables whose importances have been defined as 10. The purpose of this phase is to remove immediately from consideration any hypotheses that contradict the user's input. For each rule, INLEN keeps track of whether it is still in consideration due to no contradictions with the user's answers. As values are given, the degree of match for rules still under consideration with conditions satisfied by the values are increased based on the conditions' distinctivenesses.

At the end of reduction, decisions with rules which have not been eliminated have their degrees of match calculated by taking the maximum or the probabilistic sum (based on an inference parameter) of their active rules' degrees of match. Meanwhile, distinctiveness values for the attributes without an importance of 10 are recalculated based on the rules that are still in consideration.

If more than one hypothesis remains after reduction, INLEN continues to the discrimination phase, in which the effort is to single out one outstanding candidate hypothesis from the set of hypotheses still under consideration. The order of questioning during the discrimination phase will be based half upon the recalculated distinctiveness values, and half upon the importance values for the attributes provided during definition of the domain. Each answer during this phase adjusts the degrees of match of the active rules whose conditions refer to the attribute in question, based on the distinctiveness of the answer and the *distinctiveness confidence* inference parameter. Eventually, the *ratio threshold* (specified by an inference parameter) between the leading and the next candidate is satisfied, and INLEN can move on to confirmation.

The purpose of the final phase (confirmation) is to confirm or disconfirm the leading hypothesis; if it is disconfirmed, a new leading candidate hypothesis is selected, and confirmation continues. During this phase, the questioning of the user continues (one possibility for future implementations of INLEN is to make commonality a factor in the ordering of questions during the confirmation phase, but at present, it simply picks up where discrimination left off), only now degrees of match are updated based upon commonality rather than distinctiveness. The rationale for this is that the goal of this phase is not to distinguish between candidate hypotheses, but rather to confirm or disconfirm the leading candidate; in doing so, answers that are more representative of the hypothesis will more

strongly support the confirmation. Rules' degrees of match are updated based on the *commonality constant* and the leading candidate's confidence value is based on the *ratio constant*. Confirmation continues until all questions are exhausted, the degree of confidence of the leading candidate hypothesis exceeds the *confidence threshold*, or the candidate is disconfirmed (i.e., its degree of match falls below that of another hypothesis.)

When the .adv file is read in, much of the information contained therein is stored in an array called *table*, which has one element per attribute. The structures in the table array include a doubly linked list of the values of the attribute; for each value the structures store the name and associated help file, structure information (level, parents, children), and a list of *info* structures which cite every class, rule and condition in which the value appears, along with the associated distinctiveness and commonality weights.

During the questioning process, the names of the attributes being asked about are added into the *question* array. The user's answers go into the *answers* array, and their ordinal positions into the *state* array. Confidence values are stored in the array *rule[class#][rule#]*, which also maintains whether each rule is still in consideration after the reduction phase. These values are updated after each answer in the function update_rules.

A.2.13 KGL language interpreter: kdl

The **kdl** executable contains the code for selecting and processing a KGL program. Lines of the program are interpreted in the function *interpret*, which farms out the processing to other functions (named *do_<command name>*) depending on the command being performed.

When a reference needs to be evaluated, the procedure begins with a call to function *parse* in order to generate a parse tree of the string representing the reference. The function *eval* is then passed this tree in order to determine whether the reference is a literal, a function, or a reference to a variable, and then (possibly recursively), it processes the reference and returns its value in the form of a *VLREC* data structure.

A VLREC contains information on the value's type (real, string, or formula to be evaluated when encountered), the value itself, and two pointers, so that these structures can be agglomerated into a binary tree. In fact, variables in a KGL program, whether user-defined or system-defined (e.g., number of attributes in the data table being currently examined; system-defined variables' names are entirely upper-case), are stored in a binary tree, sorted ASCIIbetically, and may be referenced or initialized through the function *find_var*. Currently, there are no functions to ensure that the tree maintains any degree of balance.

There are times when *eval* will encounter special functions (either a name followed by its arguments in parentheses) or a name beginning with a # sign. In the first case, control is passed to function *eval_funct*, which is designed to handle functions such as max, min, avg, var_name (name of the nth attribute in the data table), and various attribute type Boolean functions.

In the latter case, the reference is instructing the interpreter to count the number of occurrences of the specified type in the knowledge base. Function *eval_count* reads the specified portion of the knowledge base, and then hands the reins back to *eval* as necessary to determine whether the specified conditions apply to each rule or condition under examination.

Eval_count may also be accessed through the "forall" command. In this case, rather than maintaining a count of the knowledge structures that satisfy the given specifications, the interpreter performs the block of code following the forall whenever it encounters such a knowledge structure.

INLEN's knowledge generation operators are invoked from within KGL through the "do" command. The function *do_do* passes control to the function handling the particular operator being called. These functions create parameter files for the operator based in order of precedence on the specifications provided in the KGL command, an existing relevant parameters file, and system defaults. Function *prepare_spawn* cleans up the environment by closing active files, and then calling out to the appropriate executable.

When another executable is called from the KGL program, its local *chn* value will have been set to a number in the 150s. This signals to the module that it should not interact with the user normally, it should rather run in the background and automatically save results as specified by the KGL program. The KGL interpreter will take the responsibility for updating the user on its status, and it will record results as appropriate in the file kgl.log.

A.2.14 Statistical report generation: stat1

The **stat1** executable is used to generate a statistical analysis of the data and add this report to the knowledge base. Four tables are generated, with dynamically allocated size based on the data set. The first table holds means (of numeric non-decision attributes) and modes (of non-numeric non-decision attributes) for each decision class, and for the training data as a whole, using a variant record structure that can contain either strings or reals. If no mode exists, an empty string is used in the appropriate cell in the table. If no mean exists (due to no known value of the attribute in the training examples for the class), this is represented by a negative value (we can get away with this since INLEN currently does not have the capacity for negative-valued numeric data).

The other three tables contain real-valued entries only. The second table holds the variances for each class and the entire data set of all numeric non-decision attributes. Since the variance is by defiaition non-negative, a negative value is used to indicate the absence of a variance. The third table contains covariances between all pairs of numeric attributes. Since the range of possible covariances spans the range of real numbers, the unlikely-to-occur value of -11111.1 is used to flag an undefined covarience. The fourth table contains the correlation coefficients between all pairs of numeric attributes. Since the maximum absolute value a correlation coefficient can have is 1, the value -2 is used to flag undefined correlations.

The functions *get_modes*, *get_means_vars* and *get_covs_corrs* are used to generate the values in the appropriate cells of the appropriate tables. Function *write_stats* outputs the four tables to a knowledge base file, while function *display_stats* manages the selection, display and scrolling of the tables on the screen.

CURRICULUM VITAE

Kenneth A. Kaufman was born on February 9, 1961, in Hartford, Connecticut, and is a U.S. citizen. He received his B.A. in Mathematics from Wesleyan University in 1982 and his M.S. in Computer Science from the University of Illinois at Urbana-Champaign in 1986. Among his honors and awards were the Rice Prize in Mathematics from Wesleyan University and selection for the Honor Society of Phi Kappa Phi. He has served as a software developer and consultant for such employers as the US Army Construction Engineering Research Laboratory and Computer Sciences Corporation.

Along with his involvement in the development of the INLEN knowledge discovery system, Ken was one of the original developers of EMERALD, an integrated system for education and research in machine learning, and its predecessor ILLIAN, which was presented at the exhibition Robots and Beyond: The Age of Intelligent Machines at eight U.S. science museums, and is estimated to have been viewed by over half a million visitors. He has also been instrumental in the implementation of the AQ15c rule learning program.

His work has appeared in the volume *Knowledge Discovery in Databases* and in the *Journal of Intelligent Information Systems*, and he has collaborated on a chapter to apper in the volume *Methods and Applications of Machine Learning, Data Mining and Knowledge Discovery* . He has presented his work and collaborations at conferences, workshops and seminars on three continents. He has been involved in the organization of the three Workshops on Multistrategy Learning (MSL-91, MSL-93, MSL-96).

Ken's Ph.D. work was supervised by Ryszard S. Michalski, and supported by the Machine Learning and Inference Laboratory at George Mason University. The Laboratory is supported in part by grants from the Defense Advanced Projects Research Agency, the Office of Naval Research, the National Science Foundation, and other organizations. His research interests include machine learning, knowledge discovery in databases, and multistrategy learning.