To the Graduate Council:

I am submitting herewith a dissertation written by Min Ke entitled "Inductive Inference in an Intelligent Database System." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

J. Douglas Birdwell, Major Professor

We have read this dissertation
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of the Graduate School

INDUCTIVE INFERENCE IN AN INTELLIGENT DATABASE SYSTEM

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Min Ke
December 1992

DEDICATION

This dissertation is dedicated to those people

who make this world a better place to live.

ACKNOWLEDGMENT

The author would like to express his appreciation to many people who have helped in various ways during the process of conducting and completing this dissertation.

First, the author would like to thank his wife Suili and his son Sheen-Sheen for their great sacrifices during the long course in his pursuit of the Ph.D. degree.

Special thanks go to the author's major professor and committee chairman Dr. J. Douglas Birdwell. His reading and commenting on each chapter of this dissertation has led to great improvement of the content and quality in the writing, and his guidance and encouragement in the course of this study have been most valuable and helpful.

Great appreciation also goes to Dr. B. A. Kupershmidt for his encouragement and constructive comments which were important to this research. Helpful advice and constructive comments from Dr. Michael G. Thomason are appreciated. The author would like to thank Dr. M. Vose's efforts in helping to improve this dissertation. Encouragement and helpful advice from Dr. M. M. Trivedi are appreciated.

The author would like to express his appreciation to Dr. T. W. Wang for her help in the application domain of chemical process monitoring. Great thanks are also extended to Mrs. Mary Lo for her help in literature searches, and to Ms. Ethel Wittenberg for her help on many occasions.

The author would like to thank Dr. J. M. Wu for his great help and encouragement. The author also would like to thank Dr. K. C. Reddy, Dr. C. F. Lo and Dr. C. W. Minkel for their help and encouragement. Finally, the author would like to thank his previous major professor Dr. Moonis Ali for his time and effort on this dissertation.

ABSTRACT

Enhancing a database management system with inductive inferential capabilities makes it possible to extract new knowledge from a large amount of data stored in a database. There are special requirements for conceptual induction in a database environment: (1) the large amount of data and incremental insertion of new data require an efficient incremental induction method; (2) descriptions of complex entities and utilization of knowledge require an expressive representation language; and (3) the large variety of application domains requires a general method of induction.

The main results of this dissertation are a conceptual inductive model called the Knowledge-Directed Model (KNOWD model) and a general methodology (called the KNOWD methodology) for conceptual induction in a database environment. There are three major innovations in this research work: First, the model supports an expressive representation language which allows efficient incremental processing. The model represents a concept description by a set of logical expressions, and this representation makes it possible to fulfill the above requirements. Second, the methodology has a unique control strategy which combines basic abstraction and advanced abstraction so that the conceptual induction can be either supervised or unsupervised and can address a broad range of applications. Third, as a general inductive method, the KNOWD methodology explicitly contains a bias management system. This provides comprehensive management of a variety of inductive biases including general heuristics and domain specific knowledge, so that this methodology can be applied efficiently to a wide spectrum of situations. The KNOWD model employs domain knowledge for directed searching for relationships among concept features instead of assuming that all features are independent or are related.

Both the KNOWD model and the methodology have been implemented, providing an inductive database system (IDB1). The inductive database system has been tested on several applications, and the results demonstrate the high representation capability, inductive quality and efficiency of the methodology.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

LIST OF TABLES

TABLE PAGE

CHAPTER 1

INTRODUCTION

In this chapter, we first describe the research goal and tasks. Next, we discuss the importance of our research work. Then, we present formal descriptions of database models. Last, we describe our approach to the research goal.

1.1. Inductive Database and Research Tasks

In this section, we define our concept of an inductive database. Using this we describe the focus of this research work.

A database is a collection of stored data that can be used for two fundamental purposes: (1) for retrieving data on a particular record or a group of records, and (2) for deriving or inferring information (generalization) that is not explicitly stored in the database. An example of the first purpose is obtaining the grades for a particular student, or the grades of all students for a class. An example of the second purpose is requesting a general description about an engine fault based on the data stored in a database of engine-fault testing records. The answer to this query is not stored in any record in the database, but is induced from many specific instances of the engine faults. The first purpose is well-served by current database systems; however, queries of the second type are performed by people instead of the database systems. In order to handle the second type of queries, a database system must possess reasoning, or inference, capabilities.

There are two types of inference: inductive inference and deductive inference. **Inductive inference** is a process which hypothesizes general (conceptually higher level) knowledge from many specific (conceptually lower level) facts. Hypotheses are kept which are supported by facts, where "support" may be interpreted differently by different systems. The information generated

by induction may not be true, but can be viewed as a generalization of existing knowledge. **Deductive inference,** in contrast, generates only knowledge which can be proved true from available facts. The results from deduction are new facts, or theorems, which can be added to stored knowledge.[1] A database stores specific facts, such as: "The pressure measurement at time 10 seconds was 513 psi." Only simple relations between these facts are known a priori, such as the coincidence in time of two measurements or multiple measurements of the same value. We want to derive general knowledge based on the data in the database which describes more complex relations, such as a relationship between temperature and pressure. We concentrate on inductive inference methods to accomplish this. Since a database is an environment with more data and less explicit knowledge, it provides a better situation for applying inductive inference than deductive inference for getting useful information.

Conventional database management systems (DBMS) can efficiently store large quantities of rigidly formatted data, but they have no facilities to manage and use inferential knowledge, such as rules. Conventional DBMSs can answer only precise queries about data stored in the database or about arithmetical aggregation functions, such as SUM, AVERAGE, MIN, and MAX, but in many situations users wish to ask imprecise questions. One application of an inductive database is Rocket Engine Test Analysis, where a large amount of data from hundreds of tests has been accumulated in databases. The data is in the form of numerical values of sensor parameters for certain faults. An analyst needs to know not only the specific data of a specific test but also general descriptions about a certain fault. For example, the following queries can be asked:

"Find all the unique features for valve failures."

"What are the common features for the fault f1?"

"Find all the features in all faults closely related to a valve failure."

"Find the features of all the sensors with more than 3 significant events in the fault f1."

---

[1] Deductive inference is truth-preserving. Inductive inference is not truth-preserving, and the resulting knowledge is only plausible with high certainty. The truthfulness of the resulting inductive assertions is assumed to be judged by human experts.

"Find the features of the sensor s1 in all related faults."

"Find the relationship between sensor HPFT-DS-T1 and sensor HPOT-DS-T1 in all MCC-MANIFOLD failures."

In this example, general patterns of sensor behavior and fault characteristics are not stored in the database but must be inferred from test data. Inference in conventional DBMSs is performed only through natural joins or theta-joins [2] of existing relations, and this kind of inference is not capable of addressing the imprecise queries outlined above.

An **inductive database** is a database that has an inductive inference capability to generate new knowledge from the data. It should be capable of generalizing from many individual data values, of generating decision rules from examples, and of finding the concepts from instances. It should be able to generate answers to the imprecise queries as outlined above using information induced from stored data. We call a database with inferential capabilities an **intelligent database.**

Database applications permeate our lives. People are faced with huge amounts of data for analysis. The data stored in databases have become an important source of knowledge. This knowledge, however, is difficult to interpret and use because the majority of the data represent only stored measurements, and not concepts which describe their interrelationships. With this situation in mind, this dissertation research has the following **goal**:

We wish extend a database system using inductive inference and automate the process of generating concept descriptions (or classification or decision rules) from stored data.

In order to achieve the above goal to implement extensions to a conventional database system using inductive inference, we have performed the following tasks:

(1) Develop an induction model in a database environment;

(2) Develop an induction methodology in a database environment;

(3) Implement an inductive database system; and

[2] Natural joins and theta-joins are relational algebra operations which can be applied to two relations to generate a third relation based on arithmetic comparisons between attributes (refer [Ullman 1982] for details).

(4) Apply the inductive database system to several applications.

Currently the trend is to integrate artificial intelligence technology with database management systems. We believe inferential capabilities are the most important extensions to the database systems. An intelligent database has a distinctive capability which allows answers to a query to be derived from existing data and knowledge by various kinds of inference. During the inference process, rules are accessed, examined, selected and executed, and facts are read from and written into the database. As a result, to support inference, a database must manage three basic classes of objects: facts, rules, and metarules. New functions[3] are needed in addition to the classical database query functions. In addition to database techniques, an intelligent database employs artificial intelligence techniques, such as an efficient control structure and representations for deductive and inductive inference. With the objective of making a contribution to the integration of artificial intelligence and database system technologies, specifically, the design and implementation an inductive database, this dissertation focuses on:

The development of a new model of conceptual induction and an efficient general methodology of knowledge-based conceptual induction in a database environment.

1.2. Importance of Inductive Databases

In this section we answer the question: Why are inductive databases important?

1.2.1. Research Trend

The research reported in this dissertation is a part of the effort to integrate Artificial Intelligence (AI) and Database Management Systems (DBMS). This integration has become a trend in both application and theoretical research areas [Brodie et al. 1986; Mylopoulos 1989; Ullman 1988; Hillyer 1986; Parsaye et al. 1989; Kerschberg 1986; Schmidt et al. 1988; Brodie et al. 1984; Gallaire et al. 1987; Deering 1986; Shin 1988; Bert el al. 1988].

[3] For example, functions executing recursive rules to answer some queries.

In addition to the need for developing the next generation of intelligent information systems [Brodie 1989], the motivation for integration also comes from limitations of both AI and DBMS when they stand alone. AI systems must manage large knowledge bases which contain facts, rules, or frames. Meanwhile, DBMSs users demand richer semantic modeling and intelligent processing of data. In the integration of AI technology with a DBMS, we believe inferential capabilities are the most important extensions of a DBMS.

An increasing number of application areas, including equipment test analysis, medicine, CAD/CAM [Adiba 1985; Smith 1984], software engineering [Rine 1988], financial applications [Risch 1988; Shaw 1990], business applications [Jarke 1984], and military command and control require intelligent databases which have a knowledge-directed processing capability using a large amount of shared information (data as well as knowledge). For example, RX by Blum [1982] is a computer program which finds causal relationships through analyzing a time-oriented clinical database by utilizing a knowledge base which contains knowledge about medicine and statistics. The resulting discoveries are automatically incorporated into the knowledge base, so that the learned causal relationships can be used to direct further analysis. Scott is a software system which combines an expert system with a relational database [Modesitt 1988] and makes it possible to transform historical data from hundreds of Space Shuttle Main Engine tests automatically into production rules which can be used to facilitate further test design and test analysis. The Naval Tactical Situation Assessment System [Smith 1984] requires that a DBMS store both static and dynamic information, check hundreds of situation predicates quickly, and support many expert systems to analyze threats and alert users in real time. The Automated Map Production System [Smith 1984] also allows many expert systems to retrieve and update information in a DBMS simultaneously. The Decision Support System [Karagiannis and Scheider 1987] requires a deductive inferential capability and a natural language facility for processing information in a database. In this system, inferential knowledge, natural language knowledge, meta knowledge, and other kinds of knowledge are stored in a knowledge base which cooperates with the database to support decision making.

Many research efforts have been devoted to deductive inference for DBMSs as shown in the work by [Leung and Lee, 1988; Chang and Walker 1984; Tsur 1986, 1988; Eick et al. 1988; Missikoff and Wiederhold 1984; Deering 1984; Risch et al. 1988; Delcambre 1988; Gardarin and Valduriez 1989]. On the other hand, inductive inference is also important in many applications, but little work has been done on its incorporation with DBMSs. Inductive inference makes it possible to automatically discover unknown features and relationships which exist in a large database. With inductive inference, new knowledge can be discovered in databases. Parsaye [1989] pointed out that the use of machine learning tools to discover knowledge from large databases signals the beginning of a new era in the applications of artificial intelligence.

1.2.2. Knowledge Acquisition

Currently there is an explosive growth of interest and social need to develop expert systems in many areas. The expert system is a computer program that uses expert knowledge in solving problems in a specific domain with expert-level performance. Expert systems have already provided substantial benefits in many sectors of the society.

Development of an expert systems requires a process of encoding expert knowledge into the system. This process is called **knowledge acquisition.** Traditionally, this process involve interaction between domain experts and knowledge engineers. Experts explain their knowledge, and the knowledge engineers formalize the knowledge into the program representation. This approach has several problems as pointed out by Shalin [1988]:

(1) Knowledge acquisition is a time-consuming and difficult process. Developing an expert system requires enormous effort.

(2) Experts are usually not trained to explain to others their own knowledge and decision making processes in a formal way so that they can easily be represented in a program.

(3) Knowledge acquisition is an error-prone process, and it is difficult to determine whether the knowledge base is correct.

Therefore, knowledge acquisition is the bottleneck in the development of expert systems.

Inductive databases can simplify the knowledge acquisition process so that a system is able to generate inference rules from historic examples accumulated in databases through automatic analysis. Databases are important sources of knowledge, and should be utilized in knowledge acquisition [Hayes-Roth et al. 1983]. An extension of databases with inductive inference capabilities will facilitate extracting knowledge from databases.

Inductive learning techniques can facilitate knowledge acquisition [Shalin et al. 1988] in several ways: (1) They can aid the initial construction of knowledge bases. (2) They can help to refine existing knowledge bases (that is to detect and correct inconsistent rules, remove redundancies, and simplify rules). (3) They can adapt knowledge bases to incorporate users' expertise. (4) They can replace traditional complex knowledge acquisition approaches with an automatic mechanism.

1.2.3. Conceptual Data Analysis

With the rapid penetration of database systems in the market place, many sectors, including government, business, medicine and engineering, have been flooded with oceans of data. Database systems only provide a means to store, organize and retrieve the data, but there is a need for using the data in new, creative and productive way. More people have to cope with the overwhelming amount of data for making everyday decisions. Databases are potentially important sources of knowledge which can benefit the society. To make use of the knowledge that is implied in the database, people have to analyze the data and extract the knowledge. For example, in a financial application a decision support system may need a general evaluation of a certain investment risk. However, this evaluation is not explicitly stored in the database but must be inferred from a large amount of data in the database.

In order to use data effectively, people perform conceptual analysis on the data. **Conceptual data analysis** is a type of data analysis that generates conceptual descriptions of relationships existing in the data by using various types of inference.

Conventional DBMSs can answer only precise queries about data stored in the database, but cannot provide inference capabilities. An inductive database can provide assistance to people

requiring conceptual data analysis. A database with inductive inference is important because the number and size of available databases is growing so fast that there will never be enough human analysts to analyze all the data in the databases.

1.2.4. Application Areas

Induction on databases, as indicated by Michie [1991], has found applications in many areas, including aerospace, instrumentation, manufacturing, pharmaceutical, electronics trouble-shooting, interpretation of biomedical monitoring, generating software from specifications, nuclear engineering, gas and oil processing, circuit fault diagnosis, steel and chemical process, seismic measurement interpretation, clinical diagnosis, credit control, and stock-market assessment.

Michalski [1983c] has reported that in diagnosing soybean disease domain, his inductive learning program generated rules that were the same or better than those supplied by human experts. Cheng and Fu [1985] applied inductive learning to the medicine domain to form intermediate concepts used in an expert system. Their experiment shows that inductive machine learning can generate knowledge that is more complete, more concise and more consistent. The PERSPICACE project [Cannat 1988] applies machine learning technique to the air traffic control, in order to build an expert knowledge base in radar conflict resolution. Shaw and others employed inductive learning in business loan evaluation, bond rating and bankruptcy prediction. They found that inductive learning has good predictive performance and produces informative decision models [Shaw et al. 1990]. Birdwell and others [1992] have applied inductive inference methods in the power electronics domain. They use decision trees for generating optimal real-time control signals. Their results have successfully shown that in engineering domains inductive learning techniques can also find important applications.

Other application domains are introduced by Parsaye and others [1989]. By using induction system on a database of car trouble reports, a car manufacturer found the reasons for wiring problems which people have never expected before. By using induction system, a computer manufacturer discovered the reasons for a repeated disk-drive problem. Inductive inference capabilities may also be applicable to crime databases, financial databases, and patient databases.

1.3. Database Environment

1.3.1. Introduction

Before we discuss the model and the methodology of conceptual induction on a data base, let us first formally define what is a database. A database is an organized collection of stored and operational data shared by several applications. A database can be logically described by its data model. A **data model** consists of two parts (1) a mathematical description of data, and (2) a set of operations defined in the mathematical framework to manipulate that data [Ullman 1988]. Traditionally, there are three major data models -- relational models, network models and hierarchical models. Recently, object-oriented models have been getting more attention.[4]

In this section, we first give brief formal descriptions of the four models of databases. The reader is referred to the appropriate literature for more details. Next, we will discuss the functions of a database system. Last we are going to describe the database model we employed and the interface to the database system upon which we developed the induction system.

1.3.2. Database Models

In the literature, different database models use different terminology. We try to keep the terms being used consistent in all models. Before we introduce each data model, let us define some terms that are used in all models. All databases store data logically in structured forms and allow a set of operations on the data. Data types specify the structures of data and the set of allowed values.

All databases accept values of **basic types** which we define as types such as integer, real, or character string that are commonly found in programming languages. Let $B = \{B_1, B_2, ..., B_m\}$ be the set of basic types. Each B_i, $i = 1, 2, ..., m$, has an associated **domain** D_i which is a set of elements defined by each specific database system. For example, the domain of the integer type can be a set of integers from -2^{16} to 2^{16}. Each element in D_i is called a **value** of type B_i. Let D

[4] There are other types of data models [Tsichritzis et al. 1982]. They are either not widely used or lack a set of operations on data. For example, the entity-relationship model [Chen 1976], which is a generalization of network and hierarchical models, is used only for conceptual scheme design before an application is mapped into a data model. So we are not going to cover those models.

$= D_1 \cup ... \cup D_m$ be the set of all the basic values. In addition to those basic types, some database models allow constructed data types with values composed of ordered sets of values. Any element of the value set may be a constructed data type's value, allowing nested structures.

Let U_a, the universe, be a set of elements A_1, A_2, ..., A_n called **attributes**, that is, $U_a = \{A_1, A_2, ..., A_n\}$. The scheme of a database is a set of data types, which defines the structure of a database.

1.3.2.1. The Relational Data Model

The relational model first introduced by E. Codd in 1970 [Codd 1970] has become the most widely studied and used model because of its simplicity and its sound theoretical foundation. In this subsection, we are going to answer the question: What is a database in the relational model?

The mathematical concept underlying the relational model is the theory of relations and first-order logic. Before we define the database, let us define some terms.

Definition: A **scheme** is a list of attribute-type pairs

$$S_j = ((A_{j1},B_{j1}), (A_{j2},B_{j2}), ..., (A_{jk},B_{jk}))$$

where $A_{ji} \in U_a$ and $B_{ji} \in B$, for i=1,2,...,k. We say that attribute A_{ji} is of type B_{ji}.

An ordered list of values is called a **tuple**, and the number of elements is called the **arity** of the tuple. For example, t = (e1,e2,e3) is a tuple, its arity is 3, and we call it a 3-tuple.

The domain of the scheme S_j is the **Cartesian product** of the domains D_{j1}, D_{j2}, ..., D_{jk}, associated with the attributes-type pairs, written as $CP(S_j) = D_{j1} \times D_{j2} \times ... \times D_{jk}$. This is the set of all k-tuples: $\{(v_1, v_2, ..., v_k) \mid v_i \text{ in } D_{ji}, i = 1, 2, ..., k\}$, and is the set of all possible tuples that may exist for the given scheme.

Definition: A **relation** R_j over a scheme S_j is a subset of $CP(S_j)$. S_j is called the scheme for relation R_j, and k is the arity of R_j. Each relation is a set of tuples.

Definition: A **formula** can be defined recursively as follows:

(1) $A_j \$ A_k$, $A_j \$ c$, and $c \$ A_j$ are formulas where A_j and A_k are attributes of the same basic type, $A_j \in U_a$, $A_k \in U_a$, $c \in D_j$, and $\$$ is an arithmetic comparison operator in $\{ =, \neq, >, <, \geq, \leq\}$

(2) If F1 and F2 are formulas, then the conjunction F1 $\wedge$ F2, the disjunction F1 $\vee$ F2, and the negations $\neg$F1 and $\neg$F2 are formula.

(3) Parentheses can be placed around formulas as necessary. The order of precedence is that arithmetic comparison operators are higher than logical operators. Among the logical operators, $\neg$ is the highest, $\wedge$ is lower, and $\vee$ is the lowest.

A formula can often be used to represent a relation which is derived from existing relations in the database. The derived relation is used to represent the user's current interests. A selection operation (to be discussed shortly) selects tuples, from existing relations, that satisfy the conditions specified by the formula.

Definition: A **database** DB is a set of relations R1, R2, ..., Rm; that is

$$DB = \{R1, R2, ..., Rm\}$$

such that Rj is a relation over scheme Sj, for each j = 1,2,...,m.

The set of schemes S = {S1, S2, ..., Sm} is called the scheme for the database DB. So a relation is a set of tuples, and each value's type in a tuple is specified by the relation's scheme. We can regard a relation as a table with the column headings as attributes, and each row corresponding to a tuple. The following is an example of a relation.

Example: In Table 1.1, we show a relation R with three attributes: COUNTRY, CAPITAL, POPULATION. The scheme of this relation is:

S = ((COUNTRY, String), (CAPITAL, String), (POPULATION, Integer)).

The scheme indicates the attributes used for referring values in a tuple and the types of the values. There are three tuples in this relation, and the arity is three. A database may consist of more than one relation. [End of Example]

We have introduced the mathematical notion of a relation; now let us introduce operations on the relational database. Operations on a database can be divided into two categories: (1)

Table 1.1. Example of a Relation.

COUNTRY	CAPITAL	POPULATION
Unite States	Washington D.C.	270
Japan	Tokyo	220
China	Beijing	1100

retrieval operations, and (2) modification operations.

For the retrieval operations, there are five basic operations that define the **relational algebra** [Ullman 1988]:

Let relation R1 have a scheme $((A_{11},B_{11}), ..., (A_{1k},B_{1k}))$; let relation R2 have a scheme $((A_{21},B_{21}), ..., (A_{2k},B_{2k}))$,

(1) **Union.** Let $B_{1i} = B_{2i}$ for all i=1,2,...,k. The union of the relations R1 and R2 is defined as

$$R1 \cup R2 = \{ t \mid t \in R1 \text{ or } t \in R2 \}$$

(2) **Set Difference.** Let $B_{1i} = B_{2i}$ for all i=1,2,...,k. The set difference of relation R1 and R2 is defined as

$$R1 - R2 = \{ t \mid t \in R1 \text{ and } t \notin R2 \}$$

(3) **Cartesian Product.** Let R1 and R2 be relations of arity k1 and k2, respectively. The Cartesian product of R1 and R2 is defined as

$$R1 \times R2 = \{(a_1,...,a_{k1}, b_1,...,b_{k2}) \mid (a_1,...,a_{k1}) \in R1 \text{ and } (b_1,...,b_{k2}) \in R2 \}$$

(4) **Projection.** Let R1 be a relation of arity k, The projection of R1 onto components $1 \le i1 < i2 < ... < im \le k$ is

$$P_{i1,i2,...,im}(R1) = \{(a_{i1},...,a_{im}) \mid (b_1, ..., b_k) \in R \text{ and } b_{ij} = a_{ij}, j = 1, 2,..., m \}$$

(5) **Selection.** Selection is an operation to select some tuples from a relation by a condition specified by a formula F. The selection of a relation R under a formula F is

$$SE_F(R) = \{ t \mid t \in R \text{ and } t \text{ satisfies } F \}$$

where t satisfies F means that the formula F' obtained by substituting each attribute A_k occurring in F by the A_k-value of t is evaluated to be true.

Example: Using the database R in the above example, then the selection $SE_{POPULATION<300}(R)$ would contain the two tuples corresponding the first two rows in Table 1.2. [End of Example]

There are some other operations that can be expressed in terms of the above basic operations.

(1) **Intersection:** Let R1 have a scheme $((A_{11},B_{11}), ..., (A_{1k},B_{1k}))$; let R2 have a scheme $((A_{21},B_{21}), ..., (A_{2k},B_{2k}))$, and let $B_{1i} = B_{2i}$ for all i=1,2,...,k. The intersection of relations R1 and R2 is defined as

$$R1 \cap R2 = R1 - (R1 - R2)$$

(2) **Quotient:** Let R1 have a scheme $((A_1,B_1), ..., (A_r,B_r), (A_{r+1},B_{r+1}), ..., (A_{r+s},B_{r+s}))$; and let R2 have a scheme $((A_{r+1},B_{r+1}), ..., (A_{r+s},B_{r+s}))$. The quotient of relations R1 and R2 is defined as

$$R1/R2 = P_{1,2,...,r}(R1) - P_{1,2,...,r}((P_{1,2,...,r}(R1) \times R2) - R1)$$

The quotient R1/R2 is the set of r-tuples $(a_1,..., a_r)$ such that for all s-tuples $(a_{r+1}, ..., a_{r+s})$ in R2, the tuple $(a_1, ..., a_r, a_{r+1}, ..., a_{r+s})$ is in R1.

Table 1.2. Result of Selection

COUNTRY	CAPITAL	POPULATION
Unite States	Washington D.C.	270
Japan	Tokyo	220

Example: Let R1 and R2 be the relations shown in Table 1.3(a) and (b), then R1/R2 is shown in Table 1.3(c). [End of Example]

(3) **Join:** The join of R1 and R2 on attributes A1 and A2 is defined as

$$J_{A1\$A2}(R1,R2) = SE_{A1\$A2}(R1 \times R2)$$

where $ is an arithmetic comparison operator (=, <, > and so on).

Example: Let R1 and R2 be the relations shown in Table 1.4(a) and (b), then the join $J_{A2<A4}(R1, R2)$ is shown in Table 1.4(c). [End of Example]

(4) **Natural Join:** Let R1 and R2 be two relations sharing common attributes A1, ..., Ar. Let R1.A1 denote the attribute A1 in R1 and R2.A2 denote the attribute A2 in R2. Then the natural join of R1 and R2 is defined as

$$J(R1,R2) = P_{i1,i2,\ldots,im}(SE_{(R1.A1=R2.A1) \wedge \ldots \wedge (R1.Ar=R2.Ar)}(R1 \times R2))$$

where i1, i2, ..., im is the list of all attributes of R1 X R2 except R2.A1, ..., R2.Ar

Example: Let R1 and R2 be relations shown in Table 1.5(a) and (b), then the natural join of them J(R1, R2) is shown in Table 1.5(c). [End of Example]

In order to make a data manipulation language more adequate for practical problems, some data modification operations are also defined as follows.

Table 1.3. Result of Quotient

(a) R1				(b) R2		(c) R1/R2	
A1	A2	A3	A4	A3	A4	A1	A2
a	b	1	2	1	2	a	b
a	b	2	3	3	4	c	d
a	b	3	4				
c	d	1	2				
c	d	2	3				
c	d	3	4				
e	f	1	2				

Table 1.4. Result of Join

(a) R1		(b) R2		(c) $J_{A2<A4}(R1,R2)$			
A1	A2	A3	A4	A1	A2	A3	A4
a	2	x	3	a	2	x	3
b	5	y	2	c	1	x	3
c	1			c	1	y	2

(1) **Insertion:** A tuple $t = (v_1, ..., v_k)$ can be inserted into a relation R with a scheme $((A_1,B_1), ..., (A_k,B_k))$ if v_i is of type B_i for i=1,...,k. The effect of insertion is that the contents of R is replaced by $R \cup \{ t \}$.

(2) **Deletion:** A tuple t in relation R can be deleted. The effect of deletion is that the contents of R is replaced by R - {t}.

(3) **Update:** The values of a tuple t in relation R can be changed. The effect of update operation is equivalent to deleting the t from R and inserting a new tuple with new values.

We have given a simplified description of a relational model. A full description would also cover some semantic constraints and structure constraints on representation of and operations upon the data. As described in [Yang 1986], a relational database model consists of three parts:

Table 1.5. Result of Natural Join

(a) R1		(b) R2		(3) J(R1, R2)		
A1	A2	A1	A3	A1	A2	A3
a	1	a	x	a	1	x
b	2	b	y	b	2	y
		a	z	a	1	z

(1) a set of relations R1, R2, ..., Rm with schemes S1, S2, ..., Sm where each S_j has a set of data dependencies[5] C_j associated to it, j = 1, 2, ..., m;

(2) the set of insert-update-delete rules[6]; and

(3) a set of operations at least as powerful as the relational algebra.

1.3.2.2. The Network Data Model

The network data model was first introduced in the 1971 report published by the Data Base Task Group (DBTG) of the Conference on Data Systems Languages [CODASYL 1971]. The network data model has more modeling capability than the relational data model because relationships between real world entities can be explicitly represented by links in the network model. The underlying mathematical concept of the network model is graph theory. However, the operations in the network model do not have the same elegance of formalism as the relational model.

Let the attributes A_j, j=1,...,n, and the basic types B_i, i=1,...,m, be defined as before.

Definition: The **record type** is defined recursively as follows:

(1) If t is a set of pairs of an attribute and a basic type, that is, $t = \{(A_1,B_1), ..., (A_k,B_k)\}$, then t is a record type;

(2) if $t_1, ..., t_p$ are record types, and $t = \{(A_1,t_1), ..., (A_p,t_p)\}$, then t is a record type.

The record type is a concept corresponding to the scheme in a relational data model, but it allows a more complex structure.

Definition: We define **value** recursively as follows:

---

[5] Data dependencies are constraints imposed on the structure of database and specify relationships among several attributes. Examples of data dependencies include functional dependencies, multivalued dependencies and join dependencies. Because of limited space. we do not cover details of these topics. Interested readers are referred to [Codd 1970, Yang 1986, Ullman 1988].

[6] The insert-update-delete rules are used to constrain modification operations and is given as follows:
(1) A primary key for a base relation cannot have a null value where a base relation is a relation defined independently from other relations in a database.
(2) If an attribute A1 of a compound primary key for a relation R1 is defined on the domain of a simple primary key A2 of a base relation R2, then each value of A1 in R1 must occur as a value of A2 in R2.

(1) If B_i is a basic type with a domain D_i, and $v \in D_i$, then v is a value of the basic type B_i;

(2) if the record type $t = \{(A_1,B_1), ..., (A_k,B_k)\}$, and $v_1 \in D_1$, ..., $v_k \in D_k$, then $v = \{(A_1,v_1), ..., (A_k,v_k)\}$ is a value of record type t.

(3) if the record type $t = \{(A_1,t_1), ..., (A_k,t_k)\}$, and v_i is a value of type t_i (either a basic type or a record type), for i=1,...,k, then $v = \{(A_1,v_1), ..., (A_k,v_k)\}$ is a value of record type t.

Let ID be a set of symbols called identifiers.

Definition: If a record type $t = \{(A_1,t_1), ..., (A_p,t_p)\}$, then a **record** r of type t is defined as

$$r = (id, \{(A_1,v_1), ..., (A_p,v_p)\})$$

where $id \in ID$ and v_i is a value of corresponding type t_i.

Definition: A **link** L is a pair of record types: L = (t1, t2). t1 is called the owner type, and t2 is called the member type.

Definition: Let L = (t1, t2), id be the identifier of a record of type t1, and let id_1, ..., id_s be identifiers of records of type t2. A **link occurrence** of L, denoted as l, is a pair

$$l = (id, \{id_1, ..., id_s\})$$

where id is called the owner and id_1, ..., id_s are called members.

A link is used to model a many-to-one relationship from the member type t2 to the owner type t1.

Definition: Let T be a set of record types, and let LS be a set of links over T. Then the pair SC = (T, LS) is called a **scheme.**

Definition: A **database** DB in the network data model is defined as a 3-tuple:

$$DB = (SC, R, S)$$

where SC = (T, LS) is a scheme, R is a set of records of types in T, and S is set of link occurrences of links in LS such that each identifier appearing in those link occurrences must identify a record in R.

The scheme SC = (T, LS) can be represented as a directed graph, called the scheme graph, where each logical record type in T is represented as a node, and each link in LS is represented as an arc from the owner type to the member type. Similarly, if we represent each record r by a vertex and each link occurrence l = (id, {id_1, ..., id_s}) by s directed arcs from id to each of id_1, ..., id_s, then the database DB can be represented as a directed graph called the database graph.

Example: Let a database scheme consists of three record types and two links:

T = {COURSE, STUDENT, ENROLL}

LS = {COURSE-ENROLL, STUDENT-ENROLL}

The scheme graph can be constructed as shown in Figure 1.1, and a possible database graph is also shown. [End of Example]

The operations on network databases are record-based which means that one record is processed at one time. In the network model links are regarded as two-way so that access to records can follow the link from owner to its member or from a member to its owner. Many operations take the current record as a default parameter which is the record most recently accessed. The current record of all types is called the current record of the run-unit. For each type, there is a current record, and for each link occurrence there is a current member record.

(1) **Find**. This operation locates a record and establishes it as the current record of the run-unit. It may follow the link to find the owner or a member of the current record. It has several forms:

FindFirstRecord$_F$(t) -- locates the first[7] record of type t that satisfies condition F, where F is a formula as defined before. F can be Nil if no condition is specified.

FindNextRecord$_F$(t) -- locate the next record of type t that satisfies condition F.

FindFirstMember$_F$(L) -- locates the first member record of a link occurrence whose owner is the current record of the run-unit.

[7] All records are stored in a random selected order, if the order is not important and all records are to be scanned, or in an order determined by some key attributes.

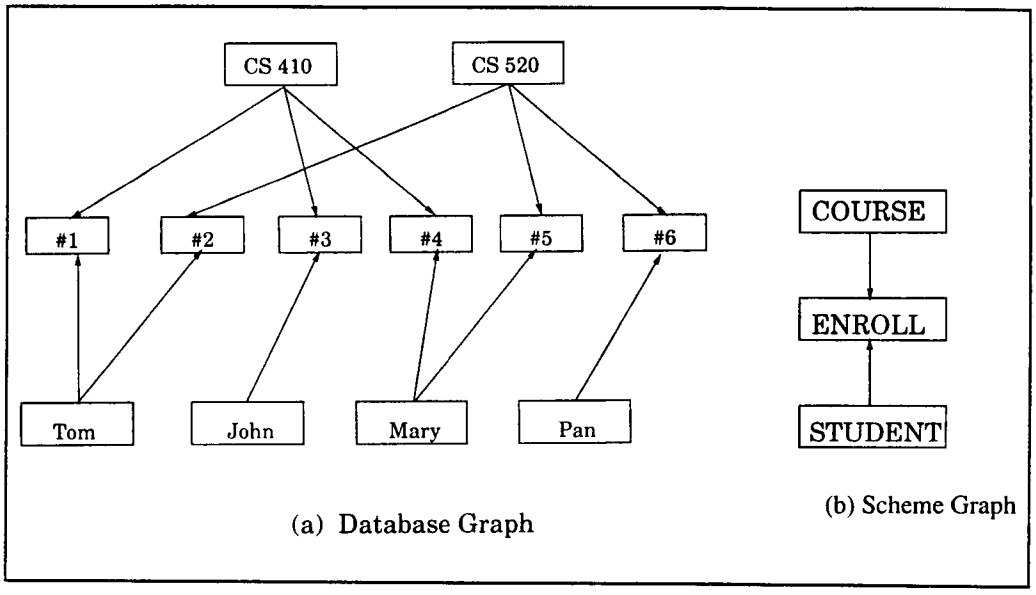

Figure 1.1. A Network Database

FindNextMember$_F$(L) -- locates the next member record.

FindOwner(L) -- locates the owner record of a link occurrence with a member record indicated by the current of run-unit.

FindRecord(id) -- locates a record with its identifier.

(2) **Get**. This operation retrieves specified attributes' values of the current record. This is similar to projection in the relational model but it can be performed only on one record.

Get$_{A1,...,Ak}$(t) -- where A1, ..., Ak are selected attributes, and t is a record type. If no attribute is specified then the whole record is retrieved.

In addition to those retrieval operations, modification operations are also included in the network database.

(1) **store**(r,t) -- stores a new record r of type t into the database.

(2) **insert**(r,L) -- inserts a record r as a member into the current link occurrence of link L.

(3) **remove**(L) -- removes the current record of the run-unit from the current link occurrence of type L.

(4) **modify$_E$** -- modifies the attributes of the current record of the run-unit, where E is a list of attribute-value pairs. The effect is to replace each attribute value by the specified new value.

(5) **delete** -- deletes the current record of the run-unit r from the database. If r is a member record of any link occurrences, it is removed from those link occurrences. If r is an owner record of a link occurrence, then their must be no member in this link occurrence; otherwise, an error occurs.

1.3.2.3. The Hierarchical Data Model

The hierarchical data model was first employed by the database system 2000 [DATAPRO 1972] and later by the successful database system IMS [IBM 1975]. Tsichritzis et al [1976] have given a survey of hierarchical database systems. The hierarchical data model is a network model with the restriction that the graph is a forest, or a set of trees. We can define the record type and the record as in the network model.

Definition: The **scheme,** denoted as S, is a tree with record types as nodes and links as arcs representing one-to-many relationships from parent to child.

Definition: A **database record,** denoted as R, of a scheme S is a tree with records as nodes, and each record is of the corresponding record type in the scheme. If type T is a node of the scheme S, and type Q is one of its children, then each record of type T in the database record has zero or more child records of type Q.

Definition: A **database** in the hierarchical data model is defined as a set of database records on a given set of schemes.

Example: Let T = {DEPT, OFFICE, FACULTY, STUDENT} be a set of record types and LS = {DEPT-OFFICE, DEPT-FACULTY, DEPT-STUDENT} be a set of links. Then a scheme SC = (T, LS) is represented by the tree in Figure 1.2 (a), and a possible database is also shown in Figure 1.2 (b). [End of Example]

Operations in the hierarchical data model are similar to those in the network model except that all links go only one way, from parent to child.

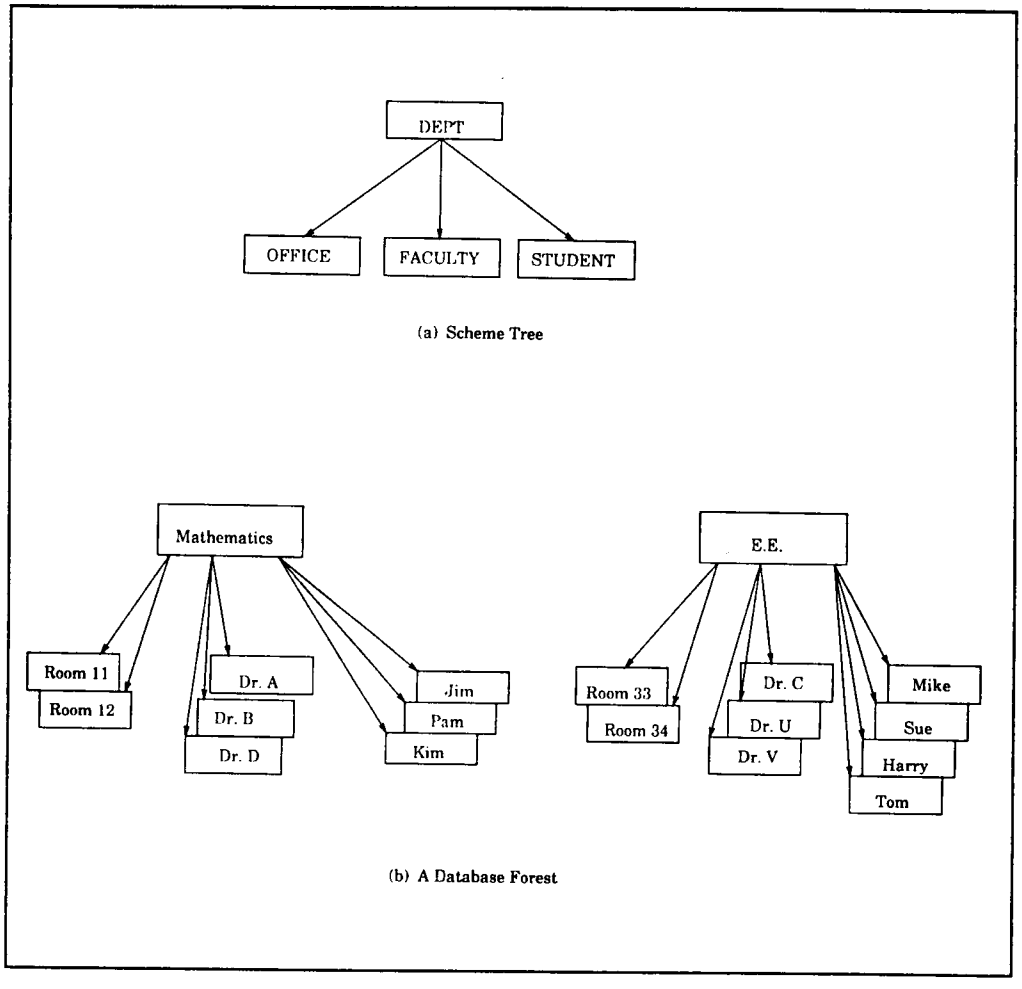

Figure 1.2. A Hierarchical Database

The selection operation selects one record from a database record based on conditions specified on the values of the record fields or values of its ancestor's fields. Selection can also be performed to scan the entire database for all records satisfying certain conditions or to scan all children of a node.

(1) **GetFirst**$_F$ -- retrieves the first record r from the database, where F specify conditions for the record r. The conditions include specification of values for selected attributes of r and r's ancestors in the database tree. If no condition is specified, then any record can be selected. The order of records is also implied by the system.

(2) **GetNext**$_F$ -- retrieves the next record that satisfies the condition F.

(3) **GetFirstChild**$_F$ --retrieves the first child record from the current parent that is the record last accessed by "GetFirst" or "GetNext".

Insertion, deletion and modification are similar to those of the network model. When inserting a record, we need to make its parent or ancestor the current record. When deleting a record, all its children will also be deleted from the database record.

(1) **insert**$_F$(r) -- inserts a record r into the database tree at the place specified by F, which specifies a path from the root record.

(2) **delete**$_F$ -- deletes a record r and all its descendants, where r is specified by F.

(3) **update**$_F$(AV) -- updates some attributes' values in a record r specified by F, where AV is a list indicating attributes and their new values.

Example: If we want to insert a new student record into the database given in the last example, the follow operation can be used:

$\text{insert}_{\text{DEPT.NAME=MATH, STUDENT}}$(ID#=283848239, NAME="Chen", Sex="M")

The resultant database record is shown in Figure 1.3. [End of Example]

1.3.2.4. The Object-Oriented Data Model

Object-oriented modeling was first developed in the programming languages Simula67 [Dahl et al 1968] and Smalltalk [Goldberg et al 1983]. In recent years, database researchers have

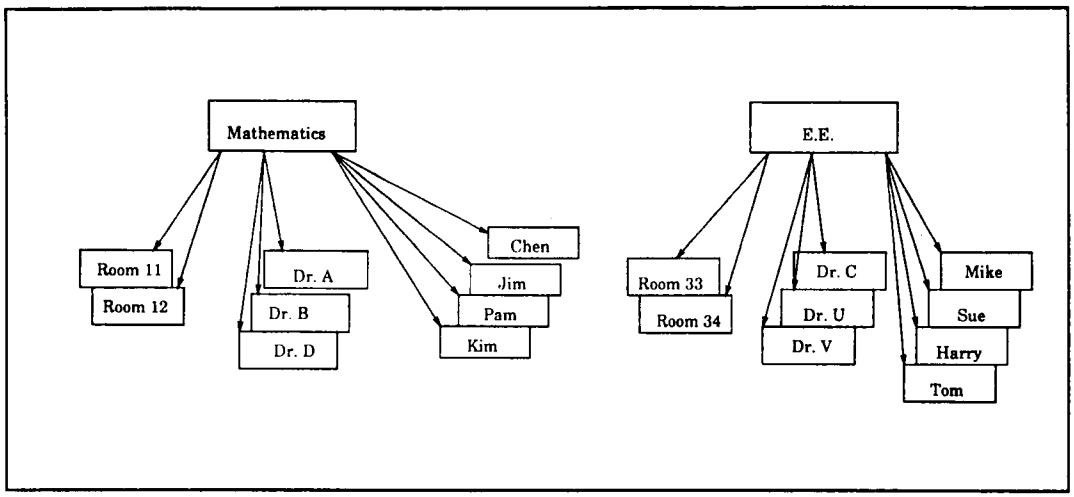

Figure 1.3. Database after Insertion

adopted an object-oriented modeling approach in database systems [Andrews et al 1987, Copeland et al 1984]. However, a strong theoretical framework for object-oriented systems is not mature. Efforts have been made in this direction by some researchers [Lecluse et al 1990]. An object-oriented database has the following major features as indicated by Ullman [1988]:

(1) data types can be complex with nested structure,

(2) procedures and data can be encapsulated to an object, and

(3) objects are identified by their unique identifiers, not by their stored data value.

Here we give a simplified model of an object-oriented database; a more comprehensive model can be found in [Lecluse 1990].

Let ID be a set of symbols called **identifiers**, and let MT be a set of symbols called **methods**. Let attributes A_i, i=1,...,n, basic types B_i and their domains D_i, i=1,...,m, be defined as before.

Definition: The **object type** is defined recursively as:

(1) If t is a basic type, and $M \subseteq MT$, then (t,M) is an object type.

(2) If t is an object type, and $M \subseteq MT$, then (SETOF(t), M) is an object type, where SETOF(t) represents a finite number of copies of t.

(3) If t_1, ..., t_p are object types, A_1, ..., A_p are attributes in U_a, and $M \subseteq MT$, then $(\{(A_1,t_1), ..., (A_p,t_p)\}, M)$ is an object type.

Type (1) is called a **basic object type,** and types (2) and (3) are called **constructed object types.** An object type is a pair of two elements; the first element represents the data structure and the second element represent the procedures which may be applied to the object.

Definition: The **value** v of an object type t is defined as:

(1) If $t = (B_i,M)$ and $v \in D_i$, then v is a value of type t, called a **basic value.**

(2) If t = (SETOF(t'), M), and v is a subset of ID, then v is a value of type t, called a **set value.**

(3) If $t = (\{(A_1,t_1), ..., (A_p,t_p)\}, M)$, and $v = \{(A_1,v_1), ..., (A_p,v_p)\}$, then v is a value of type t, called a **tuple value,** where $A_j \in U_a$, and $v_j \in D_i$ if $t_j = B_i$ or $v_j \in ID$ if t_j is a constructed object type, for all j = 1, ..., p.

We denote the set of all values by V.

Definition: An **object** of type t is a tuple o = (i, v, M), where i is an identifier, an element of ID, $v \in V$ is a value of type t, and $M \subseteq MT$ is a set of methods. o is a **basic object** if v is a basic value, a **set-structured object** if v is a set value, or a **tuple-structured object** if v is a tuple value. Each method in M is associated with an operation to be performed on the objects.

If o = (i, v, M) is an object, **id**(o) denotes the identifier i, and **value**(o) denotes the value v. Let $OBJ = \{o_1, ..., o_p\}$ be a set of objects, and let **ref** be the function from OBJ to 2^{ID} which associates to an object the set of all the identifiers appearing in its value.

Definition: A set $OBJ = \{o_1, ..., o_p\}$ of objects is **consistent** if and only if:

(1) the **id** function is injective on OBJ (that is, there is no pair of objects with the same identifier);

(2) for all i=1,...,p and $o_i \in OBJ$, $\mathbf{ref}(o_i) \subseteq \mathbf{id}(OBJ) = \{\mathbf{id}(o_1), ..., \mathbf{id}(o_p)\}$ (that is, every referenced identifier corresponds to an object in OBJ).

Let CT = {T1, ..., Tp} be a set of constructed object types which have been defined. If $T \in CT$, then we denote the set of all constructed object types appearing in T by **refer**(T).

Definition: The set CT of constructed object types is a **scheme** if and only if

(1) For any two different object types T1 and T2 in CT, if T1 = (t1,M1) and T2 = (t2,M2), then t1≠t2. That is, different constructed object types have different data structures.

(2) For all t in CT, **refer**(t) ⊆ CT. That is, every referred constructed object types must be included in the scheme.

In above definition of scheme, circular references are allowed and they are useful in many occasions. For example, we may need to define an object type PERSON to be:

PERSON = ({(Name, String),(Addr, String),(Father, PERSON)}, Mp)

In this constructed type, the attribute Father is defined as the same type PERSON.

An object set OBJ over an object type t is a subset of the Cartesian product

ID X V(t) X {M},

where V(t) is the set of all the possible values of t and M is a set of methods of t.

Definition: Let S be a scheme, and DB = OBJ1 ∪ ... ∪ OBJq, where for each i OBJi is a set of objects over type Ti, and Ti is either a basic object type or Ti ∈ S. DB is a **database** if and only if DB is a consistent set of objects.

Example: Let S = {Person, Addr, Pset} be a set of constructed object types, where

Person = ({(Name,String), (Address,Addr), (Age,Integer), (Children,Pset)}, Mp)

Addr = ({(Street,String), (City,String), (State,String), (Zipcode,String)}, Ma)

Pset = (SETOF(Person), Ms)

Then S is a scheme because

refer(Person)∪**refer**(Addr)∪**refer**(Pset) = {Addr, Pset} ⊆ S.

Let

o1 = (i1, {(Name,"Tom Lee"), (Address,i4), (Age,35), (Children,i5)}, Mp)

o2 = (i2, {(Name,"Sue Lee"), (Address,i4), (Age,3), (Children,Nil)}, Mp)

o3 = (i3, {(Name,"Jim Lee"), (Address,i4), (Age,1), (Children,Nil)}, Mp)

o4 = (i4, {(Street,"120 8th St."), (City,"Wayne"), (State,"TN"), (Zipcode,"37888")}, Ma)

o5 = (i5, {i2, i3}, Ms)

Then DB = {o1, o2, o3, o4, o5} is a database because (1) $\mathbf{id}(o_i) \neq \mathbf{id}(o_j)$ for all i,j=1,...,5 and i≠j; (2) $\mathbf{ref}(o_i) \subseteq \mathbf{id}(DB)$ for all i=1,...,5.

[End of Example]

Operations in the object-oriented model are mainly performed on one object at a time. Object identifiers are, in fact, pointers which uniquely specify objects in a database. Navigation from an object b to the objects pointed to by an attribute value of b is straightforward by reference through object identifiers. Selection of an object from an object set based on a certain condition is also provided. Let T be all objects of a type, or a set of objects specified by an object attribute.

(1) **ForEach**(T): will retrieve each object in T one at a time in a loop until all the objects of T are retrieved.

(2) $\mathbf{ForEach}_F$(T): Similar to above operation, but each time only the object satisfying the condition specified by F is retrieved, where F is logical formula defined in the same way as in the relational model, but here the comparison operators are expanded to include set comparators (equal, intersect, contain).

(3) **First**(T): will retrieve the first[8] object of T.

(4) $\mathbf{First}_F$(T): will retrieve the first object that satisfies the condition specified by F, where F is the same as above.

(5) **Next**(T): will retrieve the next object in T.

(6) $\mathbf{Next}_F$(T): will retrieve the next object in T satisfying the condition F.

Example: Using the database in the above example, the operation

ID = $\text{First}_{\text{Age}>30}$(Person)

will set variable ID to i1 for the object representing the person Tom Lee, and get o1 from the database. The operation

[8] There is a storage order for objects in a database; please refer to the remarks on the record order in the network data model.

Child = First(ID.Children)

may set variable Child to i4 and get o4 from the database. [End of Example]

Modification operations are also provided in the object-oriented model.

(1) **Create**$_I$(T): creates an object of type T, where I is a set of initial values for the object's attributes.

(2) **Delete**(id): deletes an object specified by the identifier id from the database.

(3) **Insert**$_P$(id): assigns values to attributes of an object, where P is a list of attribute-value pairs.

In the object-oriented database, if we want to delete an object, we have to specify the object by its identifier. This is different from a relational database where tuples are specified by their attribute values. In addition to the above operations, the object-oriented model provides facilities to define operations on objects by the user. With methods encapsulated in the object type definitions, many operations can be performed on the objects of those types.

1.3.3. Functions of a Database System

A database system (also called DBMS -- database management system) is a group of computer programs to record and maintain persistent data in a permanently existing database. The DBMS makes the access of data much easier and more efficient than a file system. The DBMS is an important tool for people who manage and use a large amount of data.

The DBMS has several major functions [Everest 1986] which can be listed as follows:

(1) Database establishment:

- i) database definition
- ii) database creation
- iii) database revision

(2) Database use:

- i) retrieval
- ii) update
- iii) programming user facilities

(3) Underlying Support

i) integrity control

ii) access control

The DBMS provides a data model which is a logical abstraction of data so that a user can design a database in a useful form. **Database definition functions** allow a user to specify the database schemes, including attributes, relations or record types, relationships among attributes or record types, and indices. **Database creation functions** allow the user to construct databases conform to schemes defined by the database definition functions. **Database revision functions** allow the user to modify a database's scheme. The DBMS provides a data description language to the user for database definition, creation, or revision.

Retrieval functions allow the user to obtain specified data from the database. **Update functions** allow the user to delete data from, insert data into, or modify data in the database. The DBMS supports a query or data manipulation language for data retrieval or updating. **Programming user facilities** provide an interface between data description-manipulation languages and a programming language, so that database operations can be carried out by application programs.

Integrity control functions ensure that data stored in a database is consistent with specified and implied constraints on the data, so that invalid data may be deleted or avoided. **Access control functions** assign different access rights to different users and prevent unauthorized access to the database.

1.3.4. The Generalization Data Model

Our strategy is to build a conceptual induction system as a layer above the database system. In order to integrate the two systems, we need a database model to support the semantics of induction. Semantics of induction are abstraction relationships among concepts and their instances and include

(1) transformation,

(2) aggregation, and

(3) generalization.

Entities are described by their attributes with certain values. Those attribute-value pairs are called features. **Transformation** is a form of relationship which relates an object with a low-level description to an object with a high-level description. A low-level description is a description that contains detailed and specific features about an entity, that are farther from human understanding. For example, description of a circle by a list of the coordinates of all the pixels on the circle is not easy for a human to comprehend. A higher-level description is a description that is closer to human understanding. For example, the description of a circle with the coordinates of its center and the radius is a description on a higher level. **Aggregation** is a form of relationship which relates an object with a set of features to an object with summarized or collective features, or a relationship that is established between a set of objects and an object with features characterizing the set of objects as a whole. For example, the description:

$$[\text{color(part1)} = \text{red}] \wedge [\text{color(part2)} = \text{red}] \wedge$$
$$[\text{color(part3)} = \text{red}] \wedge [\text{num-of-parts} = 3]$$

can be aggregated into the description:

$$[\text{color-of-all-parts} = \text{red}] \wedge [\text{num-of-parts} = 3]$$

Generalization is an relationship which relates a set of objects to another type of object that has a description which emphasizes the similarities and ignores the differences among the set of objects. For example, the descriptions of two objects:

$$d1 = [\text{size(part1)} = \text{large}] \wedge [\text{color(part1)} = \text{blue}]$$

$$d2 = [\text{size(part1)} = \text{large}] \wedge [\text{color(part1)} = \text{red}]$$

can be generalized into the description:

$$D = [\text{size(part1)} = \text{large}] \wedge [\text{color(part1)} = \text{blue} \vee \text{red}]$$

In this subsection, we address the question: Which data model can best support the induction semantics?

We have developed a data model called the Generalization Data Model (GDM) which supports inductive semantics. The GDM is based on an object-oriented data model [Meyer 1988;

Unland et al 1990, Lecluse et al 1990, Heiler et al. 1987]. The reason is that traditional DBMSs of the hierarchical data model, the network data model or the relational data model have limitations in their data types and modeling power: (1) A mismatch exists between their constructs[9] and real world entities. For example, in a relational data model, relations need to be normalized[10], while the normal forms do not correspond to concepts in the real world [Smith 1989]. (2) The inability to directly model relationships between concepts. For example, in a relational data model, relationships of concepts are implied by foreign keys[11]. (3) The requirement of lower level operations that are performed on low-level form of data. For example, in a relational data model, the modeling of relationships is performed through low-level operations, such as relational joins and projections [Ullman 1988]. Some essential disadvantages of traditional database systems have been identified by Unland [1990]. In recent years many knowledge-based systems have been developed for complex applications which do not map well into any of the three data models. These applications require various data types with complex relationships, inferential processing, and frequent interaction. The object-oriented model has several advantages over traditional data models. It reduces the semantic gap[12] between complex applications and the data storage supporting those applications; it supports modeling of complex entities and their relationships, and it provides flexible abstract data-typing facilities[13]. Entities correspond to objects in the model; relationships between entities correspond to links among

---

[9] Constructs are basic building blocks in a database for modeling entities and their relationships. For example, in a relational data model constructs are relations.

[10] Normalization of relations is reconstruction of relations based on data dependencies so that the resulting relations will avoid certain operational abnormalities. Detailed discussion on relation normalization is given in [Ullman 1982].

[11] An attribute of a relation is a foreign key if it is not the key of this relation, but its values are values of the key of some other relation. Refer to [Date 1977] for details.

[12] A semantic gap is the dissimilarity between a modeling tool and the portion of the real world to be modeled.

[13] The inheritance mechanism allows new object types to be easily defined in terms of the properties and structure of existing types.

objects. High-level semantics, such as abstraction relationships and concept constraints, can be captured by the object-oriented data model. An object-oriented data model can enhances the inductive inference process with its following main features: (1) data and operation abstraction and encapsulation, (2) property inheritance, and (3) easy communication with users.

The GDM utilizes an interface to an underlying object-oriented database. The interface utilizes the following object-oriented database functions.

(1) Complex object definition -- object types of concepts, clusters, and concept instances at different level of abstraction can be defined by the database entity-definition function.

(2) Abstraction relationship modeling -- abstraction relationships between objects can be modeled by link between objects.

(3) Semantics incorporation -- Induction semantics are incorporated into the object types using methods which encapsulate procedures into an object type.

(4) Data retrieval -- Retrieval functions are provided to access all objects of a type or all objects in a set field of an object. Selective access of objects can be performed based on specified logical conditions.

(5) Database manipulation -- Insertion, deletion and modification functions can be applied to objects.

The detailed description of the GDM is given in chapter 3.

1.4. The Approach to the Research Goal

In this section we describe the approach to the research goal laid out in section 1.1.

We analyze existing induction methods and identify required properties of induction in a database environment. Then, we develop a new model of conceptual induction called the KNOWD (Knowledge-Directed) model, which combines efficient incremental induction with an expressive representation capability. To support this model, a representation language is designed to represent concept instances and concept descriptions. With this language, structural concepts,

disjunctions, and relationships can be described. Different database models are reviewed, and a new model, called GDM (Generalization Data Model), is developed upon which the KNOWD induction model is built. GDM incorporates an object-oriented data model with inductive semantics.

While an induction model specifies the conditions to be followed in the inductive process of concept formation, an induction methodology specifies the actual process of concept formation. Based on the KNOWD model we also develop an inductive methodology, called the KNOWD (Knowledge-Directed) methodology. In effect, the methodology employs the model with inductive heuristics and domain specific knowledge to guide the inductive process. This methodology follows an incremental strategy and consists of multiple levels of abstractions. The inductive process is divided into two phases: basic abstraction and advanced abstraction. Basic abstraction is a type of supervised conceptual induction,[14] and advanced abstraction is a type of unsupervised conceptual induction.[15] As an integral part, the KNOWD methodology explicitly contains an inductive bias management system to manage various inductive heuristics, and a deductive inference mechanism to support constructive induction and to constrain the inductive search.

The model and methodology we developed are not merely theoretical outcomes but can also be applied to solve real world problems. We implement an inductive database management system, IDB1,[16] based on the KNOWD model and methodology. We apply IDB1 to three application domains: space shuttle main engine test analysis, chemical process data analysis, and generalization of the solutions to discrete control problems which arise in power electronics applications. Each application area provides a slightly different mechanism for evaluation of

---

[14] Supervised conceptual induction is conceptual induction in which all the input instances are classified into classes by an external teacher (a human or a program).

[15] Unsupervised conceptual induction is conceptual induction in which classes of input instances are not known in advance, and the classification is performed by the inductive system.

[16] IDB1 was implemented in the Symbolics Common Lisp Environment on a Symbolics 3670 machine, and ported to the Sun Common Lisp Development Environment version 4.0.1 on a Sun SPARCstation 1+.

conceptual induction in IDB1. The performance of IDB1 is analyzed through various experiments and theoretical analysis.

The most important properties of the KNOWD model are:

(1) The model represents descriptions of concepts by sets of logical expressions.

(2) The model is able to represent relationships of concept features directed by domain knowledge.

(3) The representation language supports the combination of expressive descriptions and incremental induction.

The most important properties of the KNOWD methodology are:

(1) The methodology consists of multilevel abstraction.

(2) It performs incremental induction for generating expressive descriptions.

(3) It utilizes various knowledge in a data rich environment.

(4) It explicitly contains a comprehensive bias management system.

1.5. A Guide to the Chapters

In this chapter, we have described what we are going to do, laid out the goal of our research and explained why it is important. We have also presented formal descriptions of database models, and described our approach to the research work.

In chapter 2, we are going to give a general mathematical framework for conceptual induction, and overview of what others have done in this area. Then, we will specialize the general induction model into two models for describing existing methods, and discuss their advantages and disadvantages. Last, we discuss the required properties of a conceptual induction system in a database environment.

In chapter 3, we will present the results of this research work: the knowledge-directed model and methodology of conceptual induction. First, we will specialize the general model developed in chapter 2 into our induction model. Then, we will give a formal description of the KNOWD methodology, detailed description of the algorithms, and their properties.

In chapter 4, we will discuss three applications of our induction methodology and various experiments performed on our induction system. Then we will give an evaluation of our implementation of the methodology on three aspects: representation capability, conceptual induction quality, and computational efficiency.

In chapter 5, we will summarize the research work and its importance, describe the contributions and important features of our approach, and comment on further research work and development.

Appendix A provides a description of the deductive mechanism for supporting knowledge-directed induction. Appendix B describes the bias management system BMS, which is a component of the KNOWD methodology. Important functions of a bias management are described; various inductive biases and their effects on induction are discussed; and the bias management strategy and organization are presented.

CHAPTER 2

BACKGROUND

In this chapter, we formally describe conceptual induction both in a general framework and using specific models, give an overview of existing methods of other researchers and specify extensions needed in a database environment. In section 2.1, we characterize conceptual induction in a general mathematical framework. In section 2.2, we overview existing methods of conceptual induction and address their pros and cons. In section 2.3, we present two formal models for describing existing induction methods and discuss their advantages and disadvantages. In section 2.4, we discuss the required extensions and properties of a conceptual induction system in a database environment.

2.1. A Framework for Conceptual Induction

In this section, we characterize the conceptual induction problem, whereby we answer the question: What is conceptual induction? Next, we give a mathematical framework of conceptual induction to set out its theoretical foundation in logic.

2.1.1. Characterization of the Conceptual Induction Problem

Conceptual induction, also called **conceptual inductive learning**, is a type of machine learning defined by Michalski [1983b]. Its final products are symbolic descriptions of concepts and their relationships expressed in high-level, human-oriented terms and forms so that they are easily understood[1] by a human being. The descriptions are typically about real world objects or situations, and the most relevant applications of conceptual induction are conceptual data analysis

[1] The comprehensibility of the resulting descriptions (detailed in [Michalski 83b]) requires that the descriptions of given classes of entities be semantically and structurally similar to those that a human expert may produce after observing the same entities, and that components of these descriptions should be comprehensible as single "chunks" of information directly interpretable in natural language.

and knowledge acquisition for expert systems. The characterization of the conceptual induction problems which our inductive database system tries to solve is given in this subsection.

Conceptual induction includes two forms of induction: supervised and unsupervised induction. **Supervised induction**, also called **concept acquisition** or **concept learning from examples**, induces general descriptions of concepts from specific instances of these concepts. The supervised induction problem is characterized in Figure 2.1. An instance can be represented by a conjunction of predicates. We define a concept C_i as a set of instances from a given universe of instances (formal definitions will be given in subsection 2.1.2). Domain knowledge **K** is represented in production rules which can deduce new features, build relationships among features, and constrain the association of features and their relationships with a concept. Preference criteria **P** are conditions which characterize the desirable properties of the possible concept descriptions. For example, preference criteria on the syntax of concept descriptions impose a preference for simpler forms of descriptions. Preference criteria on statistics impose a preference for concept descriptions that are satisfied by more instances of this concept and satisfied by fewer instances of the other concepts. The purpose of supervised induction is to induce a description D_i for each concept C_i which is a generalization of the given instances of the concept, and can best characterize the given instances of that concept by the preference criteria **P**. When we say that concept descriptions are induced, we mean that the instances in I_i are not logical consequences of the knowledge **K**. When we say that concept descriptions are generalizations of the instances, we mean that instances in I_i are logical consequences of the concept description D_i and knowledge **K**. For example, let **K** contain rules:

rule1: IF x is a triangle,
THEN x is a polygon

rule2: IF x is a rectangle,
THEN x is a polygon

let I_i contain two instances:

d1 = [shape(a) = triangle] $\wedge$ [color(a) = red]

GIVEN:

* A set of **Instances,** I_i, of a concept C_i, for all $i \in \{1, ..., u\}$, where each instance represents a specific fact about the concept.
* **Domain knowledge** (optional), **K**, which is a set of rules that generate new features and relationships of features, and which also impose constraints on the possible concept descriptions.
* **Preference criteria, P**, which is a set of conditions that characterize the desirable syntactic and statistical properties of the possible concept descriptions.

DETERMINE:

* **Descriptions,** D_i, of the concept C_i, for all $i \in \{1, ..., u\}$, which characterize its instances in I_i and satisfy the domain knowledge **K** and the preference criteria **P**.

Figure 2.1. The Supervised Conceptual Indution Problem

$d2 = [shape(a) = rectangle] \wedge [color(a) = red]$

and the concept description be:

$D_i = [shape(a) = polygon] \wedge [color(a) = red]$

Obviously, neither d1 nor d2 is a logical consequence of the rules in **K**. That is, we cannot deduce from **K** that if an instance satisfies the description of d1 or d2, then this instance belongs to C_i. But with the premises **K** and the statement that any instance satisfying D_i belongs to C_i, we can deduce that if an instance satisfies d1 or d2, then it belongs to C_i.

Unsupervised induction, also called **descriptive generalization** or **conceptual learning from observation**, establishes new concepts and their descriptions for characterizing unclassified instances. The unsupervised induction problem is characterized in Figure 2.2. The concept hierarchy **CH** is a tree structure with internal nodes representing concepts and leaf nodes representing instances. Instances with similar features are grouped together to form a concept. Similar concepts are grouped together to form higher-level concepts. The description of a concept

characterizes all instances under it. The clustering criteria **P** are based on similarities of the new instances in **I** with concepts in **CH**, the discrimination capabilities of concepts, and the qualities of resulting concept descriptions. The incorporation process consists of the classification of the new instance into the concept hierarchy and the modification of descriptions of parent concepts. We characterize unsupervised induction in an incremental way, where a new concept hierarchy **CH'** is based on a previous concept hierarchy **CH**.[2] The task of unsupervised induction is to determine the best clustering of given instances and the best concept descriptions for the clusters. In chapter 3, we will give detailed description of the algorithms of both supervised and unsupervised induction.

2.1.2 The General Model of Conceptual Induction

We define a concept as a set of instances. A formal definition of concepts will be given shortly. An instance has different definitions in different inductive models and can represent any specific physical object, such as a plant, an animal or a machine, or any specific situation, such as a disease or a machine fault. Although there are many conceptual inductive systems with different criteria to follow in generating, selecting or modifying concept descriptions, some basic induction criteria exist in most methods. Two such basic inductive criteria are identified:

(1) A concept description should closely characterize as many instances of the concept as possible.

(2) A concept description should cover as few instances of the other concepts as possible.

Different inductive models may have different interpretations of the basic criteria. Based on the representation of concept descriptions and how the above criteria are enforced, we classify existing conceptual inductive methods into two groups: **set-based models** and **logic-based models**, which will be discussed in the following subsections. Both models have difficulties in providing the desired conceptual induction in a database environment. We developed a new model to overcome

[2] For non-incremental induction **CH** is null, and **I** contains all the instances for induction. Some induction systems that do not generate a concept hierarchy can be regarded as creating a concept hierarchy of one level.

GIVEN:

* A set of **instances I** of concepts.
* A **concept hierarchy, CH**, which is a tree (initially null) that classifies concept instances into clusters representing concepts. Each concept has a description.
* **Domain knowledge, K**, which is a set of rules that generate new features and relationships of features, and also impose constraints on the possible concept descriptions.
* **Clustering criteria, P**, which is a set of conditions which characterize the desirable classification properties of the possible concept hierarchy.

DETERMINE:

* A new **concept hierarchy, CH'**, which incorporates the new instances in I to extend the existing concepts or to form new concepts that satisfy the domain knowledge and the criteria of clustering.

Figure 2.2. The Unsupervised Conceptual Induction Problem

those difficulties. In this subsection we give a formal description of a general conceptual induction model.

First, we define some symbols and terms [3] which are shared by all models. Table 2.1 provides a list of basic symbols used and their meanings.

$U = \{ K_1, K_2, ..., K_u \}$ is the set of constant symbols representing names of all the possible concepts in the universe of discourse. U is either given by input or determined by a classification component of an inductive system. [4]

Let E be a set of all possible symbolic descriptions defined by a specific inductive model. Different models may have different definitions and formats for the descriptions, and we will

[3] In order to avoid duplication we define "instance" and "concept" here, but precise definitions can be built only after other definitions are established in each specific model.

[4] In supervised conceptual induction, U is given as input, and instances have already been classified into classes corresponding to concepts. In unsupervised conceptual induction, the classification of input instances is not given. The classification component of the inductive system will classify the instances into classes based on the common features of the instances and other criteria.

Table 2.1. A Table of Basic Symbols

Symbol	Explanation
$\neg$	negation
$\wedge$	conjunction
$\vee$	disjunction
$(\exists x)$	existential quantifier over x
$(\forall x)$	universal quantifier over x
$\rightarrow$	implication
$\Rightarrow$	generalization
U	a set of all concept names in the universe of discourse
K_i	a concept name
E	a set of all possible symbolic descriptions defined by a specific induction model
d	a description of an instance
C_i	a concept (a set of instances)
S	a set of all possible instances in the universe of discourse
I_i	a set of known instances of a concept C_i
P	a set of given predicates
A	a set of given constant terms
X	a set of variables
p_i	a predicate
t_i	a term
D_i	a concept description
F	a set of all possible features in the universe of discourse
$CONJ_i$	a conjunction of atomic expressions
EXP_i	a logical expression

define them and give examples of them in later sections for each specific model.

Definition: An **instance,** d, is a symbolic description of an object in the universe of discourse.

We define S as the set of all possible instances in the universe of discourse. Let I = { d_1, ..., d_m } be a set of known (or observed) instance which will be used as input to an conceptual induction system. Then we have $I \subseteq S$ and $S \subseteq E$.

Let ρ: $I \rightarrow U$ be a classification function from I into U, so ρ can be regarded as a set of ordered pairs (d,k), where $d \in I$ and $k \in U$. Let λ: $S \rightarrow U$ be a classification function from S into U. λ can be regarded as a set of ordered pairs (d,k) such that $\rho \subseteq \lambda$.[5]

We define a relation $\equiv_\rho$ on I such that for any d_1 and d_2 in I,

$d_1 \equiv_\rho d_2$ if and only if $\rho(d_1) = \rho(d_2)$;

and also define a relation $\equiv_\lambda$ on S such that for any d_1 and d_2 in S,

$d_1 \equiv_\lambda d_2$ if and only if $\lambda(d_1) = \lambda(d_2)$

Theorem: Relations $\equiv_\rho$ and $\equiv_\lambda$ defined above are equivalence relations.

Proof: Let us prove the relation $\equiv_\lambda$ is an equivalence relation. The relation $\equiv_\rho$ can proved similarly.

(1) For all d in S, since $\rho(d) = \rho(d)$, we have $d \equiv_\lambda d$. So the relation is reflexive.

(2) For all d_1 and d_2 in S, if $d_1 \equiv_\lambda d_2$ then $\rho(d_1) = \rho(d_2)$, and $\rho(d_2) = \rho(d_1)$, so $d_2 \equiv_\lambda d_1$. Therefore, the relation is symmetric.

(3) For all d_1, d_2 and d_3 in S, if $d_1 \equiv_\lambda d_2$ and $d_2 \equiv_\lambda d_3$, then $\rho(d_1) = \rho(d_2)$ and $\rho(d_2) = \rho(d_3)$; so $\rho(d_1) = \rho(d_3)$ which implies $d_1 \equiv_\lambda d_3$. Therefore, the relation is transitive.

[End of Proof]

The distinct equivalence classes of the equivalence relation $\equiv_\rho$ on I provide a decomposition of I as a union of mutually disjoint subsets. Let I_i = { d | $d \in I$ and $\rho(d) = K_i$ }.

[5] λ is not determined before the induction but is determined by the output of the induction.

Then $I_i \subseteq C_i$ and is called a set of known (or observed) instances.[6] The distinct equivalence classes of the equivalence relation $\equiv_\lambda$ on S provide a decomposition of S as a union of mutually disjoint subsets.

Definition: A **concept** is an equivalence class of the equivalence relation $\equiv_\lambda$ on S. For a given universe of discourse, we can define u concepts as:

$$C_i = \{ d \mid d \in S \text{ and } \lambda(d) = K_i \}, i = 1, ..., u$$

That is, C_i is the set of instances that are mapped to the concept name K_i, and we have $S = \bigcup_{i=1}^{u} C_i$.[7] If $i \neq j$, then we have $K_i \neq K_j$ and $C_i \cap C_j = \varnothing$.

Usually, concepts are not known to an induction system because a concept may contain a large number of instances of which only a part is available to the induction system. Let a **concept description** be an element of E that satisfies some preference criteria. Preference criteria are specified by the criterion function defined below. Let $D \subseteq E$ be a set of concept descriptions, and let $D_i \in D$ be a description of concept C_i, i=1, ..., u. D_i is generated by conceptual induction.

Let the function H be defined as $H: I \times E \rightarrow$ {**true, false, uncertain**}. We call H the **criterion function,** which is used to specify the desired properties of the concept descriptions. We use $H^{-1}(b)$ to represent the inverse image of b under H. H will have different definitions in different specific models.

In the induction process, the decision on how to generate, select, or modify the description D_i of a concept C_i is based on the two basic criteria which we state here as two

---

[6] In case of supervised induction, I_i is the input given by an outside teacher which is a program or a human, and I_i is called the set of known instances. While in unsupervised induction, I_i is determined by a classification component of the system from a set of descriptions of instances which are given without indicating concept names. The system assigns a symbol as the concept name. In this case I_i is called the set of observed instances. From the above definitions, we know that $I = \bigcup_{i=1}^{u} I_i$ and for any $i \neq j$, I_i and I_j are disjoint.

[7] The statement above is a general definition of a concept, which does not endorse any meaning to a concept, except that it specifies a grouping of instances. To find the meaning or semantic description of a concept based on those known instances is the task of conceptual induction.

conditions that a description D_i should satisfy. The two conditions are completeness and consistency[8].

Definition: The **completeness condition** in the general model is a basic requirement for a concept description D_i, i=1, ..., u, which is expressed as:

$$[\ I_i \times \{D_i\} \subseteq H^{-1}(\textbf{true})\ ].$$

The completeness condition is a condition, on a concept description, that is specified by H (evaluated to be **true**) and is satisfied by all instances of the concept. In section 2.3, we will see specification of it in different models.

Definition: The **consistency condition** in the general model is a basic requirement for a concept description D_i, i=1, ..., u, which is expressed as:

$$[\ I_j \times \{D_i\} \subseteq H^{-1}(\textbf{false})\ ],\ \ i \neq j,\ j \in \{1, ..., u\}$$

The consistency condition is a condition, on a concept description, that is specified by H (evaluated to be **false**) and is satisfied by all instances of other concepts. In section 2.3, we will see specification of it in different models.

The general model of conceptual induction is described below:

INPUT:

1. $I = \{ d_1, ..., d_m \}$, a set of known (or observed) instances;
2. $U = \{ K_1, K_2, ..., K_u \}$, a set of constant symbols representing names of all the possible concepts in the universe of discourse;

---

[8] The two conditions were originally defined in [Michalski 1983b] in the format:

Completeness condition: $(\forall i)\ (M_i \Rightarrow D_i),\ i \in J,$

Consistency condition: $(\forall i)(\forall j)(D_i \Rightarrow \neg M_j),\ i \in J,\ j \in J,\ j \neq i$

where $J = \{1, ..., u\}$, and M_i is a description satisfied by all known instances of concept C_i and only by those instances. Note that, saying that an instance satisfies a description means that the properties of the instance satisfy the conditions expressed in the logic expression of the description. Michalski's definitions work only for a logic-based model. In order to describe and compare the existing models as well as our own model, we used more general terminology and definitions. In the logic-based model, our definitions are more general and are equivalent to his definitions only under the restriction that every instance is described by a fundamental conjunction (i.e., a conjunction with all $p_i \in P$ presented). Our definitions of conditions are easier to check in an incremental induction which is needed in a database environment.

3. A mapping ρ: I $\rightarrow$ U, (In supervised induction ρ is given by an outside teacher, that is a human or another program, while in unsupervised induction, it is determined by an classification component of the induction system);

4. A function H: I×E $\rightarrow$ {**true, false, uncertain**}. H is called the criterion function.

OUTPUT:

D_1, ..., D_u, descriptions of concepts, such that for all known instances, in I both the completeness condition and consistency condition are satisfied.

We will specialize this general induction model into specific models for characterizing different inductive learning methods.

2.2. Approaches to Conceptual Induction

In this section we give an overview on approaches to conceptual induction. For each approach we briefly describe the concept representation, control strategy, knowledge utilization and other aspects of induction and discuss its advantages and disadvantages from the point of view of an inductive database.

Conceptual induction is one type of inductive learning in the paradigms called "symbolic concept acquisition" (SCA) [Michalski 1986]. As we indicated earlier, conceptual induction has two characteristics that differentiate it from other learning methods: (1) It learns from examples (instances) by induction. The instances are the input to the learning system. (2) It constructs symbolic descriptions of concepts, which are the output of the learning system. The first characteristic differentiates it from deductive concept learning whereby concept descriptions are deduced from domain theory. The second characteristic differentiates it from neural modeling and statistical methods. With neural modeling methods, the learned result is represented as numeric weights on the connections between the elements of a network. This result does not give any concept description for human understanding. Statistical methods do not generate symbolic concept descriptions either, where the learned result is the numeric measures on statistical aspects

of concepts. A comparison of machine learning with artificial intelligence methods and statistical methods is given by Gams and others [1991].

The early approaches that perform conceptual induction include Winston's method [1970, 1975] which performs incremental, supervised learning and uses a semantic network [9] as the representation in generating concept descriptions of block world instances. Structured descriptions[10] can be represented and processed. Although semantic networks are very expressive to depict various semantic relationships and logical relationships among attributes and objects, in his method only conjunctive concept descriptions are generated. This methods can perform many types of generalization including dropping-condition rule, turning-constant-to-variable rule, and climbing-generalization-tree rule [Dietterich et al. 1982; Michalski 1983b]. The disadvantages of the method are that it requires that both positive and negative examples be chosen carefully and presented to the system in carefully selected order; and it cannot handle noisy data. This method has not been applied to real world application.

Quinlan's ID3 [1983, 1986] is a method of supervised learning from examples. It uses a conjunction of attribute-value pairs to represent instances and constructs a decision tree to represent concepts learned. At each node of the tree an attribute is tested to determine the branches. At each leaf of the tree a concept identifier is attached. A path from the root to a leaf corresponds to a conjunctive description of a concept (or a classification rule). The quality of the concept description can be measured by the number of leaves and average length of the path, which correspond to number of rules and simplicity of rules. To attempt to optimize the quality, a heuristic measure based on information theory is used to select an attribute for testing at each mode. This is a local optimization which helps to find good decision tree but not the best one. The underlying strategy of ID3 is non-incremental learning which requires all examples are simultaneously available for processing. However, some variants of ID3 have been developed for

---

[9] A semantic network consists of a collection of nodes interconnected by arcs. Each node represents an object and each arc represents a binary relation between two objects. Typically, arcs convey some semantics about the objects.

[10] A structural description is a description that represents a structural object composed of several components.

incremental learning [schlimmer and Fisher 1986, Utgoff 1988]. ID3 has several advantages: (1) It is simple and easy to implement. (2) it is computationally efficient for generating concept descriptions from a large number of examples. The disadvantages of this method are: (1) Its representation of concepts as a decision tree is relative simple and lacks the expressive power for complex descriptions. For example, it is not able to handle structured concepts. (2) It has a limited generalization capability and uses only selective generalization. No new attributes can be constructed during induction process. (3) It does not incorporate available domain knowledge to guide inductive search for better concept descriptions. (4) It does not properly deal with inconclusive data [Spangler 1989].

Chan [1989] developed BCT, a variant of ID3, which represents concept descriptions as a binary polythetic decision tree. At each node a combination of attributes is tested instead of only one attribute in ID3. This method can discover relationships between attributes; for example, in an exclusive-or domain, BCT surpasses ID3.

Kubat [1991] introduces FLORA, a learning system based on ID3's decision tree. Instead of using the information entropy, this method determines the importance of attributes by means of the rough set theory. The system is designed for learning in case that input instances are not reliable. It is an incremental method. A good feature of the method is that irrelevant knowledge can be handled by forgetting.

Michalski and others use a supervised inductive learning methodology, the Star methodology, in their systems AQ and INDUCE [Michalski 1983b]. The APC (Annotated Predicate Calculus) is used as the representation language to express concept descriptions. This representation formalism can express structured concepts with disjunctions or internal disjunctions.[11] The logical foundations of his methodology are the completeness and consistency conditions.[12] The completeness condition of a concept description requires that the description

[11] An internal disjunction is a logical expression with or-form value, which will be discussed in Chapter 3.

[12] Detailed in Chapter 3.

is satisfied by all examples of the concept. The consistency condition of a concept description requires that the description is not satisfied by any examples of other concepts. The Star method randomly selects a positive instance and maximally generalizes it into many partial concept descriptions. If any of them is complete and consistent, then this description is a concept description. To those partial concept descriptions that are not consistent, specialization is applied because those descriptions are too general so as to cover some negative instances. The specialization will generate many consistent descriptions which are called a star. From the star select one description with the highest preference. The process is repeated for another positive instance that is not covered by any selected consistent description so far until all positive instances are covered. Finally, the disjunction of all the selected consistent descriptions is the concept description. This method can be regarded as a search through the description space. The search is constrained by the size of a star and number of concept descriptions generated. The search is guided by a preference criterion called the lexicographic evaluation functional (LEF) and the built-in induction bias of finding the most general partial concept descriptions. This method provide the facility for incorporating domain-specific knowledge into the generalization process, and the facility for constructive induction[13]. The advantages of the method are (1) it has a powerful representation formalism; (2) it can incorporate available domain knowledge for guiding the induction process; (3) it has many forms of generalization indicated by the number of generalization rules employed; and (4) it can perform constructive induction. This method has following disadvantages: (1) it is not an incremental strategy; (2) it cannot deal with noisy data properly, and the algorithm cannot generate approximate results.

Michalski and others introduced AQ15 [1986] which is an incremental conceptual induction system but with a less power of representation. AQ15 associates two weights with each conjunction of a concept description, and the weights are used to measure the fit between an instance and a concept. This method is based on the same exact matching technique of his other

[13] Constructive induction is an inductive method that is able to create or construct new terms for attributes and relationships (see Chapter 3).

method, that two attribute values are either matching or totally not matching. This makes it difficult to handle matching between continuously-valued attributes. The method also shares an assumption with his other method, that there exist many complete and consistent descriptions for each concept. This may not be true for many application domains, for example, in industrial diagnosis databases or patient treatment databases in a hospital.

Clark and Niblett designed CN2 [1989], a supervised induction system which draws ideas from both AQ and ID3. The learned concepts are in the form of decision list, which is an ordered set of if-then rules. Two heuristic evaluation functions are used to control search during rule construction. One evaluation function, based on the information-theoretic entropy, is used to determine the quality of the concept descriptions (rules) for selecting the best one. The second evaluation function, based on the likelihood ratio statistic, is used to test if a rule reflects a genuine correlation between attribute values and the concepts.

Quinlan [1990] introduced FOIL, a supervised inductive learning system that combines techniques from both AQ and ID3. FOIL learns structured concepts and uses an information-based heuristic to guide its search for better descriptions. The concept descriptions are represented as Horn clauses. The method can find recursive definitions, and can develop inexact but useful rules. However, it does not handle continuous values and is non-incremental.

Conceptual clustering is a type of unsupervised inductive learning, introduced by Michalski [1980b, 1981, 1983a, c] in system CLUSTER/2, that produces a classification scheme over a set of instances and generates descriptions for classes, which correspond concepts. Different from conventional methods of cluster analysis and numerical taxonomy. Conceptual clustering classifies objects on the basis of a concept description quality, such as simplicity of concepts, fit or generality of the classes, while conventional methods are based on a numerical function of individual objects. CLUSTER/2 employs a top-down tree building strategy. At the root of the tree, all instances are grouped into several disjoint clusters that optimize the criterion of clustering quality. Conjunctive descriptions are also generated for each cluster. Then, the process is recursively applied to the set of instances in each cluster at the next level of the tree. The criterion

is mainly based on two factors: (1) the fit between a set of clusters and an instance, and (2) the simplicity of the cluster descriptions. The first factor reflects the sum of the ratio of number of observed instances covered by a cluster description, and the second factor is the total number of simple logical statements in all cluster descriptions. The advantages of the method are: (1) It generate clusters with descriptions easily understood by humans. (2) The resulting clusters usually have more concise conceptual meanings. The shortcomings of the method are: (1) It process only conjunctive descriptions with nominal attributes. It cannot not handle structured concepts. (2) It is computational expensive and non-incremental. (3) It performs exact matching of attribute values so that it cannot deal with noisy data very well. (4) Only limited domain knowledge can be incorporated and there is no deductive inference support.

New methods for inductive inference control and new organizations of knowledge and data for efficient inductive inference are needed in the database and knowledge base environment. In the database environment, an inductive inference system requires an incremental processing method which is needed in many real life environments and which has been studied by many researchers [Lebowitz 1982; Kolodner 1983; Reinke and Michalski 1986; Rendell 1986; Schlimmer and Fisher 1986; Fisher 1987; Schlimmer 1987; Gennari et al. 1989; Clearwater et al. 1989; Thompson et al. 1989].

Fisher [1987] presents COBWEB, an unsupervised inductive system, which incorporates instances into a tree following a top-down strategy. Starting from the root of the tree, it classifies the instance along an appropriate path, updates distributional information at each node, and performing one of the following operations: (1) placing an instance in an existing class; (2) creating a new class; (3) merging two classes into one class; and (4) splitting a class to two classes. Each node of the tree represents a class, corresponding to a concept, which is described by a list of attribute-value pairs and their associated probabilities P(Ai=Vij | Ck), where Ai=Vij is an attribute-value pair and Ck is a concept. The induction is guided by a heuristic evaluation measure called category utility which is defined as "the increase in the expected number of attribute values that can be correctly guessed given a partition over the expected number of correct

guesses with no such knowledge". The strong point of COBWEB is its incremental and computationally economical control strategy. It also has some drawbacks: (1) It does not have an expressive representation capability, only handle nominal attribute-value pairs, which implies that it processes only exact attribute matching and has a low capability to deal with noisy data. It cannot handle structured concepts. (2) It cannot represent relationships between attributes. Many concepts depend not only on their attribute values but also on the relationships of their attributes, such as exclusive-or relationship. (3) It does not employ domain knowledge and does not support constructive generalization.

Lebowitz's UNIMEM [1986, 1987] is an incremental unsupervised learning method that uses a Generalization-Based Memory, a hierarchical data structure to represent concepts learned. In his method instances and concepts of different levels of generalization are recorded in the hierarchy using discrimination networks and the concepts already learned are constantly updated and new ones are constantly sought. Each instance may be stored with multiple nodes, so the classes are not disjoint. Each node in the concept hierarchy is associated with a concept description which is a list of attribute-value pairs. This method has following advantages: (1) It allows inexact matching between attributes, and maintains a confidence value for each attribute in concept descriptions, so that it is possible to deal with noisy data. (2) It can handle many types of attributes including continuous numeric values. (3) It is computationally efficient for incremental processing. The disadvantages are: (1) It cannot represent logical relationships between attributes. (2) It does not represent structural concepts. (3) It does not utilize domain specific knowledge and general knowledge for guiding the induction process. Lebowitz views UNIMEM both as a concept formation system and as an intelligent information system that can incorporate large amounts of data into memory and retrieve appropriate information in response to user queries. This view is very close to the view of an inductive database.

Knowledge-directed induction is a method in which inductive heuristics and domain knowledge can be utilized to guide inductive inference, to constrain the search space for efficient search, and to derive useful descriptions. The abilities of this type of induction derive in part by

reasoning over an explicit knowledge base. Here the knowledge base is a collection of structured data with two properties: (1) A human can interpret the data as sentences which constitute at least part of the knowledge that the system exhibits; and (2) the data influence the behavior of the system in the right way [Levesque 1989]. The knowledge used in inductive inference includes inductive bias and domain theory.[14]

All induction systems employ biases including selection of attributes, allowed values for attributes, system thresholds, quality evaluation functions, choice of representation formalism, types of generalizations allowed, and so on. Inductive bias is not only necessary for choosing promising hypotheses but also necessary for efficient inductive processing. Much research work has been conducted in this area [Mitchell 1980; Lenat 1983; Utgoff 1983, 1986a, 1986b; Haussler 1986; Russell 1987; Rendell et al. 1987a, 1987b]. Most of the existing induction systems fix the biases and embed them in the system implementation. However, different application problems may require different set of biases and system developer may not fully understand the effect of each bias, so a large portion of the efforts of developing an induction system for an application lies in the testing and selecting a right set of inductive biases. Systems that provides inductive bias management and automatic bias shifting will make the induction system flexible and easily adjusted for a new application.

Although explanation-based learning [Mitchell et al. 1986; DeJong et al. 1986] is a type of deductive learning method, they can be used to support inductive learning with domain knowledge [Mitchell 1983; Lebowitz 1986; Danyluk 1987; Bergadano 1988; Callan 1989; Flann and Dietterich 1989]. An important advantage of using domain knowledge is that deduced new features and relationships can better characterize the concepts. The problem with the explanation-based learning methods is that most work emphasizes only an induction with a small number of instances. There is a need for research to be performed to incorporate knowledge-directed conceptual induction in a data intensive environment. Lebowitz proposed a

[14] An inductive bias is a heuristic which can be used to guide the inductive process. Domain theory is a set of rules which specify the relationships among features of a concept.

method that applies explanation-based techniques after the induction and uses a measure, called predictability to constrain the explanation process. Danyluk's method applies a simple explanation to instances before the induction, so that the induction is not as susceptible to the influence of coincidence. Flann and Dietterich described a method, called IOE (induction over explanation), for supervised induction. Their method applies the domain theory to construct explanations from multiple instances, then performs induction over those explanations. Elio and others [1991] describe an inductive system LAIR which uses an incremental deductive strategy for constructing new features into concept descriptions. The method reduces the amount of inference efforts by avoiding extension of each example description with all derivable features. Nunez [1991] presents an induction method called EG2, which uses two types of background knowledge: specification of hierarchical attributes and measurement cost associated with each attributes, in order to achieve a better quality of induction result. EG2 builds a decision tree with extended generalization rules: climbing-generalization-tree rule and add-alternative rule. The quality is measured by the information cost function which is computed as the ratio of the cost of an attribute to the discrimination efficiency of the attribute. An economic classification is the major feature of his method.

2.3. Different Models of Conceptual Induction

In this section, we first specialize the general model of conceptual induction introduced in section 2.1 into two specific models, the logic-based model and the set-based model, to characterize different existing approaches. Then, we compare these two models and analyze their limitations.

2.3.1. The Logic-Based Model

The logic-based model describes a concept by a logical expression. Methods described by Dietterich [1983], Michalski [1977, 1980a, 1983a], Hayes-Roth [1976b, 1977, 1978], Vere [1975, 1977, 1978, 1980], Winston [1970, 1975], Stepp[1984] and Quinlan [1983, 1986] are examples of the logic-based model.

In order to describe the logic-based model, let us give a few more definitions.

$P = \{ p_1, p_2, ..., p_m \}$ is a set of symbols called predicates.

$A = \{ a_1, a_2, ..., a_N \}$ is a set of symbols called constant terms.

$X = \{ x_1, x_2, ..., x_M \}$ is a set of symbols called variables.

Definition: An **atomic expression** is a statement in the form $p_i(t_1, t_2, ..., t_{i\_n})$ where $p_i \in P$ and each argument t_j, j=1,...,i_n, is a term which can be either a constant or a variable.[15]

A **logical expression** is defined as:

(1) If exp is an atomic expression, then it is a logical expression;

(2) If exp1 and exp2 are logical expressions, then the conjunction exp1 $\wedge$ exp2, the disjunction exp1 $\vee$ exp2, and the negations $\neg$exp1 and $\neg$exp2 are logical expressions.

We define E, in the logic-based model, as the set of all logical expressions. An **instance** d in the logic-based model is a conjunction of atomic expressions. S is the set of all possible instances. D_i, a **description of concept** C_i, i=1, ..., u, is a logical expression.

We say a logical expression A is a **generalization** of a logical expression B, if and only if B tautologically implies A, and is represented as

$$B \Rightarrow A$$

In the logic-based model the **criterion function**

$$H: I \times E \rightarrow \{\textbf{true, false, uncertain}\}$$

can be defined as

$$H(d, D_i) = \begin{cases} true, & if\ d \rightarrow D_i\ ; \\ false, & otherwise. \end{cases}$$

In the logic-based model there is no pair of d and D_i such that $H(d, D_i)$ = **uncertain**. In other words, H^{-1}(**uncertain**) = $\varnothing$.

Then the completeness condition

$$[\ I_i \times \{D_i\} \subseteq H^{-1}(\textbf{true})\ ]$$

---

[15] In a different implementation, an atomic expression may have different formats. For example, the expression $p_i(t_1, t_2, ..., t_{i\_n})$ may be represented in the form $(p_i, t_1, t_2, ..., t_{i\_n})$.

can be specialized in the logic-based model as following.

Definition: The **completeness condition** in the logic-based model

is a basic requirement for a concept description D_i, i=1, ..., u, which is expressed as

$$(\forall d)\{\ (d \in I_i) \rightarrow (d \rightarrow D_i)\ \},\quad d \in S \tag{2.1}$$

That is, the description D_i of the concept C_i should be the generalization of all the known instances of the concept.

The expression "$d \Rightarrow D_i$" indicates that D_i is a generalization of d which means that for all situations, if d is true then D_i is also true. The expression "$d \Rightarrow D_i$" may have many possibilities:

color(apple, red) $\Rightarrow$ color(apple, red $\vee$ green)

color(apple, red) $\Rightarrow$ color(apple, red) $\vee$ size(apple, big)

color(apple, red) $\wedge$ size(apple, big) $\Rightarrow$ color(apple, red)

type(apple, fruit) $\Rightarrow$ type(apple, food)

Then the consistency condition

$$[\ I_j \times \{D_i\} \subseteq H^{-1}(\textbf{false})\ ],\ \ i \neq j,\ j \in \{1, ..., u\}$$

can be specialized in the logic-based model as follows.

Definition: The **consistency condition** in the logic-based model is a basic requirement for a concept description D_i, i=1, ..., u, which is expressed as

$$(\forall j)(\forall d)\{\ (d \in I_j) \rightarrow \neg(d \rightarrow D_i)\ \},\quad j=1,...,u;\ j \neq i;\ d \in S \tag{2.2}$$

That is, the description D_i of the concept C_i is not a generalization of any known instances of other concepts.

The general model of conceptual induction can be specialized into the following logic-based model.

INPUT:

1. $I = \{ d_1, ..., d_m \}$, a set of known (or observed) instances, and each instance in I is a conjunction of atomic expressions.
2. $U = \{ K_1, K_2, ..., K_u \}$, a set of constant symbols representing names of all the possible concepts in the universe of discourse;
3. A mapping $\rho: I \rightarrow U$;
4. A criterion function $H: I \times E \rightarrow$ {true, false, uncertain} defined as:

$$H(d, D_i) = \begin{cases} true, & \text{if } d \rightarrow D_i ; \\ false, & otherwise. \end{cases}$$

OUTPUT:

$D_1, ..., D_u$, descriptions of concepts, such that for all known instances, the completeness condition (2.1) and consistency condition (2.2) are satisfied.

Conceptual induction in the logic-based model is a process to find descriptions D_i for concepts C_i, for i=1, ..., u, such that the descriptions D_i satisfy both completeness and consistency conditions. For the resulting concept descriptions, the **inductive assertion** can be expressed as:

$$(\forall d)\{ [(d \rightarrow D_i) \wedge (\bigwedge_{j \neq i} \neg(d \rightarrow D_j))] \rightarrow (d \in C_i) \}, \quad i=1,...,u; \quad d \in S$$

That is, if we know that D_i is a generalization of d and no other concept description D_j, $j \neq i$, is a generalization of d, then we can assert that d is an instance of concept C_i. The inductive assertion is a plausible rule with high certainty. This rule can be used to define an approximation λ' of the mapping λ:

$$\lambda'(d) = \begin{cases} K_i, & \text{if } (d \rightarrow D_i) \wedge (\bigwedge_{j \neq i} \neg(d \rightarrow D_j)) \\ \Theta, & \text{otherwise.} \end{cases}$$

where $K_i \in U$, and Θ is a reserved symbol in U denoting the class of unclassifiable instances. There exist instances in S which are not covered by any concept description D_i. In this situation, the logic-based model will not be able to classify those instances. In other models, similarity between an instance and each of the concepts can be computed, and the instance will be classified into the most similar concept.

With some restrictions, we can simplify the completeness and consistency conditions. Let us assume that for every $d \in S$, d is a conjunction of atomic expressions which are restricted to the form $p_k(v)$, a predicate with one argument; and that every d is a fundamental conjunction, i.e., a conjunction of exact m (the total number of predicates in P) atomic expressions with every p_k ($k = 1, ..., m$) present. Now p_k can be regarded as a discrete variable with a value domain DOM_{p_k}, and all the possible d's, called **events**, constitute an m-dimensional event space. Every event defines a point e in the event space; there is a one-to-one correspondence between a point e and an event d. We use e to represent an instance; then d is the description of the instance. Now a concept can be viewed as a set of instances, and its description defines a region in the event space.

Definition: A description D **covers** an instance e, if and only if the point corresponding to e is in the region defined by D.

The purpose of conceptual induction is to find a description (defining a region) for each concept to cover only its instances (i.e., points).

Let e be a point defined by an event d, R_i be a set of points corresponding to the known instances of concept C_i, and S_i be a set of points covered by the region defined by D_i, that is

$R_i = \{ e \mid e$ is a point defined by d and $d \in I_i \}$, i = 1,...,u

$S_i = \{ e \mid e$ is a point defined by d and $(d \Rightarrow D_i) \}$, i= 1,...,u

Now the completeness condition for a concept description C_i, i=1,..., u, can be simplified[16]

$$[R_i \subseteq S_i] \tag{2.3}$$

which means all known instances of concept C_i should be covered by the concept description D_i. The completeness condition can be shown in Figure 2.3.

The consistency condition for a concept description C_i, i=1,...u, can be simplified as:

$$(\forall j)[R_j \cap S_i = \varnothing], \quad j=1,...,u;\ j \neq i \tag{2.4}$$

---

[16] **Proof:** We prove that expression (2.3) is equivalent to expression (2.1) by proving that either of the expressions can be deduced from the other.

It is easy to see that expression (2.3) is equivalent to the following expression:

$$(\forall e)[(e \in R_i) \rightarrow (e \in S_i)] \tag{2.3.1}$$

Now, let us deduce expression (2.3.1) from expression (2.1)
For every $i \in \{ 1,...,u \}$ and $d \in S$ we have the following deduction:
If e is a point defined by d and $e \in R_i$,

then by the definition of R_i we have $d \in I_i$.
By the condition (2.1), $d \Rightarrow D_i$; and by the definition of S_i we get $e \in S_i$.

So condition (2.3.1) is true.

Deducing (2.1) from (2.3.1) can be similarly proved.

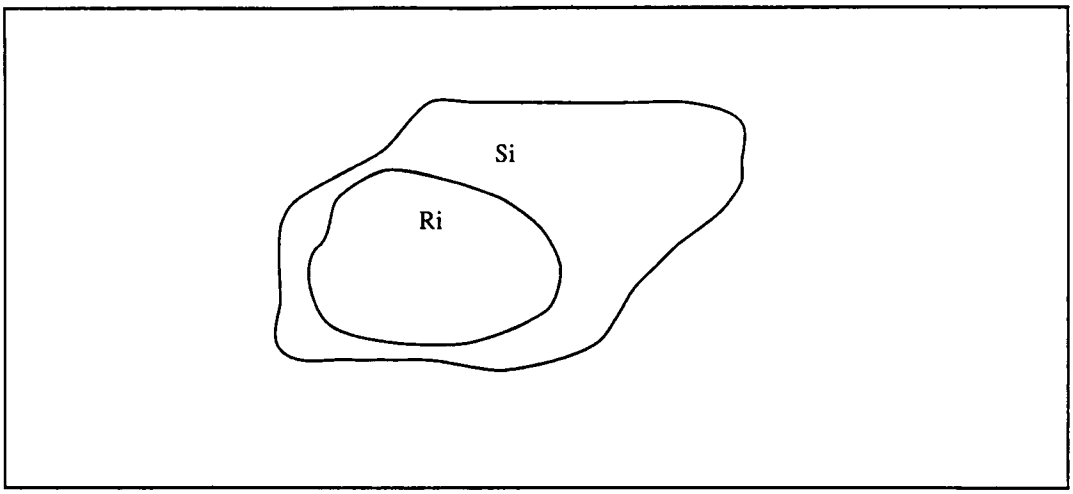

Figure 2.3. The Completeness Condition

which means[17] no known instances of other concepts are covered by the description of the concept C_i. The consistency condition can be shown as in Figure 2.4.

The two conditions expressed as (2.3) and (2.4) provide a set-theoretic description of completeness and consistency.

---

[17] **Proof**: We prove that expression (2.4) is equivalent to expression (2.2) by proving that either of the expressions can be deduced from the other.

It is easy to see that expression (2.4) is equivalent to

$$(\forall j)(\forall e)[(e \in R_j) \rightarrow \neg(e \in S_i)],\ j = 1,...,u;\ j \neq i \qquad (2.4.1)$$

First, let us deduce expression (2.4.1) from (2.2)

For every $i,j \in \{1, ..., u\}$; $i \neq j$, and every d, we have the following deduction:

If e is a point defined by d,

from $(e \in R_j)$ we have $d \in I_j$ (by definition of R_j)
then we have $\neg(d \Rightarrow D_i)$ (by condition (2.2)).
Since $S_i = \{ e \mid e$ is a point defined by d and $(d \Rightarrow D_i) \}$
Assume $e \in S_i$ then $(d \Rightarrow D_i)$ which contradicts $\neg(d \Rightarrow D_i)$.
So we have $\neg(e \in S_i)$.

Therefore, we get (2.4.1).

Deducing expression (2.2) from (2.4.1) can be similarly proved.

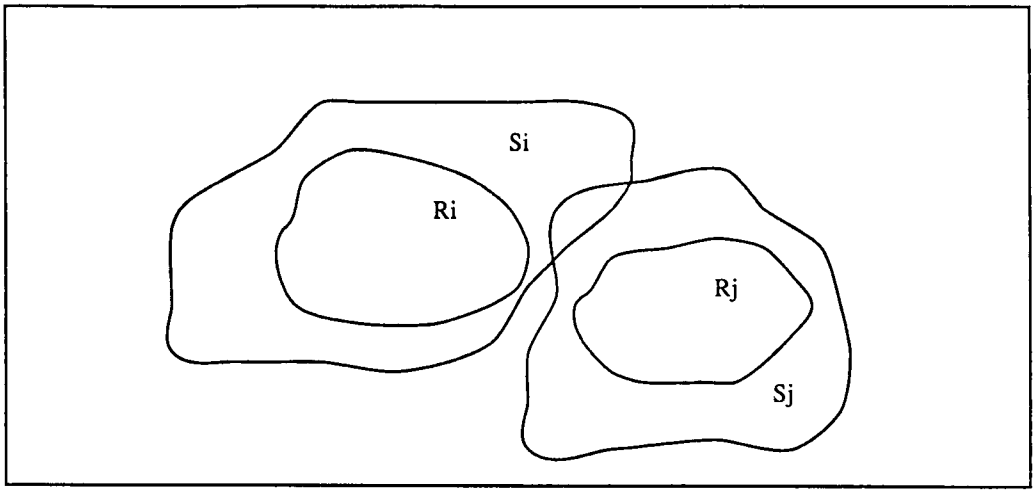

Figure 2.4. The Consistency Condition

2.3.2. The Set-Based Model

The **set-based model** describes a concept and its instances by a set of features defined below. Methods described by Lebowitz [1986] and Fisher [1987] are examples of the set-based model.

Let us define some symbols and terms used in the set-based model.

$A = \{ a_1, a_2, ..., a_N \}$ is a set of constant symbols representing names of attributes. For every $a_i \in A$ we define a domain DOM_a_i which is a set of possible values of a_i.

A **feature** is an attribute-value pair. $F_i = \{ (a_i, v_{ik}) \mid (a_i \in A) \wedge (v_{ik} \in DOM\_a_i) \}$ is a set of features associated with attribute a_i. $F = \bigcup_{i=1}^{N} F_i$ is the set of all possible features.

In the set-based model, we define E to be a set of descriptions where each **description** is a set of features associated with different attributes. An **instance** d is a description $\{ f_1, ..., f_k \}$, where $f_i \in F_i$, i=1, ..., k, and d is an element of E. Let S be the set of all possible instances. D_i is a set of features associated with different attributes representing a description of concept C_i.

In the set-based model the **criterion function**

$$H: I \times E \rightarrow \{\textbf{true, false, uncertain}\}$$

can be defined as:

$$H(d,D_i) = \begin{cases} true, & if\ d \supseteq D_i; \\ false, & if\ d \cap D_i = \varnothing; \\ uncertain, & otherwise. \end{cases}$$

The inductive process is to construct D_i from I_i, i=1,...,u, such that the two basic criteria of conceptual induction can be best satisfied. With the definition of the criterion function, the completeness condition

$$[\ I_i \times \{D_i\} \subseteq H^{-1}(\textbf{true})\ ]$$

can be specialized in the set-based model as follows.

Definition: The **completeness condition** in the set-based model requires that every concept description D_i, i=1, ...,u, should include only common features of its known instances. This condition is formally expressed as:

$$(\forall d)\{\ (d \in I_i) \rightarrow (D_i \subseteq d)\ \},\quad d \in S \tag{2.5}$$

Then the consistency condition

$$[\ I_j \times \{D_i\} \subseteq H^{-1}(\text{false})\ ],\ i \neq j,\ j \in \{1, ..., u\}$$

can be specialized in the set-based model as following.

Definition: The **consistency condition** in the set-based model requires that every concept description D_i, i=1, ..., u, should not include any features of known instances of other concepts. This condition is formally expressed as:

$$(\forall d)(\forall j)\{\ (d \in I_j) \rightarrow [(D_i \cap d) = \varnothing]\ \},\ j=1,...,u;\ j \neq i;\ d \in S \tag{2.6}$$

The general model of conceptual induction can be specialized into the following set-based model.

INPUT:

1. $I = \{ d_1, ..., d_m \}$, a set of known (or observed) instances, where each instance in I is a set of features.
2. $U = \{ K_1, K_2, ..., K_u \}$, a set of constant symbols representing names of all the possible concepts in the universe of discourse;
3. A mapping $\rho: I \rightarrow U$;
4. A criterion function $H: I \times E \rightarrow$ {**true, false, uncertain**} defined as:

$$H(d,D_i) = \begin{cases} true, & \text{if } d \supseteq D_i; \\ false, & \text{if } d \cap D_i = \varnothing; \\ uncertain, & otherwise. \end{cases}$$

OUTPUT:

$D_1, ..., D_u$, descriptions of concepts, such that for all known instances, both the completeness condition (2.5) and the consistency condition (2.6) are satisfied.

With the resulting concept description D_i for concept C_i, i = 1,..., u, the **inductive assertion** can be expressed as

$$(\forall d)\{[(d \supseteq D_i) \wedge (\bigwedge_{j \neq i} (d \cap D_j = \varnothing))] \rightarrow (d \in C_i)\}, \quad d \in S$$

That is, if we know that an instance d contains all features of a concept description D_i and no features of other concept description D_j, $j \neq i$, then we can assert that d is an instance of concept C_i. This inductive assertion can be used to defined an approximation λ' of the mapping λ:

$$\lambda'(d) = \begin{cases} K_i, & \text{if } (d \supseteq D_i) \wedge (\bigwedge_{j \neq i} (d \cap D_j = \varnothing)) ; \\ \Theta, & otherwise. \end{cases}$$

where $K_i \in U$ and Θ is a reserved symbol in U denoting the class of unclassifiable instances. It is too restrictive to cover all instances. Therefore, most induction systems of set-based model use

the similarity of an instance to each of the concepts to determine the classification. The similarity is computed based on the number of common features between an instance and a concept, and statistical weights of the features.

Example: Let the concept Ci named "GoodStudent" have a description

Di = { (Grade = A), (#of-friends = many) }

and an instance be

d = { (Name = John), (Grade = A), (Major = Math),
(#of-friends = many) }

then we can say d $\in$ Ci, which means John is a "GoodStudent" because d $\supseteq$ Di.

[End of Example]

The above two conditions are too restrictive to be useful in a real world application where input data may have noise, including missing features, erroneous features and approximate values. In the set-based model, the criterion function may have a value "uncertain". The two conditions can be relaxed to deal with those "uncertain" cases. In the set-based model all features of a description are assumed to be **independent** which means that each feature can be added, deleted or modified independently of other features during the inductive process. Because of independence in the set-based model, the two conditions can be easily relaxed in the induction process by modifying a numerical value attached to each feature of a concept to represent the quality of this feature in characterizing instances of the concept. Different set-based systems may have different computing methods for evaluating the quality of concept features, but most of them rely on two factors: the completeness factor and the consistency factor which are defined below.

Definition: The **completeness factor** of a feature f of concept C_i in relation to a concept description D_i, i = 1, ..., u, is equal to the ratio of the number of instances in I_i that have feature f to the total number of instances in I_i,

$$FR_1(D_i,f)=\frac{card(\{d \mid [d\in I_i]\wedge[f \in d]\})}{card(I_i)}$$

where card(A) represents the cardinality of a set A.

Definition: The **consistency factor** of a feature f of concept C_i with a concept description D_i, i = 1, ..., u, is equal to the ratio of the number of instances in I_i that have feature f to the total number of known instances that have feature f, i.e.,

$$FR_2(D_i,f)=\frac{card(\{d \mid [d\in I_i]\wedge[f \in d]\})}{card(\{d \mid [d\in I]\wedge[f\in d]\})}$$

When the completeness factors of all features of C_i is 1 then D_i, the description of C_i, satisfies the completeness condition. When the consistency factors of all features of C_i is 1, then D_i, the description of C_i, satisfies the consistency condition. Allowing features in D_i to have completeness and consistency factors with values less than 1 relaxes the completeness and consistency conditions. The description of a concept becomes a fuzzy set of features. The numeric values to represent the membership of features are based on the completeness and consistency factors. Since features in a description are assumed to be independent, the numeric values can be modified independently to incorporate new instances into a concept description. The time for modifying a concept description is linear in the number of features in a new instance's description. This is because that only the weight associated to each feature is modified and the time for modifying each weight does not depend on the number of features. Therefore, the set-based model is able to perform incremental conceptual induction efficiently on noisy input data.

2.3.3. Comparison and Discussion

The logic-based model matches an instance to a concept description based on the logical implication relationship between them. In the set-based model matching is based on the set-inclusion relationship. In the logic-based model a concept description is either a generalization of an instance or not a generalization of it. This "all-or-none" principle makes the logic-based model

less suitable for application domains with data of great diversity because it often happens that no concept descriptions can be found to satisfy both completeness and consistency conditions, except the trivial concept description, that is the disjunction of all the input instances. The set-based model, on the other hand, does not follow the "all-or-none" principle. The number of common features in two descriptions can be used to measure the degree of match. The features in a concept description are associated with numeric weights that are computed based on the frequencies of the features.

Under some assumptions a set-based model can be considered a special case of a logic-based model. If we assume that all features in a set-based description are conjoined together, that is, each description stands for a conjunction, and that completeness and consistency conditions are strictly satisfied by concept descriptions, then the model can be considered a logic-based model with several restrictions. Those restrictions include: limitation of feature forms, restricted logic expressions, and limited generalization rules. However, the purpose of the set-based model is not to simulate a naive logic-based model, but to deal with a large amount of noisy data by an efficient incremental method.

It is difficult for the logic-based model to deal with noisy input because with strict logical relations, an instance's description is either covered or not covered by a concept description. When incorporating an instance into a concept description, the system has to generalize the concept description to cover the instance's description. When generalizing a concept description, the system must ensure that no descriptions of other concepts' instances are covered. With one wrong feature in an input instance, the logical expressions of concept descriptions may change drastically. In contrast, in the set-based model one wrong feature in an instance has a small effect on the concept descriptions.

The assumption of feature independence makes the set-based model efficient for incremental induction. In the logic-based model features are not assumed independent and may be related in a logical expression by connectives. Then strict logical relations exist between features, and the completeness and consistency conditions have to be observed during all

generalization and specialization operations. Since the number of the logic combinations of features can be much larger than the number of features, searching the logical expression for a concept description to incorporate an instance is very time-consuming. All this makes the logic-based model inefficient for incremental induction. The following example gives further explanation. In contrast, in the set-based model, the incorporating process involves only independent modification of the numeric value of each feature in a concept description.

Example: In a logic-based model suppose three logical expressions D_1, D_2 and D_3 have been obtained to describe three concepts C_1, C_2 and C_3 (as shown in Figure 2.5). Now i, a new instance of C_1, arrives. If D_1 does not cover i, then D_1 needs to be generalized to cover i. If i is covered by D_2 (or D_3), then D_2 (or D_3) needs to be specialized to exclude i. When generalizing D_1, we need to ensure that all known instances of D_2 and D_3 should not be covered by D_1, and when specializing D_2 (or D_3), we need to ensure that none of the instances of C_2 (or C_3) should be excluded from D_2 (or D_3). When the number of known instances is large, checking all the instances can become very inefficient. [End of example]

In the set-based model, the representation is not expressive (see the following example) because structural concepts[18], complex logical expression of features (such as disjunctions) and various relationships (including causal relationships, and concurrent relationships of attributes and components) cannot be represented in the set-based model. For example, in case of concepts with exclusive-or relationship between features, the feature itself does not characterize a concept; instead, the relationship between features is important to characterize a concept.

Example: In the set-based model, the following generalization for structural concepts with internal disjunction cannot be represented and processed.

By applying the generalization rule (which will be discussed in Chapter 3), the expressions

(color (block1) = red $\vee$ black)

and

[18] A structural concept is a concept whose instances have internal structures, and is described by features of its components and relationships among those components.

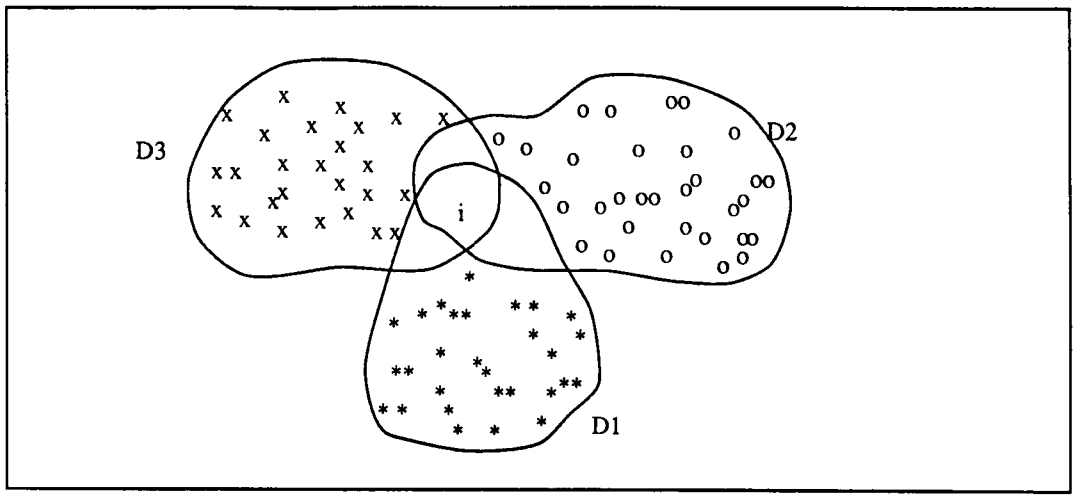

Figure 2.5. Incorporating a New Concept Instance

(color (block1) = blue)

are generalized into

(color (block1) = red ∨ black ∨ blue) [End of example]

The above two conceptual inductive models do not combine a rich representation with an efficient incremental inductive process; however, this combination is required for the conceptual induction in a database environment. In the set-based model all features are independent, while in the logic-based model all features are not independent, and they are related together into a logical expression. In a real situation, however, there usually are some features related to each other, though not all features are related. Feature relationships can be directed by domain knowledge. This is the basic idea of the new model we developed.

2.4. Conceptual Induction in a Database Environment

In this section, we discuss the required properties of a conceptual induction system in a database environment.

The conceptual induction system in a database environment should possess all the following properties:

(1) An incremental control strategy for efficient processing of a large amount of data;

(2) An expressive representation for concept descriptions and knowledge utilization;

(3) The ability to deal with multiple concepts;

(4) Knowledge-directed induction; and

(5) A general methodology.

Although existing conceptual inductive methods may possess one or two of the above properties none of them has all the properties required in a database environment. The integration of all the properties listed above in one inductive system is a challenging task which requires the development of a distinctive new model and methodology of conceptual induction.

A database is not a static repository of data, instead, it stores operational data. Since data may be added, deleted, or changed, the induction on the database cannot be a one time process. Therefore, an incremental induction method is a requisite. An incremental induction method accepts input instances over a span of time and modify existing concept descriptions to incorporate each new instance without rerun from scratch. Incremental induction makes it possible to accumulate knowledge, and to learn new knowledge based on known knowledge. An induction system that reprocesses all the available data when new data is available is not incremental and is not suitable for the data-intensive database environment where the amount of data is huge. In a database environment with a large number of instances, the efficiency of an incremental induction method is very important.

As a database system, an intelligent database with induction capability is not designed for a specific application problem. Therefore, there are possible many types of concepts in different applications. There are two reasons that we need an powerful representation formalism:

(1) To describe a large variety of attributes and relationships of many possible concepts in a database and to present concept descriptions in a high-level, human-oriented forms, we need an expressive representational language which is able to support many types of attributes, structural concepts and logic

relationships between features of concepts (detailed discussion will be given in Chapter 3).

(2) In addition to describing concepts, an intelligent database needs to utilize various types of available knowledge in supporting and guiding inductive inference. This also requires a rich representation for knowledge. Concept representation should allow easy interface between induction and knowledge-supporting components.

However, many conceptual inductive methods are either inefficient for incremental induction or inadequate for expression of complex descriptions. Combining an expressive representation with an efficient incremental control strategy is one of the major problems to be solved.

Because in a real world application there can be many classes of entities and a database may contain many types of objects, the induction system needs to deal with multiple concepts to characterize and classify those objects, not only positive examples and negative examples, as usually assumed in many inductive learning systems.

A human's learning mechanism relies heavily on existing knowledge. Existing knowledge makes learning more efficient and effective. In many applications, people already have some knowledge about the domains modeled by databases. This knowledge can be used for guiding the induction process, generating useful descriptions of concepts and producing meaningful classifications of instances. An intelligent database system should use the available knowledge as much as possible in problem solving. To make use of existing knowledge to enhance the conceptual induction for discovering high-level human-oriented symbolic concept descriptions, decision rules and logical relationships that are concealed in a database, mechanisms should be developed to apply domain knowledge and to incorporate general heuristics in inductive processing.

Since database systems are not domain bound, inductive database systems need to employ a general methodology where many types of generalizations are provided and both supervised and unsupervised induction are supported. However, a general method of inductive inference is usually inefficient. Therefore, techniques need to be developed to solve this difficulty. Knowledge

acquisition, representation and management facilities should be incorporated into the inductive database to let users easily input, store and adjust domain specific knowledge and inductive biases for different applications. The main idea is to separate the general induction capability from the domain specific knowledge and inductive biases so that the knowledge base is exchangeable and inductive biases are easily adjusted.

We summarize the major extensions of induction method as the combination of a rich representation with an efficient incremental processing, and the combination of empirical example-based induction with analytical human-related guidance.

In this chapter, we have given a formal mathematical framework of conceptual induction and an overview of existing methods, developed two inductive models for formally describing those methods, and described the required properties of a conceptual induction system in a database environment. In the next chapter, we are going to present our approach to conceptual induction. Our model and methodology will be discussed in detail.

CHAPTER 3

THE MODEL AND METHODOLOGY OF CONCEPTUAL INDUCTION

This chapter presents the theoretical results of this dissertation research work: the model and methodology of conceptual induction in a database environment. The following chapter presents several applications of an implementation of these results, providing an evaluation of the approach.

The present chapter contains two major sections. In section 3.1, we present the knowledge-directed model. First, we specialize the general induction model developed in chapter 2 to our induction model. Second, we specify the formalism of the representation language. Following that we compare our model with other models. Finally, we describe the underlying database model of the induction system we implemented. In section 3.2, we present the knowledge-directed (KNOWD) methodology. First, we give a formal description of the KNOWD methodology. Second, we describe deductive support to conceptual induction. Third, we present details of the induction algorithms. Then, we discuss the properties of the induction algorithms. Finally, we describe the structure and functionality of the inductive database system.

3.1. The Knowledge-Directed Model

From the analysis in Chapter 2, we know that in a database environment a conceptual inductive system requires the combination of an expressive representation capability with an efficient incremental process. This requirement is not satisfied by existing conceptual inductive methods. In order to overcome the drawbacks of the two previous models, we present a new inductive model called the Knowledge-Directed Model (KNOWD Model), which describes a concept by a set of logical expressions. A logical expression can be a compound feature which

combines related features as directed by domain knowledge. In this section we describe the KNOWD model, discuss concept descriptions, and present the representation formalism.

3.1.1. Description of the KNOWD Model

Let us define some symbols and terms for the new model. In the KNOWD model, P, A and X are defined to be the same as in the logic-based model. Atomic expressions have the same definition as in the logic-based model. We define E to be a set of descriptions where each description is a set of logical expressions composed of atomic expressions and logical connectives.

An **instance** d is a description denoted as $d = \{ CONJ_1, ..., CONJ_t \}$. Each element in d, called a **feature** of the instance, is a conjunction of atomic expressions. We use S to denote the set of all possible instances.

A **concept description** $D_i = \{EXP_{i1}, ..., EXP_{is}\}$ of concept C_i, $i \in \{1, ..., u\}$, is a set of logical expressions composed of atomic expressions and logical connectives. Each EXP_{ik} is called a **feature** of concept C_i. If the feature involves more than one atomic expressions, it is called a **compound feature**.

In the KNOWD model the **criterion function** H: $I \times E \rightarrow$ {**true, false, uncertain**} is defined as:

$$H(d,D_i) = \begin{cases} true, & if\ AND(d) \rightarrow AND(D_i)\ ; \\ false, & if\ for\ all\ EXP \in D_i,\ \neg[AND(d) \rightarrow EXP]\ ; \\ uncertain, & otherwise. \end{cases}$$

The resultant concept description D_i needs to satisfy two requirements: (1) By the inductive assertion, all instances of concept C_i should be correctly classified, (2) No instances of other concepts are misclassified into C_i. These two requirements are the two following conditions. With the definition of the criterion function, the completeness condition

$$[\ I_i \times \{D_i\} \subseteq H^{-1}(\mathbf{true})\ ]$$

can be specialized in the KNOWD model as follows.

Definition: The **completeness condition** in the KNOWD model for a concept description D_i, i=1,...,u, is defined as

$$(\forall d)\{[d \in I_i] \rightarrow [AND(d) \rightarrow AND(D_i)]\} \tag{3.1}$$

This condition states that for every known instance d of a concept C_i the conjunction of all the elements in the concept description D_i should be a generalization of the conjunction of all elements in d.

Then the consistency condition

$$[\ I_j \times \{D_i\} \subseteq H^{-1}(\textbf{false})\ ],\ \ i \neq j,\ j \in \{1, ..., u\}$$

can be specialized in the KNOWD model as follows[1].

Definition: The **consistency condition** in the KNOWD model for a concept description D_i, i=1,...u, is defined as: for all EXP in D_i,

$$\neg(\exists d)\{[d \in I_j] \wedge [AND(d) \rightarrow EXP]\},\quad j=1,...,u;\ \ j \neq i \tag{3.2}$$

This condition states that for any feature EXP in the concept description of C_i there should not exist an instance d of a concept C_j, $j \neq i$, such that the feature EXP is a generalization of the conjunction of all the features in the instance d.

The general model of conceptual induction can be specialized into the following KNOWD model.

---

[1] $[\ I_j \times \{D_i\} \subseteq H^{-1}(\text{false})\ ],\ \ i \neq j,\ j \in \{1, ..., u\}$
means that for all d in I_j, $H(d, D_i) = \text{false},\ \ i \neq j,\ j \in \{1, ..., u\}$
i.e., $(\forall d)\{[d \in I_j] \rightarrow [H(d, D_i) = \text{false}]\},\ \ i \neq j,\ j \in \{1, ..., u\}$
By the definition of function H, we have: for all EXP in D_i
$(\forall d)\{[d \in I_j] \rightarrow \neg[\text{AND(d)} \Rightarrow \text{EXP}]\},\ \ i \neq j,\ j \in \{1, ..., u\}$
that is $(\forall d)\{\neg[d \in I_j] \vee \neg[\text{AND(d)} \Rightarrow \text{EXP}]\},\ \ i \neq j,\ j \in \{1, ..., u\}$
Then $(\forall d)\neg\{[d \in I_j] \wedge [\text{AND(d)} \Rightarrow \text{EXP}]\},\ \ i \neq j,\ j \in \{1, ..., u\}$
i.e., $\neg(\exists d)\{[d \in I_j] \wedge [\text{AND(d)} \Rightarrow \text{EXP}]\},\ \ i \neq j,\ j \in \{1, ..., u\}$

INPUT:

1. $I = \{ d_1, ..., d_m \}$, a set of known (or observed) instances, and each instance in I is a set of features which are conjunctions of atomic expressions.
2. $U = \{ K_1, K_2, ..., K_u \}$, a set of constant symbols representing names of all the possible concepts in the universe of discourse;
3. A mapping $\rho: I \rightarrow U$;
4. A criterion function $H: I \times E \rightarrow$ {**true, false, uncertain**} defined below:

$$H(d,D_i) = \begin{cases} true, & \text{if } AND(d) \rightarrow AND(D_i) ; \\ false, & \text{if for all } EXP \in D_i, \neg[AND(d) \rightarrow EXP] ; \\ uncertain, & otherwise. \end{cases}$$

OUTPUT:

$D_1, ..., D_u$, descriptions of concepts which are sets of features, such that for all known instances the completeness condition (3.1) and the consistency condition (3.2) are satisfied.

With the concept descriptions D_i, i=1,...,u, an **inductive assertion** can be expressed as

$$(\forall d)\{[H(d,D_i)=true \wedge (\bigwedge_{j \neq i} H(d,D_j)=false)] \rightarrow (d \in C_i)\}, \quad d \in S$$

This assertion states that for every possible instance d in S, if the conjunction of all features in the concept description D_i is a generalization of the conjunction of all features in the instance and none of the features in other concept descriptions is a generalization of the conjunction of all

features in the instance d, then the instance d belongs to the concept C_i.[2] The inductive assertion can be used to define an approximation λ' of the mapping λ:

$$\lambda'(d) = \begin{cases} K_i, & \text{if } [H(d,D_i)=true] \wedge [\bigwedge_{j \neq i} (H(d,D_j)=false)] \ ; \\ \Theta, & \text{otherwise.} \end{cases}$$

where $K_i \in U$, and Θ is a reserved symbol in U denoting the class of unclassifiable instances.

The KNOWD methodology extends the mapping λ' to all instances in S. In case an instance cannot be classified by the inductive assertion, a similarity value is computed for the instance and each of the concepts. The instance will be classified into the most similar concept. In the KNOWD methodology, the computation of the similarity is based on the number of matching features, closeness of the matching features, and weights of features. The weights are determined by the importance of the attributes, relevance of components, simplicity of the expressions, completeness factors and consistency factors (defined below) of the features. Detailed discussion on the similarity will be given in subsection 3.2.6.

Completeness and consistency would be satisfied by an ideal concept description. In real world applications, these two conditions are usually too strong to be satisfied by any possible concept descriptions. Similar to the set-based model, the two conditions can be relaxed by introducing the completeness factor and consistency factor to deal with those cases which have a criterion function value of "uncertain". Let C_i, $i \in \{ 1,...,u \}$, be a concept, and let I be the set of all incorporated instances, or instances which have already been entered into the inductive system at a given time t. Let I_i be the set of incorporated instances of concept C_i, and let EXP be an expression in the concept description D_i. We define the **completeness factor** FR_1 and the **consistency factor** FR_2 of a feature EXP in the concept description D_i as follows:

---

[2] The condition in the inductive assertion could be relaxed, and the new inductive assertion would be:

$$(\forall d)\{[H(d,D_i)=true \wedge (\bigwedge_{j \neq i} H(d,D_j) \neq true)] \rightarrow (d \in C_i)\}, \qquad d \in S$$

This assertion would allow more errors.

$$FR_1(I_i,EXP)=\frac{card(\{d \mid d\in I_i \wedge AND(d)\rightarrow EXP\})}{card(I_i)}$$

$$FR_2(I_i,EXP)=\frac{card(\{d \mid d\in I_i \wedge AND(d)\rightarrow EXP\})}{card(\{d \mid d\in I \wedge AND(d)\rightarrow EXP\})}$$

where card(A) represents the cardinality of a set A. From the above we can see that if N denotes the number of incorporated instances of concept C_i at a given time t, each of which has a feature that has a generalization EXP, then $FR_1(I_i, EXP)$ is "N divided by the total number of incorporated instances of C_i" and $FR_2(I_i, EXP)$ is "N divided by the total number of incorporated instances which contain a feature that has a generalization EXP".

Theorem: A concept description D_i satisfies the completeness condition, if and only if for every EXP in D_i, $FR_1(I_i, EXP) = 1$.

Proof: Given that D_i satisfies the completeness condition, we have

$$(\forall d)\{[d \in I_i] \rightarrow [AND(d) \Rightarrow AND(D_i)]\}, \quad d \in I.$$

then for all EXP in D_i we have

$$(\forall d)\{[d\in I_i]\rightarrow[AND(d)\rightarrow EXP]\}, \quad d\in I, \qquad (c1)$$

Let $J_i = \{ d \mid [d \in I_i] \wedge [AND(d) \Rightarrow EXP] \}$

Because of $(J_i \subseteq I_i)$ and condition (c1), we have $I_i = J_i$. Therefore, $FR_1(I_i, EXP) = card(J_i)/card(I_i) = 1$.

Given that for all EXP in D_i, $FR_1(I_i, EXP) = 1$, then

because $J_i \subseteq I_i$, $card(I_i) = card(J_i)$, and J_i and I_i are finite sets, we have $I_i = J_i$.

Thus, for all EXP in D_i, if $d \in I_i$, then $d \in J_i$, and we have $AND(d) \Rightarrow EXP$.

Thus, if $d \in I_i$, then $AND(d) \Rightarrow \bigwedge_{EXP\in Di}(EXP)$, or $AND(d) \Rightarrow AND(D_i)$. Therefore, the completeness condition is satisfied. [End of Proof]

Theorem: A concept description D_i satisfies the consistency condition, if and only if for every EXP in D_i, $FR_2(I_i, EXP) = 1$.

Proof: Given that D_i satisfies the consistency condition, we have: For all EXP in D_i,

$$\neg(\exists d)\{ [d \in I_j] \wedge [AND(d) \Rightarrow EXP] \}, \quad j=1,...u; \; j \neq i; \; d \in I.$$

That is

$$(\forall d)\{ \neg[AND(d) \Rightarrow EXP] \vee \neg[d \in I_j] \}, \; j=1,...,u; \; j \neq i; \; d \in I.$$

Thus $(\forall d)\{ [AND(d) \Rightarrow EXP] \rightarrow \neg[d \in I_j] \}, \; j=1,...,u; \; j \neq i; \; d \in I.$

Since I_i is disjoint with all I_j, and $I = \bigcup_{i=1}^{u} I_i$, $\neg[d \in I_j]$, $j=1,...,u; \; j \neq i$, means $[d \in I_i]$, so we have

$$(\forall d)\{[AND(d) \Rightarrow EXP] \rightarrow [d \in I_i]\}, \quad d \in I, \tag{c2}$$

Let $J_i = \{ d \mid d \in I_i \wedge AND(d) \Rightarrow EXP \}$

and $J = \{ d \mid d \in I \wedge AND(d) \Rightarrow EXP \}$.

Because of $(J_i \subseteq J)$ and condition (c2), we have $J_i = J$. Therefore, for all EXP in D_i, $FR_2(I_i, EXP) = card(J_i)/card(J) = 1$.

Given that for all EXP in D_i, $FR_2(I_i, EXP) = 1$, we have

$$J_i = J, \tag{c3}$$

because $J_i \subseteq J$, $card(J_i) = card(J)$, and J_i and J are finite sets.

Assuming that there exists an instance d in I such that for an EXP in D_i,

$$[d \in I_j] \wedge [AND(d) \Rightarrow EXP], \quad j \neq i, \; j \in \{ 1, ..., u \},$$

then $d \in J$. But $d \notin J_i$ because I_i and I_j are disjoint and $d \notin I_i$. So $J_i \neq J$ which is a contradiction with condition (c3). Therefore, D_i satisfies the consistency condition: for all EXP in D_i,

$$\neg(\exists d)\{ [d \in I_j] \wedge [AND(d) \Rightarrow EXP] \}, \; j=1,...,u; \; j \neq i; \; d \in I$$

[End of Proof]

The completeness factor and consistency factor are important criteria to determine the quality of a concept description. With the completeness factor we can measure the characterizing capability of a concept description, that is how well the instances in this concept have been summarized by the concept description. With the consistency factor, we can measure discrimination capability of a concept description, that is how well the concept description can differentiate instances of this concept from instances of other concepts. One way of relaxing the completeness and consistency conditions for a concept description is by allowing the completeness and consistency factors of each feature of the concept description to have a value less than 1.0.

These two factors have a formal basis in probability theory; their explanations in terms of probabilities are given below. Let C_i, $i \in \{1,...,u\}$, be a concept, and let EXP be a feature in C_i's description. Let S be the sample space composed of all possible instances in the universe of discourse, $S = C_1 \cup ... \cup C_u$. Each sample point is an instance d, $d \in S$. We define A as the event that the following condition for an instance is satisfied: "the conjunction of all features in the instance description has a generalization EXP". Define B as the event that the following condition is satisfied: "The instance belongs to concept C_i". The conditional probability

$$F_1(C_i,EXP)=Prob\{A \mid B\}$$

is the probability of an instance having a description which has a generalization EXP given that the instance belongs to the concept C_i. The conditional probability

$$F_2(C_i,EXP)=Prob\{B \mid A\}$$

is the probability of an instance belonging to the concept C_i given that the instance has a description which has a generalization EXP.

Assuming every sample point is equally probable, then each sample point has a probability 1/card(S). Let S_A be the set of all possible instances which satisfy condition A; then

$$Prob\{A\} = \frac{card(S_A)}{card(S)} ,$$

$$Prob\{B\} = \frac{card(C_i)}{card(S)} ,$$

and

$$Prob\{A \wedge B\} = \frac{card(S_A \cap C_i)}{card(S)} .$$

Therefore,

$$F_1(C_i, EXP) = \frac{Prob(A \wedge B)}{Prob(B)} = \frac{card(S_A \cap C_i)}{card(C_i)} ,$$

and

$$F_2(C_i, EXP) = \frac{Prob(A \wedge B)}{Prob(A)} = \frac{card(S_A \cap C_i)}{card(S_A)} .$$

Since the sets S_A and C_i are not known, we cannot compute the above probabilities. We may use the probabilities based on the sample space of all incorporated instances at a given time t to estimate $F_1(C_i, EXP)$ and $F_2(C_i, EXP)$. We define A^t as the event in the new sample space such that the following condition is satisfied: "The conjunction of all features in the instance description has a generalization EXP". Define B^t as the event such that the following condition is satisfied: "The instance belongs to concept C_i". Then

$$F_1(C_i, EXP) \approx Prob\{A^t \mid B^t\} ,$$

and

$$F_2(C_i, EXP) \approx Prob\{B^t \mid A^t\}$$

Let I_i be the set of all incorporated instances of concept C_i at time t, and let $S_A{}^t$ be the set of all possible instances which satisfy condition A^t; then

$$F_1(C_i,EXP) \approx \frac{card(S_A^t \cap I_i)}{card(I_i)} ,$$

and

$$F_2(C_i,EXP) \approx \frac{card(S_A^t \cap I_i)}{card(S_A^t)}$$

By comparing the above estimates of F_1 and F_2 with the completeness factor FR_1 and consistency factor FR_2, we can see that FR_1 and FR_2 are the same as the above estimates of F_1 and F_2, respectively.

3.1.2. Representation Formalism

In our knowledge-directed induction model, instances of concepts and generalized concept descriptions are represented in the same language; every concept is represented by a set of expressions. To define a representation formalism is much easier than to implement it in an inductive system. Usually there are some restrictions in the implementation. In the KNOWD model, the following is the description of the expressions:

EXP ::= CONJ-EXP | EXP ∨ CONJ-EXP

CONJ-EXP ::= ATOM | ATOM ∧ CONJ-EXP

ATOM ::= [DES OP VALUE] | [REL(COMP1, ..., COMPn)]

| [DES OP VALUE COND]

COND ::= WITH ATTRIB OP VALUE

DES ::= ATTRIB | ATTRIB(COMP)

OP ::= < | > | = | <= | >= | !=

VALUE ::= CONST | FUNC | ATTRIB(COMP) | COMP | VARIABLE

where ATTRIB is an attribute in **A**, a set of symbols; VARIABLE is a variable in **X**, a set of symbols; COMP is a component[3] in **P**, a set of symbols, or is a variable in **X**; REL is a

[3] A physical object may consist of several parts called components, and a description of this type of object is composed of descriptions of its components and descriptions about the whole object.

relationship in **R**, a set of symbols; DES is a descriptor which can be either an attribute of a component or an attribute of a concept; CONST is either a single-form constant value in **C**, a set of values, or is some complex-form constant value (which is discussed later), or is a variable in **X**; OP is an operator which can be "equal", "greater than", "less than" and others as indicated, and FUNC is a function call in Lisp format (a function call returns a constant value in **C**). The function is either defined by the system or by a developer. Parentheses may be placed around expressions as needed. The syntax defines an order of precedence: Operators (OPs) are evaluated before conjunctions ($\wedge$), which are evaluated before disjunctions ($\vee$).

Our representation formalism is a variation of predicate calculus. The attributes and relationships correspond to predicates. We have

$$\mathbf{A} \cup \mathbf{R} = P \text{ , (i.e., the set of all predicates)}$$

and

$$\mathbf{P} \cup \mathbf{C} \cup \mathbf{X} = A \cup X \text{ (i.e., the set of all constants and variables).}$$

The logic connectives include conjunction ($\wedge$) and disjunction ($\vee$)[4]. The generalization is determined not only by the logical expression, but also by the values in each atomic expression. For example

$$[p(x) = a] \Rightarrow [p(x) = a \vee b].$$

This type of generalization is called **internal generalization.** The universal quantifier is represented in an implicit quantifier form by using matching variables (symbols preceded with "?", such as ?x). We just drop the ($\forall$ x) and replace every occurrence of "x" by "?x". For example, the predicate formula

($\forall$ x) [COLOR(x, black) $\vee$ COLOR(x, white)]

is represented as

---

[4] We have not included negation. The reason is that in our model concept descriptions may have a different number of features (i.e., some features may be absent) and the negation of a feature associated with an attribute may make it impossible to differentiate a description containing the feature associated with the same attribute with different values from a description without a feature associated with the attribute. For instance, assuming an attribute a has possible values 1, ..., 10, a concept description containing a feature $\neg$(a = 1) would cover all instances with (a = 2), ..., or (a = 10) and also cover instances without a feature associated with attribute a.

[COLOR (?x) = black] ∨ [COLOR (?x) = white].

There are many ways to represent the existential quantifiers. For example the predicate formula

(∃ x) [COLOR(x, yellow)]

is represented as

[Num-of-components >= 1 WITH COLOR = yellow].

Since this is a representation formalism, the attributes, components and relationships (i.e., **A, P** and **R**) are domain related, and they are specified by application developers.

Our representation language is also used to represent the deductive rules in which predicate functions can be used as an atomic expression. Combining functions in the representation language makes it easier to formulate some deductive rules. For example, in the rule

IF ((start-time (?sensor1) = ?value1)

(start-time (?sensor2) = ?value2)

(<= ?value1 ?value2))

THEN ADD ((activated-before (?sensor1 ?sensor2))

the predicate function (<= ?value1 ?value2), unlike other expressions, needs to be evaluated to the value **true** or **false**. In rule

IF ((start-time (?s) = ?t1)(end-time (?s) = ?t2))

THEN ADD ((duration (?s) = (- ?t2 ?t1)))

the function (- ?t2 ?t1) needs to be evaluated, and the returned value is presented in the expression that is to be added into the description. Some examples of ATOM are shown below

[magnitude(sensor1) = large]

"The magnitude of sensor1's value change is large."

[same-trend(s1, s2, s3)]

"s1, s2 and s3 have the same trend of change."

[first-started-sensor = sensor-i]

"sensor-i is the first activated sensor."

[No-of-sensors = 3 WITH direction = POS]

"The number of sensors that increase in value is 3."

This representation can describe a **structural domain** where entities have internal structures and each concept has a description containing features and relationships of components. Both entities and their components have attributes. Attributes can be of several types: (1) a **nominal attribute** which has an unordered value set; (2) a **linear attribute** which has a linearly ordered set; and (3) a **hierarchical attribute** which has a tree-like ordered value set. Examples of different attribute types are shown below:

The nominal attribute **direction** has a value set { POS, NEG }.

The linear attribute **No-of-sensors** has a value set { 1, 2, 3, ..., 20 }

The hierarchical attribute **curve-pattern** has a value set {pattern, changing, unchanging, one-event, two-event, three-event, multi-event, rising, falling, fall-rising, rise-falling, rsr, rfr, frf, fsf, rsf, fsr }, which has a tree-like order on it (Figure 3.1).

Relationships between attributes and components can also be represented; examples are shown below:

[rate(sensor1) = high WITH direction = POS],

which is equivalent to

[rate(sensor1) = high] $\wedge$ [direction(sensor1) = POS]

"The changing rate of sensor1 is large and the direction of change is upward."

Example of representing the relationship between components is given:

[rate(sensor1) < rate(sensor2)]

"The changing rate of sensor1 is less than that of sensor2."

A constant may be in one of the three forms: single-form, range-form and or-form (i.e., internal disjunction) as shown below. Range-forms and or-forms, which are composed of values in **C**, are extensions to representation with only single-forms. Now more types of generalizations, beside selecting or dropping conditions, can be performed.

Single-form:

[magnitude(s1) = large]

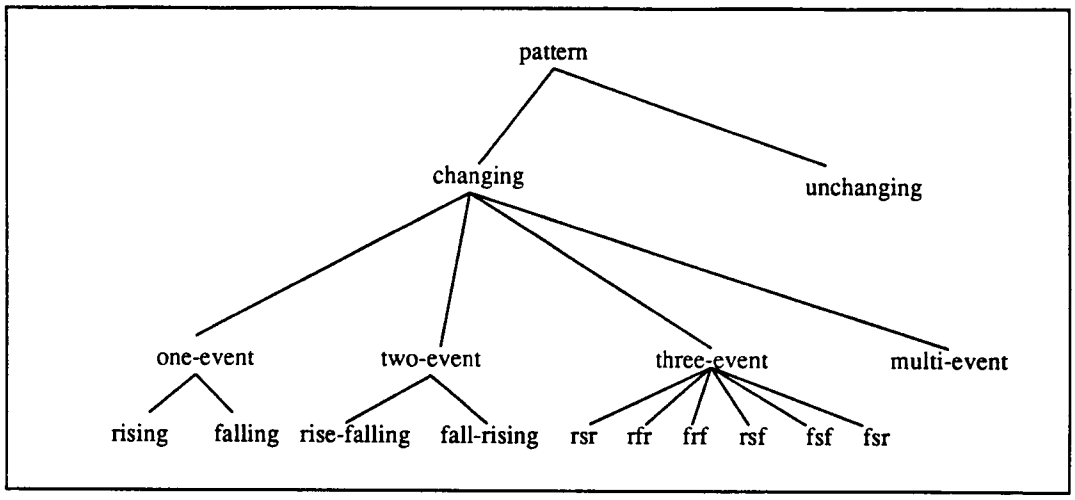

Figure 3.1. A Hierarchical Attribute.

Range-form:

[No-of-sensors = 2..4]

which is equivalent to

[No-of-sensors = 2] ∨ [No-of-sensors = 3] ∨

[No-of-sensors = 4]

Or-form:

[magnitude(s1) = large ∨ very-large]

which is equivalent to

[magnitude(s1) = very-large] ∨ [magnitude(s1) = large]

3.1.3. Concept Descriptions

A description of a concept that satisfies both the completeness and consistency conditions is called a **definitional description,** because this description gives the definition of a concept. In many domains it is too restrictive to find a definitional description satisfying both conditions, so we introduce two other types of descriptions for a concept: **characteristic description** and **discriminant description.** The following example shows that a definitional description is too restrictive for some situations. Let

$$d_{11} = \{ f_1, f_2, f_3 \}$$

and

$$d_{12} = \{ f_1, f_2, f_4 \}$$

be instance descriptions of the concept C_1, and let

$$d_{21} = \{ f_1, f_2, f_5 \}$$

and

$$d_{22} = \{ f_1, f_2, f_6 \}$$

be instance descriptions of the concept C_2. There is no feature that satisfies both completeness and consistency conditions for either concept. Therefore, both concepts have definitional descriptions with an empty set of features.

A **characteristic description** of a concept is the description that satisfies the completeness condition; all the features in it have a completeness factor value of 1.0. For instance, in the above example, both C_1 and C_2 have a characteristic description $\{f_1, f_2\}$. A **discriminant description** of a concept is a description that satisfies the consistency condition; all the features in it have a consistency factor value of 1.0. For instance, in the above example, C_1 has a discriminant description $\{f_3, f_4\}$, and C_2 has a discriminant description $\{f_5, f_6\}$. The two types of descriptions have different uses. When given an instance of a specified concept, we may predict the features of the instance by the characteristic descriptions of the concept. When given an unknown instance, we may identify which concept it belongs to by the discriminant descriptions of the concept.

There are some concept descriptions which do not describe the features of individual instances; instead, they describe general features of a concept which are aggregated from the entire set of known instances. We call this kind of description an **aggregate description.** Examples of aggregate descriptions may contain features such as the average, maximal or minimal values for some attributes describing the whole set of known instances.

3.1.4. Discussion and Comparison

Another type of relaxation of our consistency condition with the least restriction is described below.

For every known instance d of a concept C_j, there exists a logical expression EXP_m in the description D_i of a concept C_i, $i \neq j$, such that EXP_m is not a generalization of any conjunction $CONJ_n$ in d.

We do not employ this definition by the reasons given below. The least restrictive definition of consistency does not enforce any consistency requirement on most features in a concept description. A concept description that contains a large number of non-discriminant features (features with low consistency factors) is not of high quality. With this less restrictive definition, many features may be included in a concept description no matter how low their consistency factors are, as long as there is a single feature which may have a high consistency factor that has been included in the concept description. For example, let

$$d_{11} = \{ f_1, f_2, f_3, f_5 \}$$

and

$$d_{12} = \{ f_1, f_2, f_3, f_6 \}$$

be instance descriptions of concept C_1, and let

$$d_{21} = \{ f_1, f_2, f_4, f_7 \}$$

and

$$d_{22} = \{ f_1, f_2, f_4, f_8 \}$$

be instance descriptions of concept C_2. With the new definition of consistency, the description of C_1 is

$$D_1 = \{ f_1, f_2, f_3 \},$$

and the description of C_2 is

$$D_2 = \{ f_1, f_2, f_4 \}.$$

Now, we can see that the features f_1 and f_2 which have low consistency factors are included in both concept descriptions only because there is a feature f_3 (or f_4) which has a high consistency factor. In the least restrictive definition, even the logical expression EXP_m of the concept description D_i may not have a high consistency factor, because it only discriminates instances of

the concept C_j from the concept C_i, and it may not discriminate instances of concepts C_k ($k \neq j$, $k \neq i$) from the concept C_i.

The KNOWD model is more expressive than the set-based model. For example, structured concepts can be represented, and generalization of internal disjunctions can be performed in the KNOWD model. Also, features which cannot be generated in the set-based model can be generated in the KNOWD model as shown in the following example.

Example: In a set-based model, let F = { A, B, C, D } be a set of features each of which is an attribute-value pair. Let

d_{11} = { A, B, C } and d_{12} = { A, B, D } be instance descriptions of concept C_1; let d_{21} = { A, C, D } and d_{22} = { B, C, D } be instance descriptions of concept C_2. The characteristic descriptions of C_1 are { A, B }, and the characteristic descriptions of C_2 are { C, D }. Let dd_1 be the discriminant description of C_1. Since $dd_1 \subseteq F$, $dd_1 \cap F = dd_1$. By the consistency condition $dd_1 \cap d_{21} = \varnothing$ and $dd_1 \cap d_{22} = \varnothing$, so $dd_1 \cap (d_{21} \cup d_{22}) = \varnothing$; since $d_{21} \cup d_{22} = F$, $dd_1 \cap F = \varnothing$, that is $dd_1 = \varnothing$. Therefore, there are no discriminant descriptions for C_1 and C_2.

In the KNOWD model, suppose C_1 and C_2 have the same features as above, A and B have a logical AND relationship, and C and D have a logical AND relationship. Now d_{11} = { $A \wedge B$, C }, d_{12} = { $A \wedge B$, D }, d_{21} = {A, $C \wedge D$ }, and d_{22} = { B, $C \wedge D$ }. C_1 has a discriminant description { $A \wedge B$ }, and C_2 has a discriminant description { $C \wedge D$ }. Since a set-based model cannot represent logic relationships of features, this kind of description cannot be generated. [End of example]

In the following example, we can see that some relationships between features cannot be processed by the set-based model but can be processed in the KNOWD model. Similar examples can be found in situations where exclusive-or relationships of features are used for describing a concept.

Example: In a set-based model, descriptions of two instances are given:

d1 = { ($attrib_1$, val_1), ($attrib_2$, val_3), ($attrib_3$, val_4) } , and

d2 = { ($attrib_1$, val_2), ($attrib_2$, val_3), ($attrib_3$, val_4) } .

They can be generalized as

$G = \{ (attrib_2, val_3), (attrib_3, val_4) \}$.

But if in a real world situation we think that attribute $attrib_2$ can have meaning only when it is conjoined with the attribute $attrib_1$, then generalization G is not correct.

In the KNOWD model, the descriptions can be represented as:

$d1 = \{ [(attrib_1, val_1) \wedge (attrib_2, val_3)], (attrib_3, val_4) \}$,

$d2 = \{ [(attrib_1, val_2) \wedge (attrib_2, val_3)], (attrib_3, val_4) \}$.

They can be generated as

$G = \{ [attrib_3, val_4] \}$.

Now the meaningless description is avoided. [End of example]

Both the set-based model and logic-based model are extreme cases of the KNOWD model. If we restrict all logical expressions in every d and D_i in the KNOWD model to be of the form $p_i(v)$, $p_i \in P$, then we can equate $p_i(v)$ with an attribute-value pair (p_i, v). With this restriction, the completeness and consistency conditions in the KNOWD model can be transformed into the conditions in the set-based model. Thus, the KNOWD model can be applied to situations for a set-based model. If we restrict D_i and d to be sets with a single element, then the completeness condition (3.1) is essentially (2.1) and the consistency condition (3.2) is essentially (2.2). Therefore, the KNOWD model can also be applied to situations appropriate for a logic-based model. From the preceding we can see that the KNOWD model can be applied to a larger range of domain problems than the previous two models.

In the KNOWD model a relatively smaller hypothesis space (that is the set of all possible concept descriptions) needs to be searched. For a given domain problem, let N be the number of components, M be the average number of attributes for each component, and K be the number of the global attributes (attributes not about components but about a whole instance) of the concepts. Assume only conjunctions are used, and attributes are of the binary type (having only two possible

values). Then there are (N·M + K) different features. In the logic-based model the total number of possible hypotheses would be

$$\sum_{i=1}^{N\cdot M+K}\binom{N\cdot M+K}{i}=2^{N\cdot M+K}-1$$

For a simple domain problem, assume N = 10, M = 5, and K = 10. In the logic-based model the total number of possible hypotheses will be $(2^{60} - 1) \approx 10^{18}$. In the KNOWD model, if the average number of features in a compound feature is 2, then there are 60/2 = 30 different features and the total number of hypotheses will be about $2^{30} - 1 \approx 10^9$. With concept constraints, the number of hypotheses would be further reduced. In the following example, we show that a much larger description space needs to be searched by the logic-based model than the KNOWD model.

Example: In a logic-based model, let f_1, ..., f_5 be features. Suppose only conjunctive generalizations (not including internal generalizations and constructive generalizations[5]) are considered. For the given instance description:

$$DES = f_1 \wedge f_2 \wedge f_3 \wedge f_4 \wedge f_5$$

the possible generalizations would include all conjunctions of different combinations of the features, such as

$$f_1,\ f_1 \wedge f_2,\ f_1 \wedge f_3,\ f_1 \wedge f_2 \wedge f_3,\ f_1 \wedge f_2 \wedge f_3 \wedge f_4,\ f_4 \wedge f_5,\ f_2,\ f_2 \wedge f_3, \ldots$$

The total number of generalizations is $2^5 - 1 = 31$.

In KNOWD model, suppose f_1 is related to f_2, and f_4 is related to f_5. The description of the example can be represented as:

$$DES = \{\ f_1 \wedge f_2,\ f_3,\ f_4 \wedge f_5\ \}$$

The possible generalizations would be:

$\{\ f_1 \wedge f_2\ \}$, $\{\ f_1 \wedge f_2,\ f_3\ \}$,

$\{\ f_1 \wedge f_2,\ f_4 \wedge f_5\ \}$, $\{\ f_3\ \}$,

$\{\ f_3,\ f_4 \wedge f_5\ \}$, $\{\ f_4 \wedge f_5\ \}$,

[5] Constructive generalizations will generate vocabulary for new attributes and relationships.

{ $f_1 \wedge f_2$, f_3, $f_4 \wedge f_5$ }

The total number of generalizations is $2^3 - 1 = 7$. [End of example]

The logic-based model cannot easily account for worth (or importance) of features in conceptual induction. Since attributes are of different importance and components are of different relevance measures, different features should have different worth values. For example, given descriptions of three instances:

d_1 = (color = red)(size = big)(type = apple),

d_2 = (color = red)(size = big)(type = house), and

d_3 = (color = green)(size = small)(type = apple),

a model that regards all the attributes as equally important and classifies instances based on the number of common features may group the instance with d_1 and the instance with d_2 as one class. In fact, background knowledge can be used to assign different importance values to the different attributes, so that "type" is assigned with a higher value, and a better classification can be found which groups the instance with d_1 and the instance with d_3 as one class.

In the KNOWD model the number of hypotheses can be reduced by the use of application-specific knowledge. Meaningless conjunctions can be avoided. The logical relationships of features are determined by domain knowledge. This model combines the advantages of the set-based model and the logic-based model. Different features in the KNOWD model are considered to be independent of each other, so that each feature can be generalized, specialized or deleted during the incremental inductive process. Therefore, the search space is much restricted at each intermediate inductive step. This makes incremental induction much more efficient in the KNOWD model than the logic-based model. Values of confidence can be assigned to the features, making it possible to perform induction on noisy data.

In summary, the KNOWD model has the following characteristics which are not shared by other models: (1) it represents a description of an instance or a concept by a set of logical expressions; (2) it uses domain knowledge to determine the relationships among features; and

(3) it supports the combination of an expressive representation with an efficient incremental process of induction.

3.1.5. The Underlying Database Model

In order to integrate the KNOWD induction model with a database system, we build the Generalization Data Model (GDM) to support induction and concept description retrieval. In this subsection we describe GDM, the query language, and the advantages of GDM.

In GDM objects are constructs used to encapsulate data and operation. Objects are used to model real world entities and are divided into instance-objects, concept-objects and cluster-objects. Concepts can be described by different levels of abstractions. **Instance-objects** represent individual instances of concepts, **concept-objects** represent basic concepts, and **cluster-objects** represent high-level concepts. **Basic concepts** are concepts known in advance into which input instances have already been classified, and **high-level concepts** or **clusters** are union of several related basic concepts. The description of a high-level concept characterizes a class of basic concepts.

Objects have **relationships** among them. In GDM inductive semantics exist as abstraction relationships between objects. GDM has (1) built-in abstraction relationships including transformation, aggregation, and generalization (refer to subsection 1.3.4); and (2) predefined object types of different levels of abstraction. During processing, abstraction relationships can be regarded as operations which map lower-level objects to higher-level objects. Transformation and aggregation are applied to a single instance-object so that higher levels of descriptions are obtained. Deduction is applied to an instance-object or a concept-object so that new features are obtained and constraints are enforced. Generalization is applied to a group of instance-objects so that characteristic and discriminant descriptions are obtained. Generalization is the most important relationship among objects, and it is also the most important to conceptualize and understand the real world. Generalization connects specific observations to a general concept; this allows us to make predictions about future situations.

In GDM a database DB can be formally represented as a 5-tuple:

$$DB = (T, O, P, R, M)$$

where

T is a set of object types $\{ T_1, T_2, ..., T_n \}$.

O is a set of objects $\{ Obj_1, Obj_2, ..., Obj_m \}$ where each object contains a concept description or an instance description.

P: $O \rightarrow T$, is a mapping from each object to an object type.

R is a set of relationships $\{ r_1, r_2, ..., r_s \}$ among objects. R contains built-in abstraction relationships: $R_T \cup R_A \cup R_D \cup R_G \subseteq R$, where R_T is the set of transformation relationships, R_A is the set of aggregation relationships, R_D is the set of deduction relationships, and R_G is the set of generalization relationships. R may also contain other relationships defined by the user.

M is a set of all methods associated with the object types. Let M_R be a set of all methods that implement the relationships, then $M = \bigcup_i M_{Ti}$, and $M \supseteq M_R$, where M_{Ti} is a set of methods associated with the object type T_i.

Relationships among objects are implemented by methods in the object types and established through a system command:

(build-relationship Obj1 Obj2 R_i)

If object Obj_1 has a relationship R_i to Obj_2, the methods associated with the type $P(Obj_1)$ that implement the semantics of R_i will be activated whenever there is a change in the descriptions in Obj_1. The activated methods will use the descriptions in Obj_1 to cause proper changes in the descriptions of Obj_2.

The generalization methods employ generalization rules which are system defined rules. Most other rules (including transformation rules, aggregation rules and deduction rules) are domain related. GDM has a set of predefined object types and methods defining relationships, which are shown in Figure 3.2. Preprocessed data, qualitative data, and concept instances are represented

by instance-objects which have abstraction relationships to other instance-objects or concept-objects. The higher-level objects have more general and abstract descriptions. The data accessing operations on each object are defined by the associated methods on the object. Each basic concept is assigned a preprocessor which transforms the particular forms of input raw data into the uniform data acceptable to the system. Developers can easily incorporate the above system-defined object types into their own object types; then, the inductive semantics are inherited by the user's object types.

In a conventional database, only attributes of stored instances can be queried. Deductive descriptions and generalization descriptions of concepts cannot be formed from the instances. However, GDM provides users with the capability to query and view concepts' descriptions on many levels, so that different users may be satisfied with different abstract levels of data.

The following describes the query language:

(cha-description concept) gives the characteristic descriptions of **concept.**

(dis-description concept) gives the discriminant descriptions of **concept.**

(agg-description concept) gives the aggregation descriptions of **concept.**

(all-concept root) gives the list of all the basic concepts in the hierarchy rooted at **root** which is a cluster object.

(sup-concept concept) gives the parent of **concept** in the concept hierarchy.

(sub-concept concept) gives the children of **concept** in the concept hierarchy.

(hierarchy) gives the hierarchy of the concepts.

(description object) gives the descriptions of **object,** which can be an instance, a concept or a cluster.

(find-concept :with <description>) gives the list of concepts that have features described by **<description>,** this is the conditional query and in the **<description>** matching variables can be used to facilitate the queries. For example, in the engine fault test analysis domain, the query

(find-concept :with [RATE (SENSOR1) = HIGH])

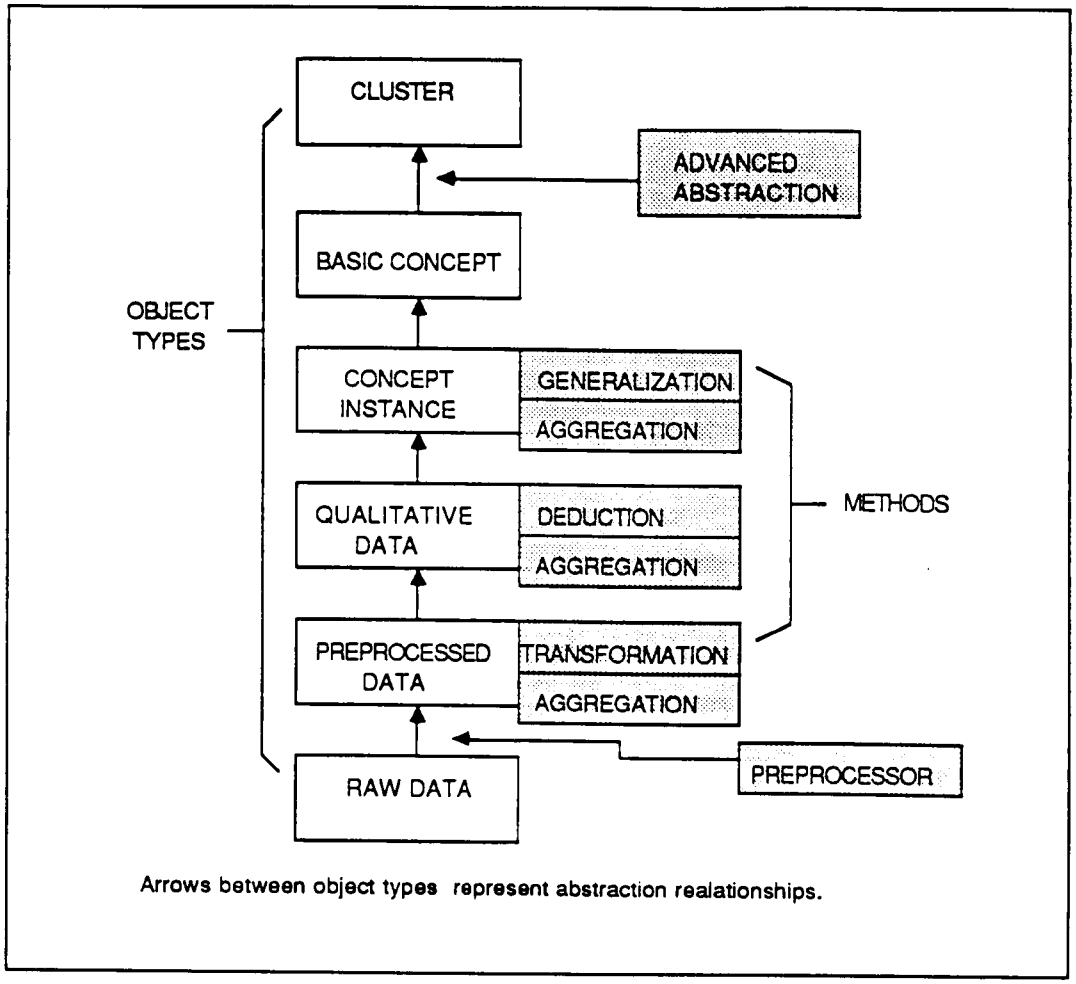

Figure 3.2. System-defined Object Types, Relationships and Associated Methods

will give the list of all concepts (engine faults) that have the feature [RATE (SENSOR1) = HIGH]. The result is a list of engine faults: (FAULT1, FAULT3, FAULT4).

(feature concept :about attrib) gives all the features related to **attrib** of **concept**. For example, the query

(feature FAULT1 :about RATE)

will give a list of features of FAULT1 about the attribute RATE: [RATE (SENSOR1) = HIGH], [RATE (SENSOR2) = LOW], [RATE (SENSOR58) = HIGH]

(**feature concept :about component**) gives all the features related to **component** of **concept.**

(**attrib-relation concept attrib-1 attrib-2**) gives the features related to **attrib-1** and **attrib-2** on all components of **concept.**

(**comp-relation concept comp-1 comp-2**) gives the features related to **comp-1** and **comp-2** on all attributes of **concept.**

In addition to the query language, users can view the descriptions and relationships of all the objects by navigating over the concept hierarchy through the interactive window. The user is able to perform the following operations:

view-hierarchy

view-cluster

view-concept

view-instance

view-parent

view-children

By utilizing object-oriented techniques and providing constructs that closely parallel the concepts and relationships typically arising in applications, GDM has the following advantages:

Modeling efficiency --- GDM provides built-in abstraction relationships. Model-component sharing supports inheritance of data types and operations on them, and information of related concepts can be accessed directly without using lower-level operations. Users can easily incorporate the GDM predefined object types and object relationships into their own object types.

Modeling flexibility --- GDM allows the user to model and view the data on many levels of generalization. Many relationships of objects can be represented; the addition of new classes

and operations is easy; changing the object representation has no impact on operations that access the object, and changing an operation's implementation does not affect applications that invoke it.

Integrity maintenance --- Data integrity is a requirement that ensures (to a certain extent) that data in the database is accurate and valid. Integrity constraints can be applied to basic concepts and high-level concepts. Typical constraints may specify attribute or component relationships for a concept, or limit value ranges and types of attributes. GDM provides facilities for defining integrity constraints on a high level; sophisticated deduction and domain theory, such as physical laws, can be employed to maintain integrity.

For inductive inference the generalization hierarchy, which is very compatible with the object-oriented model, is employed to organize objects. Specific facts form the lowest level of objects, whereas more general descriptions are expressed at higher levels. The hierarchy is organized on conceptual generalities. In the domain of rocket engine test analysis, for example, the components, sensors, faults, their properties, and their relationships can be easily captured by an object-oriented model. Generalization of knowledge from many specific tests is represented in the hierarchy organization of this model. More detailed discussion about modeling the domain of rocket engine test analysis is given in Chapter 4.

3.2. The Knowledge-Directed Methodology

In this section, we describe the KNOWD methodology and its algorithms. Subsection 3.2.1 presents a formal description of the KNOWD methodology. Subsection 3.2.2 describes the general algorithm. Subsection 3.2.3 describes the knowledge-directed method. Subsections 3.2.4 and 3.2.5 present details of the methods and algorithms. In subsection 3.2.6, we discuss the properties of the induction algorithms. Finally, we describe the structure and functionality of the inductive database system in subsection 3.2.7.

3.2.1. A Formal Description of the KNOWD Methodology

In this subsection we give the formalization of the KNOWD conceptual induction methodology. This methodology combines basic abstraction and advanced abstraction. **Basic abstraction** is a type of supervised conceptual induction which generates descriptions of **basic concepts** that are concepts into which the input instances are preclassified. **Advanced abstraction** is a type of unsupervised conceptual induction which finds high-level concepts. A **High-level concept** or a **cluster** is a union of basic concepts. The problem of conceptual induction the KNOWD methodology is designed to solve is pictured as in Figure 3.3.

We define **basic concepts** as the same as the definition of concept. The purpose to use the term "basic" is to differentiate them from high-level concepts that represent clusters. Let $MF(D_1, D_2)$, the **matching factor** of two concepts with descriptions D_1 and D_2, be a function which describes the similarity of matched features in the two descriptions. Matched features are features with the same attribute and component. Let CCEM, the **conceptual clustering evaluation measure**, be a function that measures the quality of clustering when incorporating a new basic concept C_b into a given cluster (detailed discussion of MF and CCEM will be given in section 3.2.6).

Let I_i, i = 1, ..., u, be a set of known instances of a basic concept C_i, and let domain knowledge DK be a set of constructive rules and constraint rules. Let D_i be a description of C_i, f be a feature in D_i and P1 be a preference function defined as:

$P1(D_i, f)$ = **true** if $FR_1(I_i,f)=1$ and $FR_2(I_i,f)\geq$CN-TH;

$P1(D_i, f)$ = **false** otherwise

where FR_1 is the completeness factor, FR_2 is the consistency factor, and CN-TH is a system-defined value called the consistency threshold. Let P2 be a preference function defined as:

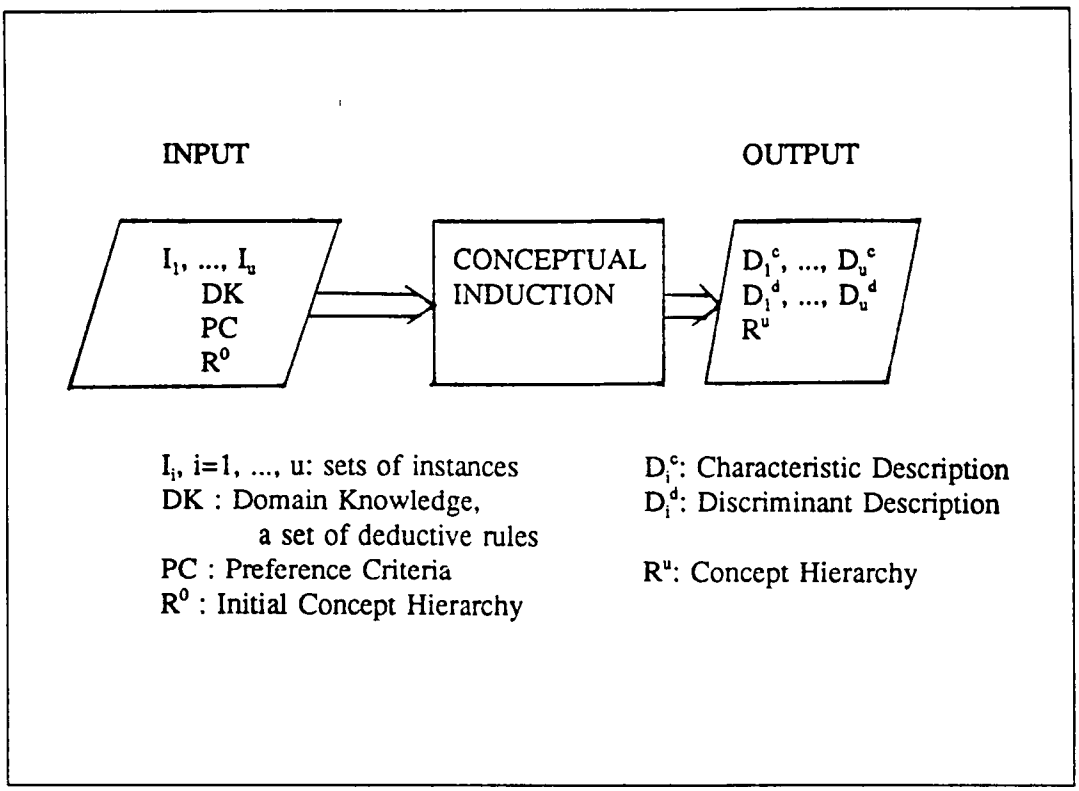

Figure 3.3. The Problem of Conceptual Induction

$P2(D_i, f)$ = **true** if $FR_1(I_i,f) \geq$ CM-TH and $FR_2(I_i,f)=1$;

$P2(D_i, f)$ = **false** otherwise

where CM-TH is a system defined value called the completeness threshold.

Definition: A **concept node** is a 3-tuple:

$$Q_j = (D_j^c, D_j^d, I_j), \quad j=1,...,u$$

where D_j^c is the characteristic description, D_j^d is the discriminant description, and I_j is the set of known instances of concept C_j.

Definition: A **cluster node** is a pair:

$$G_i = (D_i, S_i), \quad i=1,...,v$$

where D_i is the cluster description, and S_i is a set of cluster nodes and concept nodes. Nodes in S_i are called **children** of G_i, and G_i is their **parent**.

Definition: A **concept hierarchy** is a cluster node that is not a child of other node such that every contained node has one and only one parent.

Basic abstraction is a process to generate concept descriptions that can be described as two functions h1 and h2:

$$D_i^c = h1(I_i)$$

$$D_i^d = h2(I_i)$$

The function h1 generalizes instances in I_i into D_i^c which is a set of features such that for each feature f in D_i^c, $P1(D_i^c,f)$ = **true,** for all i = 1, ..., u, and f complies with the constraint rules in DK. The function h2 generates unique features of I_i such that for each feature f in D_i^d, $P2(D_i^d,f)$ = **true,** for all i = 1, ..., u, and f complies with the constraint rules in DK.

Let the initial concept hierarchy be the node

$$R^0 = (D^0, S^0),$$

where D^0 is a description of the cluster represented by node R^0, and S^0 is a set of child nodes. Initially, $D^0 = \varnothing$, and $S^0 = \varnothing$. Let the current concept hierarchy be

$$R^i = (D^i, S^i), \; i=1,...,u.$$

A given new basic concept C_i is represented by the node

$$Q_i = (D_i^c, D_i^d, I_i), \quad i=1,...,u,$$

Advanced abstraction is a process to recursively incorporate basic concepts into the hierarchy, and is defined as a recursive function H:

$$R^i = H(R^{i-1}, Q_i), \quad i=1,...,u$$

H is defined as

$$H(R^{i-1}, Q_i) = (M(D^{i-1}, D_i^c), \; E(S^{i-1}, Q_i) \;).$$

where M is a function that computes D^i, the description of the cluster, and E is a function that computes S^i, the children of R^i.

The function M is defined below:

$$M(D^{i-1}, D_i^c) = A^{i-1} \cup B^{i-1},$$

where A^{i-1} is a set of features from D_i^c which do not have matching features in D^{i-1}, and B^{i-1} is a set of features which are either

1) a generalization of a feature g in D^{i-1} to cover a feature h in D_i^c, with a higher confidence[1] than g, if h is supporting g; or

2) a feature from D^{i-1} with its confidence decreased, if a conflicting feature[2] h is found in D_i^c, and the decreased confidence is still greater than FR-TH, the feature retaining threshold.

The function E is designed to compute S^i, the set of child nodes of R^i. Let $S^{i-1} = \{N_1, ..., N_p\}$, and define the function E recursively as

$$E(S^{i-1},Q_i)= \begin{cases} S^{i-1}\cup\{Q_i\}, & \text{if } MF(D_{N_j},D_i^c)\le C\text{-}TH \text{ for all } j\in\{1,...,p\} \\ (S^{i-1}-\{N_m\})\cup\{N_m'\}, & \text{if } CCEM_m=\max_{1\le j\le p}\{CCEM_j\},\ 1\le m\le p, \\ & \text{where } N_m' \text{ is defined below.} \end{cases}$$

where C-TH is a numeric parameter value called the closeness threshold defined by the system, and

$$N_m' = \begin{cases} (M(D_{N_m},D_i^c),E(S_{N_m},Q_i)), & \text{if } N_m=(D_{N_m},S_{N_m}) \text{ is a cluster node;} \\ (D_{N_m}',S_{N_m}'), & \text{if } N_m=(D_{N_m}^c,D_{N_m}^d,S_{N_m}) \text{ is a concept node,} \\ & \text{where } (D_{N_m}^c;D_i^c \rightarrow_G D_{N_m}') \text{ and } S_{N_m}'=\{N_m,Q_i\}. \end{cases}$$

Some explanation is given below:

The function $E(S^{i-1}, Q_i) = S^{i-1}\cup\{Q_i\}$ if none of the matching factors of the new concept represented by node Q_i and concepts represented by nodes in S^{i-1} is greater than the closeness

---

[1] Confidence is a numeric value associated to a feature to measure the quality of the feature in the description.

[2] Two features are conflicting, if they have different values, and the values are far away from each other.

threshold. Otherwise, the function $E(S^{i-1}, Q_i) = (S^{i-1} - \{N_m\}) \cup \{N_m'\}$, that is, node N_m in S^{i-1} is replaced by a new node N_m', if N_m has the maximal CCEM with Q_i among all nodes in S^{i-1}.

In computing N_m', the function E is recursively used if N_m is a cluster node. If N_m is a basic concept node, a new cluster node (D'_{N_m}, S'_{N_m}) is created, where D'_{N_m} is a generalization of $D^c_{N_m}$ and D^c_i, and S'_{N_m} is a set of two basic concept nodes $\{N_m, Q_i\}$.

3.2.2. The General Algorithm

In the KNOWD methodology the paradigm for incremental inductive inference can be formulated as follows:

Before induction we have:

(1) A hierarchy of nodes with leaf nodes representing basic concepts, internal nodes representing clusters, and descriptions of concepts at each node (the hierarchy may be null, or may be created from a previous induction process);

(2) Knowledge bases which include knowledge about attributes, components, relationships and basic concepts, and also include deductive rules, generalization rules, transformation rules and aggregation rules;

(3) A raw instance d_r of a basic concept, where $d_r = \{A_1, ..., A_m\}$ is a raw description of an instance. $A_1,..., A_m$ are features in the form of atomic expressions in our representation language of preprocessed data with raw values.

After induction we get:

An extended concept hierarchy such that either a new basic concept node is added or an old basic concept's descriptions are modified to cover the new instance; the structure of the hierarchy and the cluster node's descriptions are modified to incorporate the new instance.

Let C be a basic concept, and let d_r be a raw description of a new instance of C. The following is the outline of the general algorithm of the KNOWD methodology:

(1) Read in d_r, the raw description of an instance of concept C.

(2) Basic abstraction:

a. Apply a simple transformation on d_r, generating a transformed description d_t, where d_t is a set of features in the form of atomic expressions with each raw value transformed into higher-level forms, such as symbols or category labels.

b. Apply a deductive transformation and component-level aggregation on d_t, generating an instance description d. New attributes, compound features and relationships are generated in this step.

c. Generalize concept C's characteristic descriptions to be satisfied by d.

d. Generalize concept C's discriminant descriptions and specialize (or eliminate) discriminant descriptions of other concepts.

e. Modify the concept-level aggregation descriptions.

(3) Advanced abstraction:

a. Generalize the descriptions of the clusters above the basic concept. The most frequent operation is adding, deleting, generalizing, or changing the weight of features in the descriptions of the clusters.

b. Modify the generalization hierarchy above the basic concept level. The operations include incorporating the basic concept C (if it is a new one) into a cluster, creating new clusters and deleting unqualified clusters.

The algorithm presented here describes the processing of a single instance. As an incremental method, the above algorithm can be repeatedly applied to many incoming instances. The details of the algorithm will be discussed in later sections.

3.2.3. Knowledge-Directed Induction

3.2.3.1. Knowledge and its Representation

An inductive database system needs to utilize knowledge to facilitate the induction process. The knowledge needed includes application-domain related knowledge and general

inductive heuristics. The knowledge employed by an inductive database can reduce the search space of induction and generate useful concept descriptions. The KNOWD system utilizes objects (described in Section 3.1) as the basic constructs for knowledge representation. This subsection describes various kinds of knowledge utilized by the KNOWD method and the representation of those kinds of knowledge.

There are **structural concepts** in many application domains in which the instances of a concept have components, and the descriptions of the components constitute part of the instances' descriptions. For example, in a rocket engine fault analysis domain, different faults correspond to the basic concepts, and sensors correspond to the components. The description of a fault consists of descriptions of many sensors' behavior. Domain knowledge is required to identify components whose features can characterize a concept.

Different components have different relevancies to different concepts. For example, in a rocket engine some sensors are directly related to a fault, some sensors are occasionally affected by the fault, whereas other sensors have nothing to do with the fault. Therefore, when generalizing a concept description, we want to include descriptions of components with high relevance and delete those descriptions of unrelated components. Domain knowledge about the component-concept relevance is useful for improving the performance of induction.

For a real world entity, potentially there are an infinite number of attributes to describe it. For example, a specific dog may have color, weight, height, length of the body, length of the tail, number of teeth, birthday, and birth place. But many attributes are not relevant to the description of the general concept of a dog. Selecting the right set of attributes to describe concepts requires much domain-related knowledge.

Some attributes are determined before the induction process, and other attributes are generated during the induction process. Attributes determined before induction are used to form the preprocessed data for describing instances of concepts. Those initial attributes are abstracted

directly from the raw data. For describing concepts, the initial set of attributes may not be sufficient, and further attributes may need to be generated using domain knowledge.

Attributes can be global or local. **Global attributes** describe a concept as a whole while the **local attributes** represent properties of components of a concept. Attributes can be of different types: nominal, linear and hierarchical. Different attributes have different transformation rules which transform the conceptually lower-level form of data into a higher-level form. Domain knowledge about attribute types, attribute value domains, relative importance of attribute, and transformation rules is needed for the induction process.

Relationships between attributes and between components are sometimes even more important than attributes for describing concepts. Relationships are considered compound features and are usually represented as conjunctions of atomic expressions. New descriptors can also be generated to represent relationships. Two kinds of the relationships used in our system are discussed here: attribute-attribute relationships (AA-relationships) and component-component relationships (CC-relationships). AA-relationships represent the relations between two attributes of the same component. For example, the **rate** and **magnitude** of a sensor may be combined as a compound feature:

[rate(sensor1) = v1 WITH magnitude = v2].

CC-relationships combine the same attributes of two different components. For example, in the engine fault analysis domain, CC-relationships may include: temporal relationships and direction-changing relationships of sensors. For example, two sensors may have a relationship:

[increase-together (sensor1 sensor2)]

Based on domain knowledge, users may suggest potential AA-relationships and CC-relationships. Deductive rules, which are automatically generated based on the suggestions by the system, are used to find relationships for each instance. Then the induction system tries to find any existing relationships for concepts.

Deductive rules can be divided into constructive rules and constraint rules. They are application domain related and defined by users with domain knowledge. Deductive rules are

represented by objects. Constructive rules are used to generate new terms for better descriptions of concepts. Constraint rules are used to find inconsistency of instances for avoiding incorrect generalization. Further discussion will be given in the next subsection.

Previously described knowledge is the knowledge that is used to facilitate and perform induction. Concepts and clusters are knowledge that is the result of the induction. In the incremental induction process, concepts and clusters are also used to affect further induction. In the KNOWD system, concepts are divided into two kinds: basic concepts and high-level concepts. Basic concepts are represented by concept objects, whereas high-level concepts are represented by cluster objects.

3.2.3.2. Deductive Supports to the Induction

Explanation-Based Learning (EBL) is a knowledge-intensive learning method which uses deductive inference to generate justified generalizations from a single training instance. EBL has its limitations because it is an analytic method and requires that the domain theory be perfect, but in many real world problems, only incomplete or inconsistent domain theories are available. A database system is usually a data-intensive environment, in which a complete theory to explain the instances is usually not available, and a generalization has to consider a large number of instances. However, the deductive method can be used to improve the performance of induction since new features can be derived, irrelevant features can be removed and erroneous data may be discarded.

In most domains, users have some knowledge about the domain, and the domain knowledge is not complete. So we do not try to generate a complete explanation about all the instances in the database. Instead, the KNOWD method utilizes domain knowledge in the following three aspects:

(1) deducing new features that are not directly implied by the input instances;

(2) discarding erroneous instances, which are identified by concept constraints; and

(3) influencing the inductive search by specifying interesting relationships of attributes or components.

In the following we discuss the deduction process, constructive rules for generating new vocabulary, and constraint rules for constraining concept descriptions with known knowledge. The deductive mechanism in the KNOWD methodology is given in Appendix A.

(A) Formal Description of Deduction

Now, let us formally explain the execution of a deductive rule. Let P be the if-condition part of a deductive rule, and Q be the then-part of the deductive rule, where both P and Q are conjunctions of features, so a deductive rule: "IF P THEN Q" can be logically represented as:

$$P \rightarrow Q$$

where the sign "$\rightarrow$" represents logical implication.

Let B be the Known-Fact Base (KFB) which is a set of features for a given situation, then $B \subseteq F$, where F is the set of all possible features. We assume each element s in B is true for the given situation.

A substitution σ has the form $\{(v_1, c_1), ..., (v_n, c_n)\}$, where each v_i is a matching variable and each c_i is a constant. For a set of features P, $\sigma(P)$ denotes applying σ to P, which results in a new set of features R formed by replacing each variable in P which is contained in the substitution σ by its associated constant.

Execution of a deductive rule $P \rightarrow Q$ on a given KFB is explained as the following process:

(1) find a set of substitutions S by matching P with features in KFB;

(2) for each $\sigma \in S$, if $\sigma(P) \subseteq B$, then add the features of $\sigma(Q)$ into B.

The rule means if features in $\sigma(P)$ are assumed to be true in a given situation, then features in $\sigma(Q)$ are also true in that situation.

The soundness of the derived feature statements depends upon the soundness of the deductive rules and the correctness of input facts. The deductive rules are application related and are supplied by domain experts. The system does not have the capability to confirm the soundness of deductive rules. As for the correctness of input facts, domain knowledge can be employed as concept constraints to identify erroneous input.

(B) Constructive Rules for Generating New Vocabulary

The initial vocabulary (including attributes, relationships and components) used to express input instances may not be sufficient or adequate for the concept descriptions or may be in a lower level form which is not appropriate for the user to utilize. In certain situations an important feature cannot be represented without new terms. Many methods of conceptual induction use only selective generalization rules which means that the resulting concept descriptions contain only terms from the input instances. These methods have limited capabilities to find useful concept descriptions. Thus, it is important for an inductive system to be able to create or construct new terms for attribute and relationships. Those rules for generating new features are called constructive rules. Some examples of constructive rules are shown below:

CONS-RULE1:

IF ((direction (x) = ?v)(direction (y) = ?v))

THEN ADD (same-trend (x y))

where x and y are components. This rule will add the new feature (same-trend (x y)) into the instance description.

CONS-RULE2:

IF ((start-time (x) = ?t1)(end-time (x) = ?t2)

(start-time (y) = ?t1)(end-time (y) = ?t2))

THEN ADD (concurrent-change (x y))

where x and y are components. This rule will add the new feature (concurrent-change (x y)) into the instance description.

CONS-RULE3:

IF rel(a, b), rel(b, c), ..., rel(e, f), transitive(rel)

THEN ADD (first-rel a) AND (last-rel f)

where a,b, ..., f are components, and rel is a relationship between components. This rule will add the two new features (first-rel a) and (last-rel f) into the instance description.

CONS-RULE4:

IF value(attrib, a) < value(attrib, b) < ... < value(attrib,e)

THEN ADD (min-attrib a) AND (max-attrib e)

where a,b,...,e are components. This rule will add the two new features (min-attrib a) and (max-attrib e) into the instance description.

(C) Concept Constraints

With domain knowledge, such as physical laws, we know certain relationships among attributes, components and concepts. This kind of knowledge can be used as semantic constraints on input instances. Constraint rules are used in two processes, the basic abstraction and the advanced abstraction. During the basic abstraction, when trying to incorporate a new instance into a basic concept, if the constraints of the basic concept are violated by the new instance, the induction system identifies an inconsistent instance. Then the inductive system can discard the erroneous instance so that the generalizations are not affected by the inconsistent instance. During the advanced abstraction, when trying to find the best cluster to incorporate a new basic concept, the induction system first checks whether the description of the cluster violates the constraints of the basic concept. If so, the system will not consider this cluster for further comparison.

An example depicting two constraint rules is given below:

(1) IF (direction(sensor-1) = POS)

THEN (direction(sensor-2) = NEG)

(2) IF (rate(sensor-1) = value1)(rate(sensor-2) = value2)

THEN (value1 < value2)

Inductive inference can be viewed as a biased search in the hypothesis space. Domain knowledge has been used to specify certain constraints and preferences for attributes and relationships, so that the search process is more efficient and the resulting description is more useful.

3.2.4. Methods and Algorithms for Basic Abstraction

3.2.4.1. Introduction

Corresponding to conceptual inductive learning from examples, basic abstraction is a type of supervised learning. Basic abstraction consists of several levels of operations:

(1) transformation operations,

(2) aggregation operations, and

(3) generalization operations.

A **transformation operation** changes data into a processable and useful form. It can be either simple transformation or deductive transformation. Simple transformation eliminates unnecessary detail information in an instance description, such as transforming continuous values into discrete values, or transforming numerical values into symbolic values. Deductive transformation applies deductive rules to the descriptions in order to derive new useful features.

An **aggregation operation** generates collective features about a set of objects. It can be performed on three levels: (1) attribute level, (2) component level, or (3) concept level. On the attribute level aggregation generates statistical features about a set of values, such as *average*, *minimum* and *maximum*. On the component level aggregation generates features about a set of components of a concept instance, such as

[No-of-components = 3 WITH duration = long]

[first-started-component = MCCPC]

where MCCPC is a sensor's name.

On the concept level, aggregation generates features about the set of instances of a concept, such as

[max-duration (MCCPC) = 0.2]

A **generalization operation** generates a description about a concept from concept instances. The description logically covers each instance of the concept (that is, it is satisfied by each instance). Generalization is different from concept-level aggregation. In concept-level aggregation, a description is about the whole class rather than individual instances. For example,

[duration(MCCPC) = short]

is a feature in a description resulting from generalization which describes the feature of every individual instance of a concept, but

[max-duration(MCCPC) = 0.2]

is a feature in a description of aggregation which describes the feature of the whole set of instances.

KNOWD is an incremental model, so it is more efficient than a logic-based model in the database environment, and the KNOWD model has a capability of rich logic representation which a set-based model does not support. Accommodation of many attribute types and inexact value matching are two of KNOWD's capability which are not shared by most other models. In this section we shall present the methods and algorithms for basic abstraction, which involve constructing both characteristic and discriminant descriptions.

3.2.4.2 Generalization Rules

In the KNOWD methodology, generalization operations are performed in an incremental way. Therefore, a generalization operation is applied on descriptions of a concept and a new instance. Each time a new instance is added, the previous concept descriptions will be generalized to incorporate the new instance. A generalization rule can be formally stated as follows. Let the current description of a concept be $D = \{EXP_1, ..., EXP_n\}$, where EXP_i $(i = 1, ..., n)$ are expressions, and the description of an instance be $d = \{exp_1, ..., exp_m\}$, where exp_j $(j = 1, ..., m)$ are conjunctions of atomic expressions. The generalization rule on the description level is:

$$D;d \Rightarrow_G D'$$

where "$\Rightarrow_G$" stands for the generalization of two descriptions and

$$D' = \{ EXP'_1, ..., EXP'_i \}$$

is the new description of the concept such that

$$(\forall u)\{(EXP'_u \in D') \rightarrow [(AND(D) \Rightarrow EXP'_u) \wedge (AND(d) \Rightarrow EXP'_u)]\}$$

The generalization rule on the expression level is:

$$EXP;\ exp \Rightarrow_G EXP'$$

where EXP $\in$ D, exp $\in$ d and EXP' $\in$ D' such that

$$EXP \Rightarrow EXP' \text{ and } exp \Rightarrow EXP'$$

In each expression-level generalization rule, there are two conjunctions on the left side. One is an expression of a concept; the other is an expression of a new instance of the concept. The right side is a generalization of the expressions of the concept and the new instance. The following are examples of generalization rules some of them are converted from rules described by Michalski [1983b]. Let CONJ1 and CONJ2 be conjunctions, and let A1 and A2 be atomic expressions.

Adding-Alternative Rule

AA-RULE: $CONJ1 \wedge A1; CONJ2 \wedge A2 \Rightarrow_G CONJ1 \wedge (A1 \vee A2)$

where CONJ1 is a generalization of CONJ2. This rule generalizes the two expressions on the left side and generate the new expression on the right side. This is the Adding-alternative rule on the expression level. The Adding-alternative rule on the description level is shown below:

$$CONJ1; CONJ2 \Rightarrow_G CONJ1 \vee CONJ2$$

Extending-Reference Rule

ER-RULE: $CONJ1 \wedge [L=R1]; CONJ2 \wedge [L=R2] \Rightarrow_G CONJ1 \wedge [L=R3]$

where R1 and R2 are subsets of R3, and CONJ1 covers CONJ2 .This rule generalizes the two expressions on the left side and generate the new expression on the right side. $CONJ1 \wedge [L=R3]$ is a generalization of both $CONJ1 \wedge [L=R1]$ and $CONJ2 \wedge [L=R2]$.

Closing-Interval Rule

CI-RULE: $CONJ1 \wedge [L=a]; CONJ2 \wedge [L=b] \Rightarrow_G CONJ1 \wedge [L=a..b]$

where CONJ1 covers CONJ2, L is a linear attribute, and a<b. This rule generalizes the two expressions on the left side and generate the new expression on the right side. $CONJ1 \wedge [L=a..b]$ is a generalization of both $CONJ1 \wedge [L=a]$ and $CONJ2 \wedge [L=b]$.

Extending-Interval Rule

EI-RULE: $CONJ1 \wedge [L=a..b]; CONJ2 \wedge [L=c] \Rightarrow_G CONJ1 \wedge [L=a..c]$

where L is a linear attribute, b<c, and CONJ1 covers CONJ2. This rule generalizes the two expressions on the left side and generate the new expression on the right side. For c<a, a rule can be defined similarly.

Climbing-Generalization-Tree Rule

CGT-RULE: CONJ1 $\wedge$ [L=a]; CONJ2 $\wedge$ [L=k] $\Rightarrow_G$ CONJ1 $\wedge$ [L=s]

where CONJ1 covers CONJ2, L is a hierarchical attribute, and s is the lowest ancestor of a and k in the generalization tree. This rule generalizes the two expressions on the left side and generate the new expression on the right side. **CONJ1 $\wedge$ [L=s]** is a generalization of both **CONJ1 $\wedge$ [L=a]** and **CONJ2 $\wedge$ [L=k].**

Inductive-Resolution Rule

IR-RULE: CONJ1 $\wedge$ P; CONJ2 $\wedge$ $\neg$P $\Rightarrow_G$ CONJ1 $\vee$ CONJ2

This rule generalizes the two expressions on the left side and generate the new expression on the right side. **CONJ1 $\vee$ CONJ2** is a generalization of both **CONJ1 $\wedge$ P** and **CONJ2 $\wedge$ $\neg$P.**

Dropping-Condition Rule

DC-RULE: {EXP1, EXP2, ... EXPn}; {EXP1', EXP2', ..., EXPn'}

$\Rightarrow_G$ {EXP2, ..., EXPn}

where EXPi covers EXPi' (i = 2,...,n). This is a generalization rule on the description level. This rule generalizes the two descriptions on the left side and generate the new description on the right side.

Example: The uses of the generalization rules are shown below. Given an expression of a concept description:

([color(cube) = red] $\wedge$ [color(ball) = blue]

$\wedge$ [size(cube) = small])

and an expression of an instance description:

([color(cube) = red] $\wedge$ [color(ball) = blue]

$\wedge$ [size(cube) = large]).

Using the AA-RULE will generate the description:

([color(cube) = red] ∧ [color(ball) = blue] ∧

([size(cube) = small] ∨ [size(cube) = large]))

Given an expression of a concept description:

([color(cube) = red] ∧ [color(ball) = blue]

∧ [weight(cube) = 1])

and an expression of an instance description:

([color(cube) = red] ∧ [color(ball) = blue]

∧ [weight(cube) = 3])

Using the CI-RULE will generate the expression:

([color(cube) = red] ∧ [color(ball) = blue] ∧

[weight(cube) = 1..3])

[End of Example]

For each rule three functions are defined. The condition-testing function is used to test the applicable conditions. When those conditions are satisfied, a rule becomes a candidate for execution. The worth-computing function is used to compute the worth of each rule, so that the best generalization can be performed. The rule-action function executes the selected rule and performs the generalization. A rule has two states: "enable" and "disable." In a specific application domain, not all generalization rules are needed. In order to efficiently implement the system in a given domain, a developer can enable the rules pertinent to the domain during the initialization of the system. Only enabled rules can be chosen as candidates. Unused rules can be disabled in order to improve the efficiency. An initial worth is assigned to a rule. Usage count is used to monitor the usage of rules so that worth of rules can shift during the inductive process.

Because our representation accommodates several attribute types, we need different generalization rules to process different attributes. For example, the closing-interval rule and extending-interval rule are applied only to the linear attributes; the climbing-generalization-tree rule is applied only to the hierarchical attributes. Many induction systems with limited

representation use only dropping-condition rule and adding-alternative rule. Comparing to those methods, our method can apply generalization to more situations and generate concept descriptions of higher quality.

3.2.4.3. Constructing Characteristic Descriptions

Characteristic descriptions of concepts are descriptions containing features generalized from features of instances. In this subsection we discuss matching of descriptions, generalization of features, and the algorithm to construct characteristic descriptions.

(A) Matching of Descriptions

During incremental inductive processing, a description of a concept may need to be generalized to incorporate a new instance of the concept. Generalization is performed on two levels: on the description level, where only the drop-condition rule and the add-alternative rule are applicable, and on the expression level, where matching logical expressions are generalized by various rules.

Generalization on a logical expression level is performed only on matching expressions. Two logical expressions are matching, if (1) they are matching conjunctions, or (2) they are disjunctions of conjunctions and each conjunction in one expression can find a matching conjunction in the other expression. Two conjunctions are matching if all atoms in either conjunction can find matching atoms in the other conjunction. Two atoms are matching, if (1) they have the same component, (2) they have the same attribute, and (3) they have the same relational operation. The following is an example of matching expressions.

EXP1 = [changing-rate (sensor1) = 2] $\wedge$
[changing-direction (sensor1) = Positive]

and

EXP2 = [changing-rate (sensor1) = 3] $\wedge$
[changing-direction (sensor1) = Negative]

The matching conditions eliminate the generation of unimportant or useless features as shown by the following examples.

Example: This example illustrates the case where non-matching features may have different components. Let the expression EXP1 be

[length (leg) = 10]

and EXP2 be

[length (tail) = 20].

If we generalize these two expressions, then the generalized expression would be

[length (X) = 10 $\vee$ 20),

which means there is a certain component X whose length is 10 or 20. This kind of statement does not give much information about a concept to a user and is considered useless or unimportant. Since these two expressions are not actually matching, such generalizations are eliminated by imposing matching conditions. [End of Example]

Example: In the KNOWD model, different logic connectives represent different relationships between attributes or components. The generalization of expressions with different connectives may change those relationships. For example, let EXP1 be

[color = red] $\wedge$ [size = big] $\vee$ [weight = heavy],

and EXP2 be

[color = red] $\vee$ [size = big] $\wedge$ [weight = heavy].

If we generalize EXP1 and EXP2, the resulting expression would be

[color = red] $\vee$ [size = big] $\vee$ [weight = heavy]

which represents different relationships between features. This type of generalization is eliminated since usually we need generalizations that keep the relationships of features unchanged. [End of Example]

The matching conditions restrict the number of features which are matching between two descriptions so that the number of possible generalizations of features is reduced. Therefore, restrictions generated by the above conditions contribute to the efficiency of incremental induction.

(B) Generalization of Logical expressions

A logical expression EXP1 **covers** a logical expression EXP2, if and only if EXP2 $\Rightarrow$ EXP1, that is EXP1 is a generalization of EXP2. For example, [temperature (sensor1) = 100..200] covers [temperature (sensor1) = 150], because that sensor1's value is 150 which implies that sensor1's value is within the range of [100, 200].

The expression generalization procedure takes two matching logical expressions EXP1 and EXP2 as input, and tries to find out an expression EXP which is a generalization of EXP1 and EXP2.

First, the two expressions are compared to see whether one is a generalization of the other. This is determined by comparing the formats of logic expressions and values of related atomic expressions. If EXP1 is a generalization of EXP2 then let EXP = EXP1; otherwise, if EXP2 is a generalization of EXP1 then let EXP = EXP2.

If none is a generalization of the other, the generalization operation is needed and several steps are taken:

(1) Find the set of applicable generalization rules;

(2) Select the generalization rule with the highest worth;

(3) Apply the selected generalization rule to EXP1 and EXP2 for generating the more general expression EXP.

The capability of generalizing logical expressions is a strong point of the KNOWD model, since other incremental inductive systems of set-based model can only select or drop features from a fixed set of features for describing a concept.

(C) The Algorithm for Constructing Characteristic Descriptions

The KNOWD methodology employs an incremental inductive approach. Each time an instance of a specified concept is incorporated into the concept hierarchy, the object of the concept and the object of the instance are sent to the procedure that incrementally modifies the

characteristic description of the concept. The algorithm for construction of characteristic descriptions can be described as follows:

Let **concept-obj** be a concept object representing a basic concept, and **instance-obj** be an instance object representing an instance of the concept.

generate-cha-des(concept-obj, instance-obj)

(1) Check whether **concept-obj** is a newly incorporated basic concept. If it is a new concept, the instance description is taken as the characteristic description of the concept. If it is not a new concept, then perform the following steps.

(2) For each feature of **instance-obj** try to find the matching features in the concept description.

   a. If not found, put the feature into the Alternative List AL.

   b. If found, generalize the two matching features by applying a generalization rule, use the resultant feature as one feature of the characteristic descriptions.

(3) Apply the Add-Alternative Rule to the unmatched features of the concept characteristic descriptions and features in AL. The resultant features are put into the concept characteristic description. Also some unqualified features are eliminated from the characteristic description of the basic concept.

The generalization rules assure that each resulting feature in the characteristic description will have a completeness factor 1.0.

3.2.4.4. Constructing Discriminant Descriptions

Discriminant descriptions of concepts are descriptions which contain features uniquely possessed by instances of each concept. Least generalization and specialization are two basic operations for constructing discriminant descriptions. In this subsection we discuss least

generalization and specialization, maintenance of uniqueness of features, and the algorithm used to construct discriminant descriptions.

(A) Least Generalization and Least Specialization

Generalization, denoted "$\Rightarrow$", is a relation on A, the set of logical expressions. It is easy to prove that $\Rightarrow$ is reflexive, antisymmetric and transitive. Therefore, [A; $\Rightarrow$] is a partially ordered set (poset).

Definition: We say a logical expression e is the **least generalization** of logical expressions e1 and e2, if and only if e1 $\Rightarrow$ e, e2 $\Rightarrow$ e, and for all e3, if [(e1 $\Rightarrow$ e3) $\wedge$ (e2 $\Rightarrow$ e3)] then (e $\Rightarrow$ e3). We denote the least generalization of e1 and e2 as

$$e1;\ e2\ \Rightarrow_{LG}\ e.$$

We can see that the least generalization of a pair of logical expressions is the least upper bound (or join) of them.

Theorem: For logical expressions e1 and e2 we have: (e1; e2 $\Rightarrow_{LG}$ e) if and only if (e = e1 $\vee$ e2).

Proof: From this theorem we can see that in the poset [A; $\Rightarrow$] the logical connective $\vee$ has the same meaning as the join.

First let us prove:

$$\text{If } (e1;\ e2 \Rightarrow_{LG} e) \text{ then } (e = e1 \vee e2).$$

Suppose (e1; e2 $\Rightarrow_{LG}$ e), we have (e1 $\Rightarrow$ e), (e2 $\Rightarrow$ e), and for all e3, if [(e1 $\Rightarrow$ e3) $\wedge$ (e2 $\Rightarrow$ e3)] then (e $\Rightarrow$ e3).

Let e3 = e1 $\vee$ e2. Since e1 $\Rightarrow$ (e1 $\vee$ e2) and e2 $\Rightarrow$ (e1 $\vee$ e2), we have

$$e \Rightarrow (e1 \vee e2) \tag{3.1}$$

$$(e1 \Rightarrow e) \wedge (e2 \Rightarrow e) = (\neg e1 \vee e) \wedge (\neg e2 \vee e) \tag{3.2}$$

We also have

$$[(e1 \vee e2) \Rightarrow e] = \neg (e1 \vee e2) \vee e = (\neg e1 \wedge \neg e2) \vee e$$

$$= (\neg e1 \vee e) \wedge (\neg e2 \vee e) \tag{3.3}$$

From (3.2) and (3.3) we can see that

$$(e1 \Rightarrow e) \wedge (e2 \Rightarrow e) = [(e1 \vee e2) \Rightarrow e] \quad (3.4)$$

From (3.1) and (3.4) we can say $e = e1 \vee e2$.

Now let us prove:

$$\text{If } (e = e1 \vee e2) \text{ then } (e1 ; e2 \Rightarrow_{LG} e)$$

Given $e = e1 \vee e2$. Since $e1 \Rightarrow e1 \vee e2$ and $e2 \Rightarrow e1 \vee e2$, we have $e1 \Rightarrow e$ and $e2 \Rightarrow e$.

For all e3, if $e1 \Rightarrow e3$ and $e2 \Rightarrow e3$ then we have:

$(e1 \Rightarrow e3) \wedge (e2 \Rightarrow e3)$

$= (\neg e1 \vee e3) \wedge (\neg e2 \vee e3)$

$= (\neg e1 \wedge \neg e2) \vee e3$

$= \neg (e1 \vee e2) \vee e3$

$= (e1 \vee e2) \Rightarrow e3$

So $e \Rightarrow e3$. Therefore, by the definition of least generalization we have $(e1; e2 \Rightarrow_{LG} e)$. [End of Proof]

The least generalization operation is used for modifying the discriminant descriptions of a basic concept when incorporating a new instance so that the discriminant description maintains uniqueness. The purpose of "least" generalization is to ensure that the resultant expression is not a generalization of any expression of other concept instances.

For different types of attributes the process of least generalization is different. For a linear attribute, two single values are generalized as an or-form; if one value is a range-form or an or-form, the generalized value form is a range-form only when all values of the resultant range are included in the two processed expressions; otherwise, the generalized value is an or-form. For a hierarchical attribute, the resultant value would be in a single form if all the offspring values of the resultant value are included in the two processed expressions. Otherwise, the generalized value is an or-form. For a nominal attribute, the generalized value is only of an or-form.

Definition: The **specialization** of a logical expression e2 with respect to e1 is a logical expression e3 such that e2 is a generalization of e3 and for any given situation $e1 \wedge e3 = \textbf{false}$,

which is denoted as: $e2 \xrightarrow{e1}_{S} e3$

Example: Let e2 = [length(a) = 1∨3∨5] and e1 = [length(a) = 1],

then e3 = [length(a) = 3∨5] and e4 = [length(a) = 3] are both specializations of e2 with respect to e1. [End of Example]

Definition: The **least specialization** of e2 with respect to e1 is an expression e3 such that

$e2 \xrightarrow{e1}_{S} e3$ and for all e, if $e2 \xrightarrow{e1}_{S} e$ then $(e \Rightarrow e3)$. The least specialization of

e2 with respect to e1 is denoted as: $e2 \xrightarrow{e1}_{LS} e3$

Example: Let e2 = [length(a) = 1∨3∨5] and e1 = [length(a) = 1],

then e3 = [length(a) = 3∨5] is the least specializations of e2 with respect to e1. For e4 = [length(a) = 3], a specialization of e2 with respect to e1, we have $e4 \Rightarrow e3$.

[End of Example]

Theorem: For logical expressions e1 and e2, $e2 \xrightarrow{e1}_{LS} e3$ if and only if $e2 \xrightarrow{e1}_{S} e3$

and $(e2 \wedge e1) \vee e3 = e2$.

Proof: First let us prove the statement:

if $e2 \xrightarrow{e1}_{LS} e3$ then $e2 \xrightarrow{e1}_{S} e3$ and $(e2 \wedge e1) \vee e3 = e2$.

Given $e2 \xrightarrow{e1}_{LS} e3$ By the definition of least specialization we know:

$e2 \xrightarrow{e1}_{S} e3$, and for all e, if $e2 \xrightarrow{e1}_{S} e$ then e $\Rightarrow$ e3.

Then by the definition of specialization we have e3 $\Rightarrow$ e2, e1 $\wedge$ e3 = **false**, and

for all e, if e $\Rightarrow$ e2 and e1 $\wedge$ e = false then e $\Rightarrow$ e3 (3.5)

Since e3 $\Rightarrow$ e2, that is, there is a logical expression x such that

$$e2 = e3 \vee x \tag{3.6}$$

So e2 $\vee$ e3 = e3 $\vee$ x $\vee$ e3 = e2 (3.7)

And we have

(e2 $\wedge$ e1) $\vee$ e3 = (e2 $\vee$ e3) $\wedge$ (e1 $\vee$ e3) = e2 $\wedge$ (e1 $\vee$ e3).

Since e2 $\wedge$ (e1 $\vee$ e3) $\Rightarrow$ e2, (3.8)

Now let us prove e2 $\Rightarrow$ e2 $\wedge$ (e1 $\vee$ e3)

First let us prove e2 $\Rightarrow$ e1 $\vee$ e3.

Let e = e2 $\wedge$ $\neg$(e1 $\vee$ e3)

It is easy to see that e $\Rightarrow$ e2 and e $\wedge$ e1 = **false.**

By (3.5) we get e $\Rightarrow$ e3.

Thus, we say e = **false,** that is,

$$e2 \wedge \neg(e1 \vee e3) = \textbf{false} \tag{3.9}$$

Otherwise, we have [e2 $\wedge$ $\neg$(e1 $\vee$ e3)] $\Rightarrow$ e3, that is, (e2 $\wedge$ $\neg$e1 $\wedge$ $\neg$e3) $\Rightarrow$ e3 which is a contradiction.

By (3.9), e2 $\wedge$ (e1 $\vee$ e3) = e2 which says for all situations if e2 is **true** then (e1 $\vee$ e3) is **true.**

Therefore, e2 $\Rightarrow$ e1 $\vee$ e3 (3.10)

By (3.8) and (3.10), we have e2 = e2 $\wedge$ (e3 $\vee$ e1) and we proved the statement.

Now let us prove the statement:

If $e2 \xrightarrow{e1}_{S} e3$ and $(e2 \wedge e1) \vee e3 = e2$, then $e2 \xrightarrow{e1}_{LS} e3$.

Given $e2 \xrightarrow{e1}_{S} e3$ and $(e2 \wedge e1) \vee e3 = e2$. Let e be a logical expression and

$e2 \xrightarrow{e1}_{S} e$ that is, $e \Rightarrow e2$ and $e1 \wedge e = \mathbf{false}$.

Since $(e2 \wedge e1) \Rightarrow e2$ and $e \Rightarrow e2$, we have $(e2 \wedge e1) \vee e \Rightarrow e2$

Since $(e2 \wedge e1) \vee e3 = e2$, we have

$$(e2 \wedge e1) \vee e \Rightarrow (e2 \wedge e1) \vee e3.$$

So $e \Rightarrow (e2 \wedge e1) \vee e3$, and $e \Rightarrow [(e2 \wedge e1) \vee e3] \wedge e$, that is,

$$e \Rightarrow (e2 \wedge e1 \wedge e) \vee (e3 \wedge e).$$

Since $e1 \wedge e = \mathbf{false}$, we have $e \Rightarrow e3 \wedge e \Rightarrow e3$.

Therefore, by the definition of least specialization we have

$$e2 \xrightarrow{e1}_{LS} e3 \ .$$

[End of Proof]

The least specialization is another basic operation to be applied to expressions. When a basic concept C1 incorporates a new instance, an expression EXP1 in the instance description may cause the discriminant description of another concept C2 to lose its uniqueness. Therefore, we

need to specialize the discriminant description of C2 so that any expression in C2 is not a generalization of EXP1.

We prefer a discriminant description that is satisfied by more instances of the concept (that is, with a higher completeness factor), so we try to keep the description as general as possible. For this reason we employ "least" specialization.

The processing of least specialization depends upon the types of attributes. For a linear attribute, an or-form value may be specialized by eliminating related values, and a single form would be generated if only one value remains; otherwise an or-form value would be generated. A range-form value may be specialized into a single form if only one value remains and into a range-form if remaining values constitute a range; otherwise, into an or-form. For a hierarchical attribute, the resultant value would be in a single form if the remaining values include all the offspring values of the single-form value; otherwise, it would be in an or-form. For a nominal attribute, the specialized value may be in a single form if only one value remains; otherwise, in an or-form.

(B) Maintenance of Uniqueness in Discriminant Descriptions

Incrementally constructing discriminant descriptions is a time-consuming process, because whenever a new instance d is incorporated into a concept C_i, two things need to be examined: (1) no features in the generalized discriminant description of C_i should be a generalization of any features of other concepts' instances; and (2) no feature in the discriminant descriptions of any other concept C_j, $j \in \{ 1, ..., u \}$, $(j \neq i)$ should be a generalization of any feature in d.

In order to improve the time efficiency of the process, we developed techniques to facilitate the checking. A uniqueness table (UTable) is used to record the uniqueness of all the input features and their related concepts. If a concept C has a discriminant feature EXP1, then in the Utable there is an entry for EXP1 that records the concept name C and sets the uniqueness flag on. If a feature EXP2 has been identified as non-discriminant (that is, it is shared by instances of two different concepts), then in the UTable there is an entry for EXP2 that sets the uniqueness flag off. Now we do not need to check the descriptions of all instances of the concepts, since we

can get the required information from the UTable. The number of entries in the UTable is much smaller than the total number of features in the input because the common features among all instances of the input are represented only once in the UTable.

The UTable is built on a discrimination net, and expressions are used as indices. A discrimination net is a tree-like data structure used to store and retrieve information. It uses a list of keys for indexing, and each key can be either a constant or a matching variable. The keys are used for directing the search. A discrimination net representation makes it possible to share common parts of expressions so that both space-efficiency and time-efficiency can be improved (discrimination nets are detailed in [Charniak et al. 1980]).

(C) The Algorithm for Constructing Discriminant Descriptions

Let **concept-obj** be an object of concept C1 and **instance-obj** be an object of C1's instance.

generate-dis-des(concept-obj, instance-obj)

For each feature EXP in the instance description perform the following steps:

(1) Obtain all matching features from the uniqueness table UTable.

(2) If there is no matching feature from UTable, then EXP is a unique feature. Perform the following:

Add EXP to the discriminant description of C1.

Add an entry for EXP in the UTable, and exit.

(3) If there are matching features, then try to find inconsistencies. Let ML be the list of matching feature entries. (Loop A) For each entry in ML, check for inconsistency:

a) If the feature EXP is a **discriminant feature** (that is, it belongs to the discriminant description of a concept) of another concept C2, then an **inconsistency** situation is found. Perform the following:

Delete EXP from the discriminant description of C2.

Set the entry uniqueness flag of EXP off.

Exit Loop A.

b) If EXP tautologically implies a discriminant feature EXP1 of another concept C3, then an inconsistent situation is found. Perform the following:

Perform the least specialization on the discriminant feature EXP1.

Modify the entry in UTable.

Exit Loop A.

c) If EXP tautologically implies a **non-discriminant feature** (a feature that is not contained in the discriminant description of any concept), then EXP is non-discriminant. Exit Loop A.

(4) If EXP is not found to be non-discriminant in step 3, then check for partial inconsistency. For each entry in ML, if the feature EXP2 in the entry is of another concept and is partially covered by EXP (that is, EXP contains part of EXP2's values), then

a) Perform the least specialization on both features;

b) Modify the discriminant descriptions of the concept specified in the entry by replacing EXP2 with its specialization;

c) Modify the entry by replacing EXP2 with its specialization.

(5) If EXP is matching a discriminant feature EXP3 of C1 (in which case EXP is said to be **compatible**), then

a) Perform the least generalization on EXP and EXP3;

b) Modify C1's discriminant description

c) Modify the entry in Utable.

(6) If EXP is neither non-discriminant nor compatible, then

a) Add EXP to C1's discriminant description;

b) Create an entry for EXP in Utable.

3.2.5. Methods and Algorithms of the Advanced Abstraction

3.2.5.1. Introduction

Advanced abstraction is conceptual clustering, which is a kind of unsupervised learning that discovers high-level concepts which characterize classes of basic concepts. Given the basic concepts, advanced abstraction tries to classify them into meaningful classes which can describe a system (composed of basic concepts) from a higher-level perspective. As in most incremental learning methods [Lebowitz 1986; Fisher 1987], we use a generalization hierarchy to represent the partially learned concepts. Each node in the hierarchy represents a concept, and nodes are stored as database objects. The nodes on higher levels represent more general concepts, and the nodes on the lowest level represent the basic concepts. As more basic concepts are added incrementally into the hierarchy, more nodes (representing new concepts) will be added into the hierarchy. Unlike methods of numerical taxonomy, advanced abstraction generates not only clusters over the basic concepts but also the descriptions of the high-level concepts corresponding to the clusters on various levels.

The advanced abstraction is an integration of two incremental processes:

(1) modification of the concept hierarchy, and

(2) modification of cluster descriptions.

The results of the advanced abstraction are a concept hierarchy and conceptual descriptions of the clusters. The processes of cluster generation and description generation cannot be separated because clustering determines the generation of descriptions, and descriptions are used to determine the quality of clusters.

Unlike previous methods, advanced abstraction takes basic concepts (which are learned from input instances) as the examples for expanding the generalization hierarchy. The basic concepts do not have fixed descriptions; instead, they may change with the induction over new instances. There is no external teacher to preclassify the basic concepts, so the advanced abstraction relies upon three factors to build the concept hierarchy:

(1) domain knowledge, which includes concept constraints and other knowledge that affects concept features (such as importance of attributes, relevance of components, and feature relationships);

(2) clustering criteria, which determine the clustering quality by an evaluation measure; and

(3) instance descriptions, which represent instances of basic concepts by relevant attributes and components specified by domain experts.

The combination of incremental processes with an expressive logic representation language makes KNOWD a unique methodology in the conceptual clustering area. In this subsection, we discuss (1) the operations on the concept hierarchy; (2) an algorithm for top-down extension of concept hierarchy; and (3) an algorithm for bottom-up modification of concept hierarchy.

3.2.5.2. Operations on the Concept Hierarchy

The hierarchy modification process is performed through the following operations on the concept hierarchy:

Operation 1: Adding a basic concept as a leaf;

Operation 2: Creating a new cluster node by combining two concept leaves;

Operation 3: Incorporating a concept instance with a subcluster;

Operation 4: Deleting an unqualified cluster node.

Although these operations are performed on the structure of the concept hierarchy, they are interwoven with the generalization of conceptual descriptions.

The KNOWD methodology allows a basic concept object to become a leaf of a cluster node on any level of the hierarchy, so that each cluster node has a set of child nodes representing basic concepts and subclusters. In the top-down hierarchy-extending process, when at a cluster node there are no subclusters or basic concepts that are "close" enough to the new basic concept, the new basic concept will be added as a leaf of the cluster.

There are two chances to create a new cluster. In the top-down hierarchy extending process, if there exists a basic concept that is the "closest" to the new basic concept, then a new

subcluster is created for the current cluster, and the two concept nodes are made leaves for the new subcluster. In the bottom-up hierarchy modification process, if the modified concept comes "close" enough to a sibling concept, then a new cluster is also created. Figure 3.4 shows this operation.

During the top-down hierarchy extending process, operation 3 is the most frequent operation. When a subcluster is found "closest" to the newly input concept at a cluster node, then the new concept is incorporated with it. This involves generalizing the description of the subcluster.

Deleting a cluster is similar to backtracking. If the initial set of instances is not the typical set, then the initial hierarchy strongly affects the rest of the clustering process. To counter the unfavorable initial conditions, the KNOWD method uses a cluster-deletion operation to eliminate the incorrect classification. The cluster-deletion operation is performed in the bottom up hierarchy modification algorithm when the intercluster similarity, which is computed as the number of qualified features in the cluster, becomes lower than the cluster retaining threshold. When a cluster node is deleted from the hierarchy, all its basic concepts and subclusters will be adopted by its parent. This operation is shown in Figure 3.5.

KNOWD's advanced abstraction process does not require a predetermined number of classes into which all instances are grouped. In many real situations, people do not know what high-level concepts might be found in the instance data of a database, and so it is usually impossible to provide such a number. Some methods [Stepp 1984] need this number to cut down the computation cost. In the KNOWD method, a number L_{max} is used for limiting the number of levels of the hierarchy. L_{max} is not for the purpose of time-efficiency but for the control of the hierarchy structure to better satisfy the clustering goal. For example, if we want one level of classification, we can simply set L_{max} to one.

For an incremental conceptual clustering method, it is usually impossible to generate an overall optimal clustering at each step. The later clustering must be based on the previous experience of the process, and newly arriving instances cause the current clusters to be modified

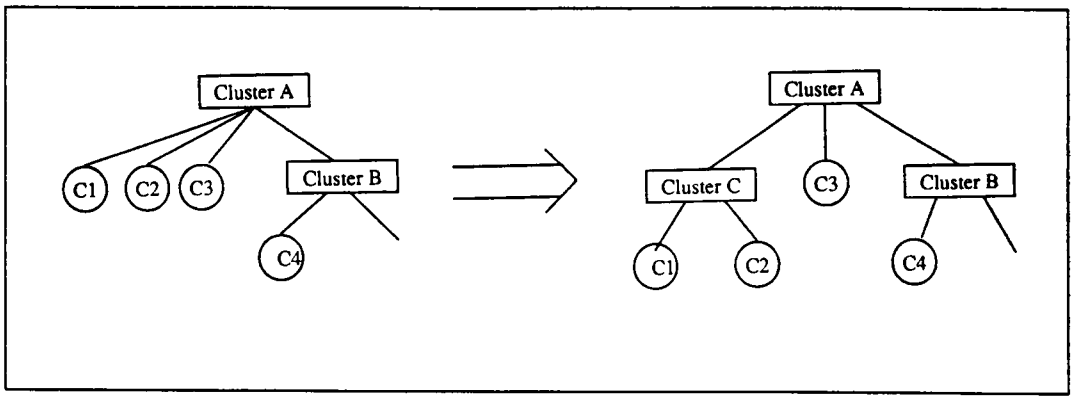

Figure 3.4. Creating a new cluster.

according to a locally-optimum criterion. Generally speaking, any method which tries to reconstruct the whole hierarchy in order to globally optimize the clustering when incorporating each new instance, is not an incremental method and is intractable because the number of possible hierarchies is incredibly large, even for a small number of instances.

The KNOWD method uses a mixed control strategy. When a new basic concept is introduced, a top-down search is performed. Either the new basic concept is merged into an existing cluster or a new cluster is created based on the classification criteria. When an existing basic concept is changed, a bottom-up modification is performed, and all the clusters above the basic concept are modified accordingly.

3.2.5.3. Top-down Extension of Concept Hierarchy

Features in a cluster description will be constantly modified during the incremental inductive process. A feature has a confidence count which may be increased or decreased. The confidence count of a feature determines whether the feature is confident or not. Only the set of confident features is used as the description of a cluster. When the first instance of a new basic concept is incorporated into the concept hierarchy, a top-down extension process is performed. The procedure to perform top-down extension of the concept hierarchy is a recursive one. The first

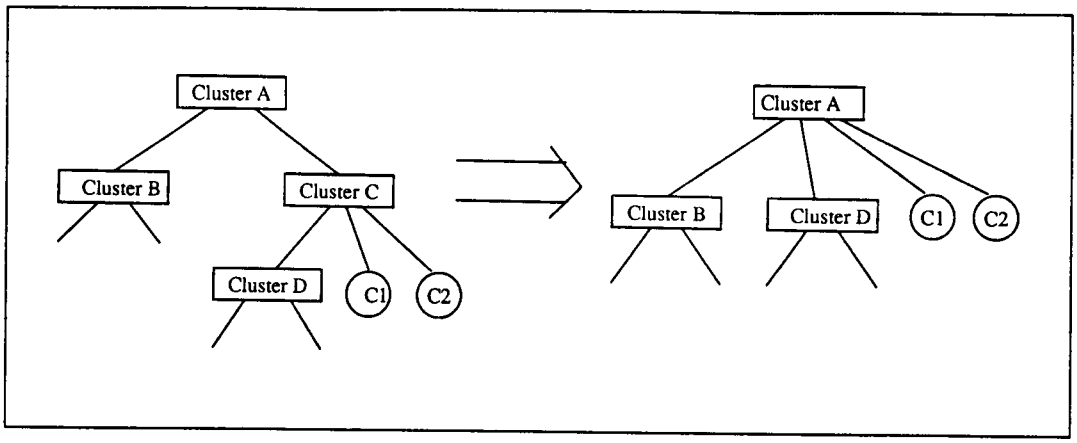

Figure 3.5. Deleting an unqualified cluster.

call uses the root node as one of the parameters. The root is a universal cluster node which incorporates all instances input to the inductive system. The description of the root is a generalization of all the incorporated instances.

A description of the procedure, named extend-hierarchy, is given below where **currentnode** is the object of the current cluster, and **newconcept** is the object representing the new basic concept to be incorporated:

extend-hierarchy (currentnode newconcept)

(1) Modify the current cluster:

a) Increase the size (by 1) of the current cluster.

b) For each feature E in the cluster description perform the following steps:

i. If there is no matching feature of E in **newconcept's confident features**[3], then check whether the feature-confidence count of E is still greater than the feature retaining threshold FR-TH[4]. Delete E if the

[3] A confident feature is a feature that has a high completeness factor and a high consistency factor.

[4] The feature retaining threshold is a system defined value used to determine when an unqualified feature needs to be eliminated from a cluster.

feature-confidence count of E is less than FR-TH. Retain E if the feature confidence count of E is not less than FR-TH.

ii. If there is a confident feature E1 in **newconcept** that matches E and E1 is covered by E, then increase the feature-confidence count of E by 1, and compute the worth of the feature.

iii. If there is a confident feature E1 in the new concept that matches E and is not covered by E, but the value matching factor of E1 and E is greater than the feature matching threshold FM-TH[5], then generalize E to incorporate E1 and increase the feature-confidence count of E by 1, and compute the worth of the feature.

iv. If there is a confident feature E1 in the new concept that matches E and is not covered by E, and the value matching factor of E1 and E is less than FM-TH but greater than the feature conflict threshold FC-TH[6], then do nothing. If the value matching factor is less than FC-TH, then decrease the feature-confidence count of E by 1, and see whether the feature-confidence count of E is less than FR-TH. Delete E if the count is less than FR-TH. Retain E if the count is not less than FR-TH.

v. Add all unmatched features of **newconcept**'s confident features to **currentnode**'s description.

(2) In the current cluster node find each subcluster S that is close to **newconcept** and whose description does not violate constraints of **newconcept**. Closeness is measured by the

---

[5] The feature matching threshold is a system defined value in the range [0, 1]. When the value matching factor of two features represented by expressions E1 and E2 is greater than the feature matching threshold, the two features are regarded as supporting each other. In a domain task, if we want to have a more strict matching, we can set FM-TH to a higher value close to 1.0. When FM-TH is 1.0, exact matching is required.

[6] Feature conflicting threshold is a system defined value in the range [0, 1]. When the value matching factor of two features represented by expressions E1 and E2 is less than the feature conflicting threshold, the two features are regarded as in conflict with each other. FC-TH is always less than FM-TH. The higher value of FC-TH makes it easier to eliminate uncommon features in a cluster's description.

matching factor between S and **newconcept**. The closeness threshold C-TH[7] is used to determine whether a subcluster is close enough to **newconcept**.

(3) Find each basic concept directly under the current cluster that is close enough to **newconcept**.

(4) If there are no close subclusters and basic concepts, then put **newconcept** directly under the current cluster as a leaf.

(5) Find the best object from those close subclusters and basic concepts according to the evaluation measure of conceptual clustering. Here the evaluation measure is modified to include basic concepts. Since a basic concept does not have a subcluster or subconcept, the cluster matching factor is replaced by the matching factor of the basic concept and **newconcept**.

(6) If the best node is a basic concept, then combine the best node with **newconcept** to form a new subcluster of the current cluster; create a new cluster; generalize features of the descriptions of the two concepts. Two features are generalized only if their value matching factor is greater than FM-TH. The new feature's confidence count is the sum of the two counts of the generalized features. The new feature's worth is the average worth of the two generalized features.

(7) If the best node is a subcluster, then recursively call:

extend-hierarchy (bestnode newconcept)

3.2.5.4. Bottom-up Modification of Concept Hierarchy

When incorporating a new instance to an existing basic concept, the description of the basic concept may be generalized to cover the new instance. This modification may cause further modifications on the cluster nodes above the basic concept.

---

[7] Closeness threshold is a system defined positive number. In order to find the best child node for incorporating a new basic concept efficiently, the closeness threshold is used to limit the scope of nodes in which the quality measure of clustering is computed. This parameter is related with the maximum number of levels of the hierarchy, so that the number of levels can be controlled. The higher C-TH value limits the hierarchy to less number of levels.

For each feature EXP in the instance's description, the following procedure is performed:

Let P be the parent cluster of the basic concept C, OLDEXP be the feature in C that matches EXP, NEWEXP be EXP or the generalization of EXP and OLDEXP.

modify-hierarchy(C, OLDEXP, NEWEXP)

(1) Get C's parent P.

(2) Try to find EXP1, the feature in P that matches OLDEXP.

a) If not found, then add NEWEXP to P's description.

b) If EXP1 is found, NEWEXP is equal to EXP, OLDEXP covers NEWEXP, and the value matching factor of OLDEXP and EXP1 is greater than FM-TH, then increase the confidence count of EXP1 (Now NEWEXP is taken as the supporting feature of EXP1).

c) If EXP1 is found, NEWEXP covers OLDEXP, and the value matching factor of NEWEXP and EXP1 is greater than FM-TH, then generalize EXP1 to incorporate NEWEXP and increase the confidence count of EXP1.

d) If EXP1 is found, NEWEXP covers OLDEXP, and the value matching factor of EXP1 and NEWEXP is less than FC-TH, then perform the following steps (now NEWEXP is considered to be in contradiction with EXP1).

i. The confidence count IC of EXP1 is decreased by 1.

ii. Test IC to see whether it is less than FR-TH.

iii. If IC is less than FR-TH, EXP1 is deleted from P's description; check the number of features in P's description; if the number is less than the cluster retaining threshold CR-TH[8], then delete P from the concept

[8] The cluster retaining threshold is a positive integer. In the process of incorporating new instances into the concept hierarchy, the number of features in a cluster description may decrease. When this number is less than the cluster retaining threshold, this cluster should be deleted from the hierarchy. This is done because instances under this cluster show more diversity than uniformity, and the cluster has little value to represent a high-level concept.

hierarchy and reassign all P's leaves and subclusters under P's parent (note: the root can never be deleted).

(3) If P has a parent, then recursively call:

modify-hierarchy(P, OLDEXP, NEWEXP).

After modifying the concept hierarchy, try to combine C with one of its sibling concept nodes and create a new cluster, since after the modification of the concept description, the concept may become close to another basic concept under the same cluster node.

3.2.6. Properties of the Induction Algorithms

For the purpose to discuss the properties of the induction algorithms, in this subsection we consider a database as a concept hierarchy which is defined in subsection 3.2.1. We will give the definition of a well-formed database and prove that the operations on the database will always result a well-formed database.

Let f and g be features, let I(f) be the set of instances $\{ d \mid d \in I \text{ and } AND(d) \Rightarrow f\}$, and let I(g) be the set of instances $\{ d \mid d \in I \text{ and } AND(d) \Rightarrow g\}$, where I is the set of all concept instances.

Definition: For a concept hierarchy R with u concept nodes $Q_i = (D_i^c, D_i^d, I_i)$, i = 1, ..., u, R is a **well-formed database** (WFDB) if and only if for every i = 1, ..., u, the following two conditions are satisfied:

(1) for all f in D_i^c, $I(f) \supseteq I_i$; and

(2) for all g in D_i^d, $I(g) \cap I_j = \varnothing$, $i \neq j$, for all j = 1, ..., u.

Initially, the concept hierarchy R does not contain any concept node. It is trivial to show that the concept hierarchy R is a WFDB. The operation on the database is a sequence of insertions of concept instances into the database. During insertion, induction is performed to generate characteristic and discriminant descriptions of concepts.

When an instance d1 of a new concept C_k is input into the database, a new concept node Q_k is created and inserted into R. The algorithm **generate-cha-des** will take d1 as the

characteristic description D_k^c, that is D_k^c = d1. Now I_k = { d1 }. For all f in D_k^c, we have f $\in$ d1, then AND(d1) $\Rightarrow$ f, so d1 $\in$ I(f). Therefore, I(f) $\supseteq$ I_k.

If an instance d1 of an existing concept C_k is input into the database, the concept node Q_k is already in the concept hierarchy R. d1 is added into I_k, and every feature f in D_k^c will be generalized with a matching feature f' in d1. Let I_k' denote the resultant set of known instances of C_k, that is, $I_k' = I_k \cup \{ d1 \}$, and let f'' be the generalization of f and f'. Assume the previous database is well-formed, that is I(f) $\supseteq$ I_k, and we have f $\Rightarrow$ f'' and f' $\Rightarrow$ f'', then we can deduce that I(f'') $\supseteq$ I_k'. The following is the proof.

Proof: By the definition we have

$$I(f) = \{ d \mid d \in I, \text{AND}(d) \Rightarrow f \}$$

$$I(f'') = \{ d \mid d \in I, \text{AND}(d) \Rightarrow f'' \}$$

For all e in I(f) we have AND(e) $\Rightarrow$ f and we know that f $\Rightarrow$ f'', then AND(e) $\Rightarrow$ f'' (since $\Rightarrow$ is transitive). That is e $\in$ I(f'').

So, $I_k \subseteq I(f) \subseteq I(f'')$ (i)

Since f' $\in$ d1 then AND(d1) $\Rightarrow$ f'. Since f' $\Rightarrow$ f'', then AND(d1) $\Rightarrow$ f'' (transitive).

So d1 $\in$ I(f'') (ii)

Therefore, I(f'') $\supseteq$ I_k' (from (i) and (ii).)

[End of Proof]

From this analysis we can see that the characteristic descriptions generated by the algorithm **generate-cha-des** satisfy the condition (1) of the definition of a WFDB. Now, let us analyze the algorithm **generate-dis-des**.

When an instance d1 of a concept C_k is input into the database, the algorithm **generate-dis-des** will check every g in d1. If $I(g) \cap I_j = \varnothing$ for all $j \neq k$, then g is added into D_k^d or is generalized with a matching feature g' in D_k^d. Since only least generalization is performed here, the resultant feature g'' will not cover any features of instances in I_j, for all $j \neq k$. That is

If $I(g) \cap I_j = \varnothing$ and $I(g') \cap I_j = \varnothing$ then $I(g \vee g') \cap I_j = \varnothing$

The following is the proof of this statement.

Proof: Suppose $I(g) \cap I_j = \varnothing$, $I(g') \cap I_j = \varnothing$, and $I(g \vee g') \cap I_j \neq \varnothing$, then there exists an instance e in $I(g \vee g') \cap I_j$.

So e is in $I(g \vee g')$ but not in I(g) or I(g'), which means $AND(e) \Rightarrow g \vee g'$, $\neg(AND(e) \Rightarrow g)$, and $\neg(AND(e) \Rightarrow g')$.

So, all the following three logical expressions are **true**

$AND(e) \rightarrow g \vee g'$, $\neg(AND(e) \rightarrow g)$, and $\neg(AND(e) \rightarrow g')$

That is, all the following expressions are **true**

$\neg AND(e) \vee g \vee g'$, $AND(e) \wedge \neg g$, and $AND(e) \wedge \neg g'$

This is impossible. Therefore, we proved the above statement.

[End of Proof]

If for a g in d1, there exists j in {1, ..., u}, such that $I(g) \cap I_j \neq \varnothing$, then there are two possible cases:

(1) There exists a discriminant feature g' in D_j^d such that $g \Rightarrow g'$.

(2) There exists i in {1,..., u} and $i \neq j$, such that $I(g) \cap (I_i \cap I_j) \neq \varnothing$. That is, g has been shown to be a non-discriminant feature.

For case (1) g' is specialized into g'' so that g'' will not cover g. For case (2) g will not be added into D_k^d and is discarded.

From above analysis we can see that the discriminant descriptions generated by the algorithm **generate-dis-des** satisfy the condition (2) of the definition of a WFDB. Therefore, we can see that the induction algorithm can guarantee that for a sequence of insertion of instances into the database the resultant database is also a WFDB.

The algorithm **generate-cha-des** generates a set of features of the characteristic description. How do we judge the quality of the features? Let us explain with the following example:

Let $f, g \in D_i^c$, so $I(f) \supseteq I_i$ and $I(g) \supseteq I_i$. Suppose

$$r(f) = card(\bigcup_{j=1, j\neq i}^{\mu} [I(f) \cap I_j]) < card(\bigcup_{j=1, j\neq i}^{\mu} [I(g) \cap I_j]) = r(g)$$

I(f) and I(g) are shown in Figure 3.6. Then we regard f to have a higher quality than g because both features cover all instances in I_i, and f covers fewer instances of other concepts than g. The consistency factor FR_2 can be used to measure the quality of features in the characteristic description because FR_2 can be represented as:

$$FR_2(I_i, f) = \frac{card(I(f) \cap I_i)}{card(\bigcup_{j=1}^{\mu} [I(f) \cap I_j])}$$

$$= \frac{card(I(f) \cap I_i)}{card(I(f) \cap I_i) + card(\bigcup_{j=1, j\neq i}^{\mu} [I(f) \cap I_j])} = \frac{card(I_i)}{card(I_i) + r(f)}$$

The best situation is when r(f) = 0, then $FR_2(I_i, f) = 1$.

Similarly, we can measure the quality of features in the discriminant description. Let f, $g \in D_i^d$, then r(f) = r(g) = 0. Suppose $|I(f) \cap I_i| > |I(g) \cap I_i|$ as shown in Figure 3.7. Then we consider f having a higher quality than g because both features do not cover instances of other concepts and f covers more instances of I_i than g does. The completeness factor can be used to measure the quality of features in D_i^d because FR_1 can be represented as:

$$FR_1(I_i, f) = \frac{card(I(f) \cap I_i)}{card(I_i)}$$

The best situation is when I(f) covers all I_i, then $FR_1(I_i, f) = 1$.

Definition: The **utility** of a concept description D_i is defined as

$$u(D_i) = \frac{\sum_{f \in D_i} [FR_1(I_i, f) + FR_2(I_i, f)]}{\sum_{f \in D_i} Complexity(f)}$$

where FR_1 is the completeness factor, FR_2 is the consistency factor, and *complexity(f)* is the the complexity of a feature.

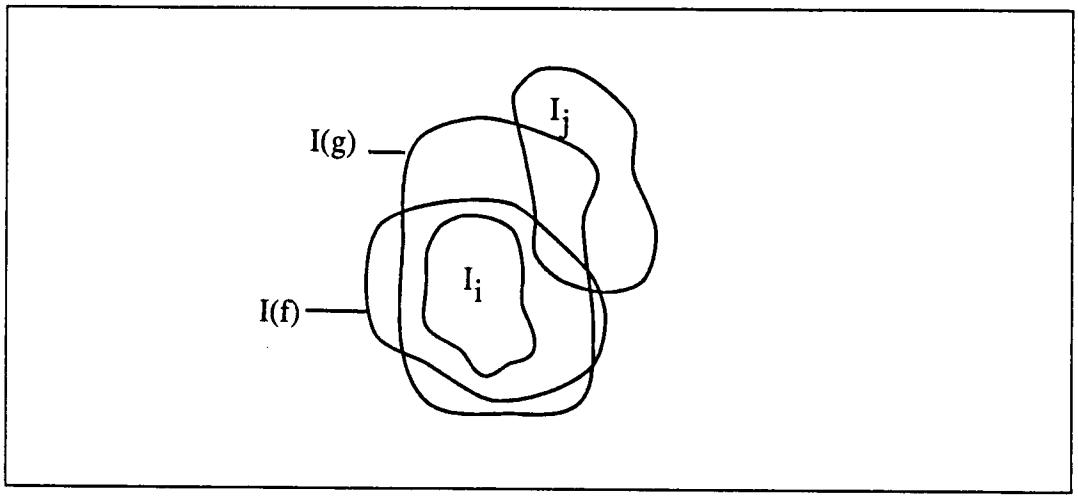

Figure 3.6. Comparison of Characteristic Features

The sum of the FR_1's gives the measure of **characterization capability** of the concept description D_i. For any instance e of concept C_i, with reasonable restriction upon the sample space we can estimate the probability that e has the feature f with $FR_1(I_i, f)$. This measure indicate how well a concept description covers the instances of the concept.

The sum of the FR_2's gives the measure of **discrimination capability** of the concept description D_i. For any concept instance e, with reasonable restriction upon the sample space, we can estimate the probability that e belongs to C_i given that e has a feature f. The estimate is $FR_2(I_i, f)$. This measure indicates how well a concept description can differentiate instances of a concept from other concepts.

The combination of characterization and discrimination capabilities is one aspect of the quality of a concept description. Another aspect of the description quality is its **simplicity**. We compute description simplicity as the reciprocal of **description complexity**. Complexity of a concept description D_i is measured by the sum of complexities of all features in D_i, while the complexity of a feature is determined by several factors including the number of atomic expressions, types of logical connectives, and forms of the attribute values. Simplicity is important for two reasons:

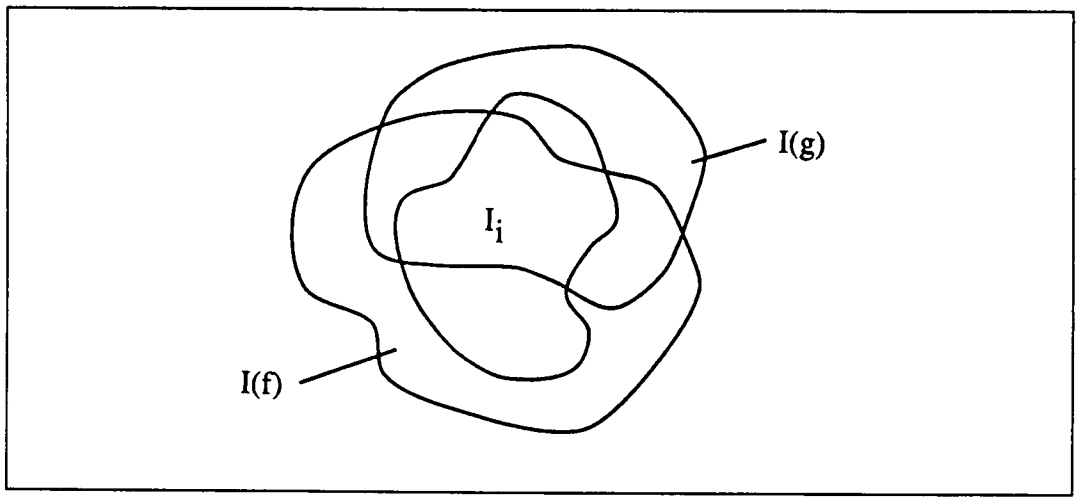

Figure 3.7. Comparison of Discriminant Features

(1) The concept descriptions are for humans to read, so the understandability is critical. Simple descriptions are easier for humans to understand.

(2) Simplicity is close related to generalization and performance of using those descriptions.

Example: Usually, a more general description is simpler. Given three instances:

e1 = [a1 = 1] ∧ [a2 = triangle] ∧ [a3 = red]

e2 = [a1 = 3] ∧ [a2 = square] ∧ [a3 = blue]

e3 = [a1 = 4] ∧ [a2 = pentagon] ∧ [a3 = red]

A generalization of them is:

D = [a1 = 1] ∧ [a2 = triangle] ∧ [a3 = red] ∨

[a1 = 3] ∧ [a2 = square] ∧ [a3 = blue] ∨

[a1 = 4] ∧ [a2 = pentagon] ∧ [a3 = red]

Another generalization of them is:

D' = [a1 = 1..4] ∧ [a2 = polygon] ∧ [a3 = red ∨ blue]

The description D' is more general and is much simpler than D. [End of Example]

Descriptions that are too specific will have less power of prediction. As indicated by Quinlan [1987], pruning the rules extracted from a decision tree makes the rules simpler. By pruning the rules, the accuracy of classification of a set of unseen test instances can be improved. Michalski compared [1987] the diagnosis performance (classification accuracy) of the concept description including all complexes (conjunction expressions) with that of the concept description including only one best complex. He found that the simpler truncated descriptions are more accurate. Besides classification accuracy, simpler descriptions require less memory space for storage and less processing time for classification.

During the induction process, if two generalization rules are applicable, the algorithm will select the rule that will generate the simpler feature.

In a noisy domain, it is very likely that no features of D_i^c and D_i^d can be found. The induction algorithm can generate a confident description with features that have high utility values. The confident description can also be used for performing domain tasks such as classification.

We have developed and employed one measure for the control of the quality of advanced abstraction: the **conceptual clustering evaluation measure** (CCEM). Matching factor is a basis for computing the evaluation measures, and is therefore described first.

Any concept, whether a basic concept or a high-level concept, has features that are shared by its instances. A matching factor is a measure to see how similar two concepts are. This similarity is based on the common features and closeness of features and is used for concept clustering.

Let D_1 and D_2 be descriptions of two concepts, then the **matching factor** of the two concepts is computed as follows:

$$MF(D_1,D_2)$$
$$=\sum_{i=1}^{k} VMF(E_{1,i},E_{2,i}) * W_{1,i} * W_{2,i} * CONF(D_1,E_{1,i}) * CONF(D_2,E_{2,i})$$

where k is the number of matched features of D_1 and D_2. For two concepts, if they have more similar features of high relevance and importance, then the two concepts will have a high matching factor, i.e., they are more similar to each other.

$VMF(E_{1,i}, E_{2,i})$ is the **value matching factor** of two features $E_{1,i}$ and $E_{2,i}$, which determines how similar the two features are. $W_{1,i}$ and $W_{2,i}$ are the **feature worths** which give different weights to features based on the domain knowledge and inductive heuristics. CONF is the **feature confidence** which also contributes to weighing the importance of features based on the statistical information.

Generally speaking, if two concepts have more similar features then they are more similar to each other. VMF is used to measure the similarity of corresponding features. VMF allows inexact matching. When the matching features are atomic, VMF is computed as an atom-value matching factor. When the matching features are conjunctions or disjunctions, VMF is computed as the average of all the atom-value matching factors. Computing an atom-value matching factor depends on the type of atoms. If the descriptor is a nominal attribute, the atom-value matching factor is 1.0 when the two values of the matching atoms are the same. Otherwise, the atom-value matching factor is zero. If the descriptor is a linear attribute, the atom-value matching factor is determined by the relative distance of the two values, which is, in turn, based on the value difference and the size of the attribute value domain. If the descriptor is a hierarchical attribute, the atom-value matching factor is 1.0 when the two values are the same; it is zero when none of the two values is an offspring of the other; otherwise it is a value between 0 and 1.0 (determined by the branch factors). If the descriptor is a relation, the atom-value matching factor is always 1.0.

Feature worths $W_{1,i}$ and $W_{2,i}$ are determined by the simplicity of the expressions, relevance of components to the concept and importance of the attributes. Different features may have different contributions to the similarity of two concepts. $W_{1,i}$ and $W_{2,i}$ are used to measure the importance of features in describing a concept. Simplicity of a concept description is important in describing a concept [Michalski 1983b]. Simpler descriptions are easier to comprehend, so they are preferred by humans. For describing a concept, different components may have different relevance measures. The attributes may also be of different importance in describing a concept. For example, in describing a disease, descriptions of some parts of the body may be more relevant to the disease, while descriptions of the other parts of the body may have no relation to the disease. The patient's body temperature may be more important than the weight of the body. In IDB1, an implementation of the KNOWD methodology, we compute the worth value W of each expression in a description as the product of the average atom worth value over n atoms in the expression and the expression simplicity:

$$W=(\frac{1}{n}\sum_{i=1}^{n} AW_i)*SIM$$

where AW_i, the worth of an atomic expression, is defined below:

$$AW_i=K*W_{attrib}*R_{comp}*S_{val}$$

where K is a constant which is chosen to make AW_i to be in the range [0, 10] for all atomic expressions, W_{attrib} is the worth of an attribute, R_{comp} is the relevance of the component, and S_{val} is the simplicity of the value form. It has the highest value for the single-form, a lower value for the range-form, and the lowest value for the or-form. The simplicity of the expression SIM is computed as:

$$SIM = C * \frac{l_{max} - l + 1}{l_{max}}$$

where C is value determined by the form of the expression. It has the highest value for the atomic expressions, a lower value for the conjunctive expression, and the lowest value for the disjunctive expression. l_{max} is a system constant representing the maximal length of expressions (length is measured as the number of atomic expressions), and l is the length of the expression. An expression with a larger SIM value represents a higher simplicity of the expression. From this formula we can see that a shorter expression will have a larger SIM value and that a conjunction has a higher simplicity than a disjunction.

The feature confidence CONF is computed as the sum of the completeness factor FR_1 and the consistency factor FR_2. Only features of high confidence in descriptions of a basic concept will be used for advanced abstraction. In the computation of matching factor the weights of features are closely related to their utility values.

When incorporating a new basic concept represented by its first instance d into a cluster C, the system needs to decide which subcluster of C is the best one for incorporating the new basic concept. To make that decision a measure of the quality of conceptual clustering is needed.

We base the quality of concept clustering on (1) discrimination capability: how well the hierarchy can be used for identifying new instances, (2) characterization capability: how well the hierarchy corresponds to a useful classification, and (3) simplicity: how well the features found in the clusters' descriptions for high-level concepts can be comprehend and used by humans. Our criteria emphasize knowledge about the concept and human oriented descriptions. The KNOWD method uses three factors to measure the quality of clustering:

(1) **discrimination factor** (DF),

(2) **coherence** (COH), and

(3) **description simplicity** (DS).

DF is a measure which represents the improvement of the discrimination capability of all clusters under a current node in the clustering hierarchy. Let cluster C have a set of subclusters: { C_1, C_2, ..., C_k }, so that $C = C_1 \cup ... \cup C_k$, where each subcluster C_i has a description D_i. The average matching factor with respect to C_i is computed as

$$AMF_i = \sum_{j=1, j \neq i}^{k} \frac{MF(D_i, D_j)}{k-1}$$

AMF_i represents the average closeness of D_i to other subclusters directly under C. A higher value of AMF_i means C_i is less distinguishable from other subclusters. Let C_i' be the cluster of C_i incorporated with d; the average matching factor with respect to C_i' is computed as

$$AMF_i' = \sum_{j=1, j \neq i}^{k} \frac{MF(D_i', D_j)}{k-1}$$

After incorporating d, if the average matching factor with respect to C_i' becomes smaller, then the discriminating quality is improved because the subclusters under C are less similar to each other. The amount of improvement is measured by the discrimination factor with respect to C_i which is computed as:

$$DF_i = AMF_i - AMF_i'$$

A higher value of DF_i indicates a greater improvement on the discriminating quality of classification with instance d under subcluster C_i, which is shown in Figure 3.8. If each time we select C_i with the maximal DF_i to incorporate an instance, the resulting concept hierarchy would be better to classify new instances. When incorporating a new instance into the concept hierarchy, we want the quality of the whole hierarchy, not just one cluster, to be improved. The discrimination factor is an important quality measure which considers not only individual nodes in the hierarchy but also a large range of surrounding clusters.

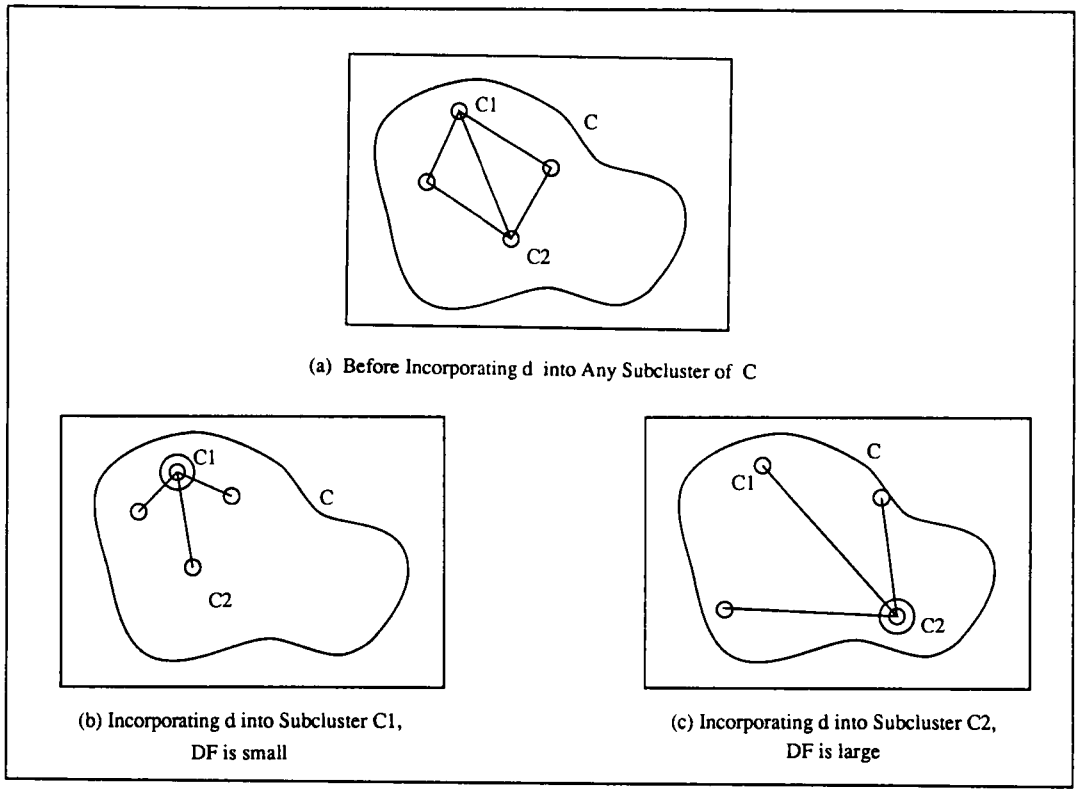

Figure 3.8. Comparison of DF

A new instance should be incorporated into a cluster that is highly similar to the instance. In order to get a better clustering, we also need to know how well an instance is fitted in cluster C_i, which is measured by coherence COH. COH is computed as **cluster matching factor** (CMF) minus **conflicting factor** (CF). Let $\{S_1, S_2, ..., S_m\}$ be a set of child nodes of C_i, then the cluster matching factor of C_i and d is

$$CMF_i = \sum_{j=1}^{m} \frac{MF(d,D_{S_j})}{m}$$

where D_{Sj} is the description of S_j. Using CMF_i instead of $MF(d,D_i)$ can be more accurate to represent the similarity between d and C_i because when generalizing descriptions of $S_1, ..., S_m$ to

the description of C_i, features may be generalized or eliminated. CF_i is the number of conflict matching features between d and C_i. Two matching features are in conflict when their VMF is less than the feature conflict threshold FC-TH.

As the results of the conceptual induction, concept descriptions should be highly human-oriented and easily understood. The description simplicity DS is another criterion to measure the quality of descriptions, and is computed as the reciprocal of description complexity which is determined by the following criteria[9]:

(1) A description with more or-form expressions is more complex than a description with fewer or-form expressions.

(2) A description with longer average or-form expressions is more complex than a description with shorter average or-form expressions.

(3) A description with more and-form expressions is more complex than a description with more atomic expressions.

(4) A description with longer average and-form expressions is more complex than a description with shorter average and-form expressions.

Features in a description are in three types of forms: atomic expressions, conjunctive expressions and disjunctive expressions. Let N_t be the number of atomic expressions in the concept description, N_c be the total number of atomic expressions in all the conjunctions of the concept description, and N_d be the total number of atomic expressions in all the disjunctions of the concept description. Let $N_t = N_a + N_c + N_d$. In IDB1, an implementation of the KNOWD methodology, the description simplicity is computed as:

[9] The criteria given here are general guidelines. Computational implementation may have different methods.

$$DS=C_0*\frac{2*N_t-N_c-2*N_d}{2*N_t}$$

where C_0 is a constant to add a weight to the simplicity in the clustering evaluation. When all expressions in the description are atomic, DS has the maximal value, and when all expressions in the description are disjunctions DS has the minimal value.

For optimizing the quality of clustering, we need to find the M each time we decide which subcluster should incorporate a new instance, such that

$$DF_M+COH_M+DS_M = \max_i\{DF_i+COH_i+DS_i\}, \quad for\ i=1,...,k$$

After identifying M, we incorporate the new concept represented by d into the subcluster C_m.

As in all incremental induction systems, the quality of induction is affected by the order of instances presented to the system. In the KNOWD methodology, this effect is reduced by two techniques. The first one uses domain knowledge as the guidance for the induction process so that random search can be avoided and useful important features and relationships can be found. The second technique allows backtracking in the advanced abstraction. Features will be dropped from the concept descriptions when they become of low utility, and clusters will be eliminated from the concept hierarchy when they cannot represent useful concepts that characterize the instances under them with high-quality features.

3.2.7. Structure and Functionality of the Inductive Database System

In this section we describe the structure and functionality of our inductive database system and explain why we select this structure and functionality.

We build our induction system as a layer above the object-oriented database systems as shown in Figure 3.9. Through the **user interface** the user can start an induction process or query the database. Queries can be of two types: (1) general descriptions of a class of data objects, generated by inductive inference; and (2) specific facts of a data object, as usually provide by a database system. There are two modes for users to view the descriptions of concepts: query

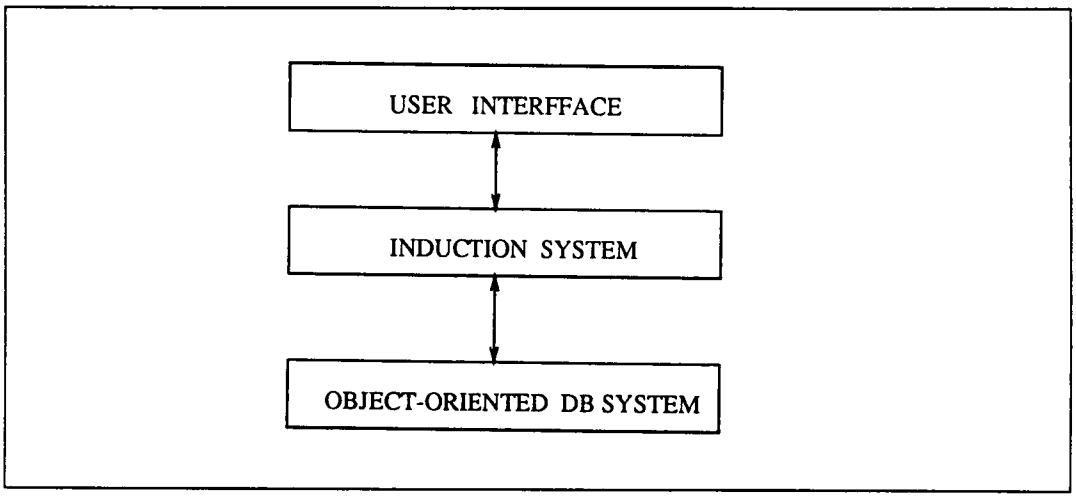

Figure 3.9. The Inductive Database System

mode and navigation mode. In the query mode users type in queries in the query language. In the navigation mode users view concepts and clusters through the concept hierarchy in an interactive window. Every object in the hierarchy can be viewed directly. The **object-oriented database system** serves as a persistent storage of data objects and knowledge used or generated by the induction system. The underlying object-oriented database provides an interface programming languages. With this structure, an ordinary database system is extended with an inductive inference capability, and the resulting inductive database includes all the facilities provided by the ordinary database.

The structure of the **induction system** is shown in Figure 3.10. It consists of several components:

Induction Engine. It is the main component which controls and performs the inductive inference.

Query Processor. It interprets user's queries and searches through the concept generalization hierarchy, and collects and returns the answers.

Knowledge/Bias Management System. It supports facilities to manage domain knowledge and inductive biases that constrain and guide the induction. Different types of objects (including

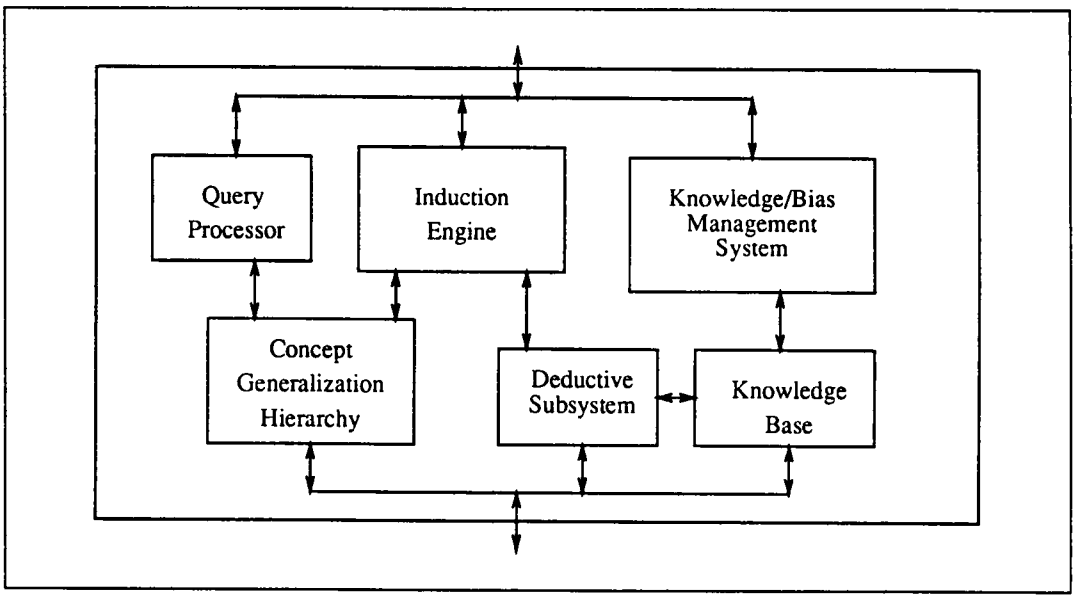

Figure 3.10. The Induction System

attributes, components, concepts, clusters, generalization rules, and deductive rules) can be selected in an interactive way. Various operations (including enable/disable a rule, and delete/add/modify a value) can be performed. Contents of the objects can be displayed through windows. Detailed discussion of the knowledge and bias management system is given in Appendix B.

Deductive Subsystem. It provides deductive capabilities which support the induction.

Knowledge Base. It contains inductive rules, deductive rules, inductive biases and various domain specific knowledge.

In this structure we emphasize the separation of induction engine and knowledge/bias components. We believe that there is no general induction system that can be both effective and efficient for all applications. For better performance an induction system must utilize domain specific knowledge and biases to guide inductive processing. The separation structure together with the knowledge/bias management component makes it possible to exchange domain knowledge and biases easily so that our induction system can be applied to many domains with good performance.

Concept Generalization Hierarchy. The results of induction are kept in the concept generalization hierarchy as shown in Figure 3.11. This hierarchy is a tree structure with leaves representing instance objects which keep the specific data of each instance. Nodes on the next higher level represent basic concepts. Basic concepts are concepts into which those instances are preclassified. Each basic concept represents the generalization of all the instances under it. Above the basic concept are one or more levels of cluster nodes that represent higher-level concepts which are generalizations of basic concepts under them. The root is the generalization of all instances in the tree.

There may be more than one concept generalization hierarchy in the inductive database. One instance belongs to only one basic concept in a concept hierarchy, but this does not prevent it from belonging to another basic concept in a different concept hierarchy. Since instances are stored as objects in the object-oriented database, there is not storage duplication of an instance belonging to two basic concepts because only the object identifier is stored in its parent objects.

The reasons to select a hierarchical structure to represent concepts and instances are that (1) hierarchical structures are the most common structures in nature and human society, so they can be applied to more situations; (2) hierarchical structures are simple, so they are easy for people to understand and manipulate.

3.3. Summary

The major results of this research work are the KNOWD model and methodology of conceptual induction.

In this chapter we have given a formal description of the KNOWD model. From the analysis we see that the KNOWD model supports a combination of a rich representation capability with incremental induction processing. A formal description of the KNOWD methodology and detailed description of the algorithms are also given in this chapter. This methodology has several unique features which include multilevel abstraction, knowledge-directed induction and comprehensive bias management. With those unique features the KNOWD methodology can be

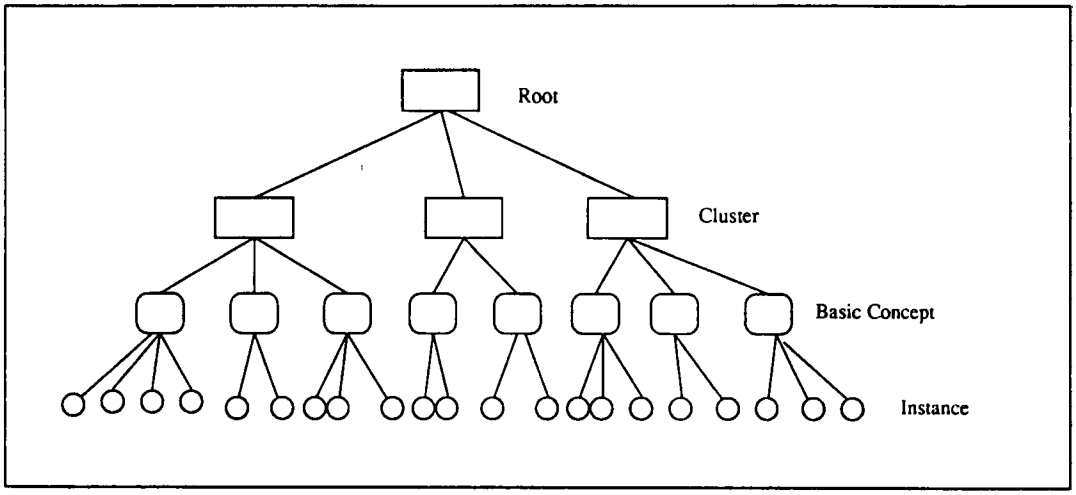

Figure 3.11. The Concept Generalization Hierarchy

applied to many domains.

In next chapter, we discuss the applications of the KNOWD methodology, perform various experiments, and evaluate its performance.

CHAPTER 4

APPLICATIONS AND EVALUATION

The KNOWD model and methodology is implemented in IDB1, an inductive database management system. In this chapter, we first discuss an application of IDB1 in the domain of Space Shuttle Main Engine (SSME) test analysis. We then discuss experiments of IDB1 in the application domain of chemical process monitoring. This is followed by tests of IDB1 in the power electronics control application. Finally, we present a summary of the tests and an evaluation of IDB1.

4.1. Space Shuttle Main Engine Test Analysis

Equipment test analysis is one of several application areas of IDB1, where conceptual descriptions about different faults can be automatically generated from a large number of fault instances, and similar faults can be classified into clusters. The inductive results including high level characteristic descriptions and discriminant descriptions of faults and the concept hierarchy with descriptions of clusters can be used to aid the fault test analysis and used for fault diagnosis.

SSME is one of the most complex reusable liquid-fuel (oxygen and hydrogen) rocket engines. Each time a test on SSME is performed, a large amount of data is collected from many sensors. Many highly-trained engineers are required to perform a thorough investigation of the test. Two difficulties are encountered in trying to improve test analysis methods: (1) As more tests are performed and more thorough investigations are required, more experienced engineers are needed. (2) More senior staff are leaving with their many years' experience. To overcome those difficulties, a computer conceptual induction technique is used to aid the engineers in analyzing the test data. In addition to its efficiency in forming concepts and generating concept

descriptions, a computer inductive system can accumulate knowledge from both the data of many tests and the expertise of the engineering staff.

We have instantiated IDB1 into the SSME Test Inductive Database (ETID) which is used for SSME fault test analysis. After SSME fault data is entered into ETID, a concept hierarchy is built by the inductive system. The concept nodes on the hierarchy represent various engine faults. The cluster nodes on the hierarchy represent high-level concepts, each description of which describes a group of similar engine faults. Descriptions of engine faults or fault groups are also generated by the inductive system and stored in the database. Features about any attributes, sensors or relationships can be easily accessed by a user. The expertise of the engineering staff can be incorporated into the system as concept constraints, deductive rules and various inductive biases.

4.1.1. Data and Domain Knowledge

The SSME domain is a good example to illustrate the use of domain knowledge in conceptual induction and to show that IDB1 can utilize domain knowledge for improving induction quality.

SSME tests generate the raw data about sensors for a fault. The raw data is simply a list of time-value pairs of each sensor. All values are real numbers. Usually this kind of data is used to plot diagrams for representing sensor behavior, and then human experts analyze the diagrams to find the characteristics of each fault. This human analysis process is time-consuming when the number of diagrams is large, and is complicated when the features of faults involve relationships between sensors. Since a human expert describes the features of a fault by a set of attributes, ETID will automatically generate those human-oriented descriptions from the raw data.

Before sending the raw data to the induction system, preprocessing is performed which smooths the curves of sensors, divides the curves of sensors into segments and denotes each segment as an event. The collection of sensor descriptions constitutes the raw description of an instance of an engine fault which, in turn, is used as the input data for the induction process. ETID emphasizes a significant event for each sensor, since most characteristic features of a fault

exist in the significant event. The preprocessor extracts attribute features from the raw data as shown in Table 4.1, and further deduction will generate more attribute features to describe a fault.

In building ETID, various domain knowledge is needed to initialize IDB1. The domain knowledge includes: attribute definitions, transformation rules, constructive rules, concept constraints, and various inductive biases.

The description of a basic concept is an abstraction of a class of real world entities which share common features. A real world entity is a thing, such as an animal, or a computer; or a situation such as a disease, a machine fault, or a state of a process. In the SSME domain we assume each fault instance belongs to only one type of fault. The basic concepts should be the main focus of an application domain when the purpose is to find the features of the basic concepts. In the SSME fault test analysis domain, the purpose is to find features for each fault and to find a possible classification of faults. Engine faults correspond to basic concepts.

Table 4.1. Attributes Directly Extracted from Raw Data.

ATTRIBUTES	EXPLANATIONS
START-TIME	starting time of an event
END-TIME	ending time of an event
RATE	average changing rate of an event
DIRECTION	changing trend of an event
MAGNITUDE	the absolute amount of change of an event
AVG-VALUE	the average value of an event
VALUE-TO-NORMAL	indication of whether the average value of an event is above, same of or below the normal value
CURVE-PATTERN	indication of the general pattern of the curve
STABLE-LEVEL	the value of the sensor after it became stable
STABLE-LEVEL-TO-NORMAL	indication of whether the stable value of a sensor is above, same of or below the normal value
NUM-OF-EVENT	the number event for a sensor
DURATION	the difference between the START-TIME and END-TIME

A structural concept is a concept with instance descriptions which contain component features and relationships between components. The description of a structural concept contains both features of and relationships between components. In a flight engine fault test analysis domain, an engine fault is described by features such as temperature, pressure, flow, and speed. These parameters are measured by many sensors; therefore, sensors are the components.

In a multi-concept inductive system, components usually have different relevance toward different concepts. For example, a sensor may have distinct features for an engine fault and provide no information about other faults. A large number of sensors exists in the flight engine, but only a few of them are related to a specific fault. To pay equal attention to all sensors for every fault is inefficient; therefore, a component in IDB1 can be assigned a different relevance for each concept, where a larger value represents a higher relevance. In building ETID, the assignment of sensor-fault relevance measures depends on domain knowledge such as functional relationships and structural relationships of the engine parts and locations of sensors and faults. In ETID sensors can be one of three types with respect to each type of engine fault: critical sensors, which show strong evidence of and are closely related to the fault; irrelevant sensors, which do not show any changes when the fault occurs; and unspecified sensors, which may show some change with the fault and whose relationship to the fault is unknown. Critical sensors are assigned a high relevance measures; irrelevant sensors are given a low relevance measure, and unspecified sensors are given a value between those two values. By domain knowledge we know that pressure sensors are usually more important for fault identification than temperature sensors. Thus, pressure sensors are given higher relevance measures than temperature sensors.

Relationships between components play an important part in the description of concepts. Components may have positional relationships, temporal relationships, or some relationships governed by domain theory. IDB1 allows a system developer to indicate interesting component pairs. Then, deductive rules, which derive component-relationship descriptions from component features, are automatically generated. Examples of these kinds of rules in ETID are shown in Figure 4.1. Known relationships between component features can be used as concept constraints.

```
(d-rule4 if      ((direction (?comp1) = pos)
                  (direction (?comp2) = pos))
         then    add ((increase-together (?comp1 ?comp2))))

(d-rule5 if      ((direction (?comp1) = neg)
                  (direction (?comp2) = neg))
         then    add ((decrease-together (?comp1 ?comp2))))

(d-rule6 if      ((direction (?comp1) = pos)
                  (direction (?comp2) = neg))
         then    add ((opposite-trend (?comp1 ?comp2))))

(d-rule7 if      ((start-time (?comp1) = ?st1)
                  (start-time (?comp2) = ?st2))
         then    add ((change-concurrently (?comp1 ?comp2))))
```

Figure 4.1. Deductive Rules for Generating Component-Component Relationships

Attributes are the basic vocabulary used to describe basic concepts. After the identification of basic concepts, the system developer needs to find out what attributes should be used in modelling the domain problem. What attributes are usually used to describe a basic concept is determined by domain experts. Attributes determined by domain experts may not be available, so the constructive rules should be formulated so as to derive the unavailable attributes from the available attributes. In IDB1 attributes are of different importance in describing a concept. The importance of an attribute is denoted by its worth value. Domain knowledge about the relative importance of attributes can be used to assign different worth values to different attributes. For example, in the SSME domain, the attribute **direction** of change is more important than the attribute **rate** of change because a sensor's direction of change is usually the same for the same fault while the rate of change can be different for different severities and durations of the fault. In many cases, relationships between attributes are more important than individual attributes. IDB1 represents every attribute by an object and supports the knowledge acquisition facilities to help the system developer to define attributes.

Since useful attributes may not be available directly from the raw data, deductive rules are used to derive them by applying various domain knowledge such as domain theory, physical laws, operational principles and domain experience. From domain knowledge in the flight engine test, we know that the attributes "starting time", "ending time", "rate of change", and "magnitude" have little value for characterization of an engine fault, because they all change with the severity or duration of an engine fault. Different faults may have the same changing rate, and faults of the same type may have different rates of change. Some relationships have a higher importance in characterizing engine faults. For example, a temperature sensor and a pressure sensor at the same location of the engine have certain relationships for a specific fault. New attributes START-TIME-DIFF (difference of the starting time) and END-TIME-DIFF (difference of the ending time) are used to represent the temporal relationships; RATE-RATIO (ratio of two rates of change) and MAGNITUDE-RATIO (ratio of the magnitudes) are used to represent the quantitative relationships. The derived attributes in ETID are shown in Table 4.2.

Logical relationships, like concurrency of trends, are represented by the logical connective AND. In the domain of SSME test analysis, human experts recognize certain features and relationships for different faults. This kind of knowledge can be used to constrain concepts. For example, (assuming the engine is in an open loop situation) an increasing pressure at a valve inlet will cause an increasing pressure at the valve outlet. This rule is applicable to all valve blockage faults. In ETID this rule is expressed as:

IF (direction (s23) = ?x)

THEN (direction (s27) = ?x)

and

IF (direction (s23) = ?x)

THEN (direction (s28) = ?x)

where s23 is the pressure at the outlet of the high pressure oxidizer pump booster which is also the inlet to FPOV (fuel preburner oxidizer valve) and OPV (oxidizer preburner oxidizer valve), s27 is the pressure of the outlet of OPV, and s28 is the pressure of the outlet of FPOV.

Table 4.2. Derived Attributes in ETID.

ATTRIBUTES	EXPLANATIONS
INCREASE-TOGETHER	both events are rising together
DECREASE-TOGETHER	both events are falling together
OPPOSITE-TREND	two events have the opposite trend
CHANGE-CONCURRENTLY	two events have the same starting time
TEMPORAL-REL-TYPE1	two events have the same starting time and the ending time
FIRST-START-TIME	the event has the earliest starting time
ACTIVATED-BEFORE	one event activated before the other
LARGEST-DURATION	the event has the largest duration
RATE-RATIO	the ratio of two events' rates
MAGNITUDE-RATIO	the ratio of two events' magnitudes
AVG-VALUE-RATIO	the ratio of two events' AVG-VALUE
STABLE-VALUE-RATIO	the ratio of two events' stable level
START-TIME-DIFF	the difference of two events' starting time
END-TIME-DIFF	the difference of two events' ending time

4.1.2. Experiments and Discussion

Employing domain knowledge for better results in conceptual induction is an important capability of the KNOWD method. Attribute worth values and component relevance measures are important means to embody domain knowledge. In many application areas, the attributes used to describe concepts have different importance, and components have different relevance measures for different concepts. When comparing two concepts, it is worthwhile to pay more attention to the important attributes and relevant components. Many conceptual induction methods do not differentiate the importance of attributes and relevance of components. However, we expect that the resultant concept hierarchy and descriptions would have a higher quality if attribute worths and component relevance are considered, because attributes with high worth values tend to show high completeness, and features of more relevant components tend to show high consistency.

The purpose of this experiment is to confirm the expectations above. SSME simulation data about five valve-blockage faults are used. The input data are representative of the typical entities in practice because structural concepts, relationships among attributes and components and

many types of attributes are included. To examine the effect of attribute worth and component relevance to the quality of the concept descriptions, we compare the relative matching factor for each pair of faults by employing the data of instances in two cases. The relative matching factor of two basic concepts C_1 to C_2 is computed as the square of the matching factor of C_1 with C_2 divided by the product of matching factor of C_1 with itself and matching factor of C_2 with itself:

$$RMF(D_1,D_2)=\frac{MF^2(D_1,D_2)}{MF(D_1,D_1)MF(D_2,D_2)}$$

where MF is defined in subsection 3.2.6.

In the first case we do not differentiate attribute worths and component relevance and let all attributes have the same worth value and all sensors have the same relevance to all faults. In the second case, attributes have different worth values and sensors have different relevance values based on whether a sensor is a critical sensor to a fault, an irrelevant sensor or an unspecified sensor. The relative matching factors of faults are computed for evaluating the quality of induction.

The relative matching factor values computed for this experiment are shown in Table 4.3 and Table 4.4. The larger matching factor of a fault with itself means that more features of high confidence are found for the fault, and a smaller matching factor of a fault with another fault means that the fault descriptions have a higher discrimination capability. A comparative study of Table 4.3 and Table 4.4 illustrates a significant reduction in the value of the relative matching factor of all pairs of different faults when different worth and relevance values are used. The "Row AVG." is the average of all the relative matching factors of pairs of different faults on each row, and "Total AVG." is the average of the relative matching factors of all different faults. The lower values of the "Row AVG." and "Total AVG." indicate higher quality of concept descriptions. The comparison of values of the "Row AVG." and "Total AVG." in Table 4.3 and Table 4.4 shows significant improvement of the concept descriptions globally.

Table 4.3. Relative Matching Factors of the Concepts without Domain Knowledge

	OPV	MOV	CCV	FPOV	MFV	AVG.
OPV	1.0	0.1195	0.1087	0.5097	0.1389	0.2192
MOV	0.1195	1.0	0.2990	0.1611	0.1662	0.1864
CCV	0.1087	0.2990	1.0	0.1543	0.1826	0.1862
FPOV	0.5097	0.1611	0.1543	1.0	0.1621	0.2468
MFV	0.1389	0.1662	0.1826	0.1621	1.0	0.1624
					TOTAL AVG.:	0.2002

In Table 4.5 we use two other factors to show the improvement of induction quality: the number of confident features in each concept description and the average feature confidence in each concept description. A concept description having more confident features with higher confidence values is of higher quality. Case 1 corresponds to not using domain knowledge, and case 2 corresponds to using domain knowledge. From Table 4.5 we can see that all the concept

Table 4.4. Relative Matching Factors of the Concepts with Domain Knowledge

	OPV	MOV	CCV	FPOV	MFV	AVG.
OPV	1.0	0.0063	0.0073	0.0366	0.0030	0.013
MOV	0.0063	1.0	0.0011	0.0070	0.0053	0.005
CCV	0.0073	0.0011	1.0	0.0	0.0132	0.005
FPOV	0.0366	0.0070	0.0	1.0	0.0305	0.019
MFV	0.0030	0.0053	0.0132	0.0305	1.0	0.013
					TOTAL AVG.:	0.0110

Table 4.5. Quality Improvement of Concept Descriptions

CASES		OPV	MOV	CCV	FPOV	MFV
1. Conf. Feature #	24	26	23	29	20	
Avg. Confidence		0.7497	0.7240	0.6823	0.6096	0.6756
2. Conf. Feature #	51	51	34	41	30	
Avg. Confidence		0.9132	0.9005	0.9001	0.7943	0.8643

descriptions have been improved. For example, the number of confident features in concept OPV has increased from 24 to 51, and the average feature confidence has increased from 0.7491 to 0.9132.

The improvement of induction quality is also shown in the concept hierarchy. With instances as basic concepts, the concept hierarchies of the two cases (described above) are shown in Figure 4.2 and Figure 4.3.

Here we performed unsupervised induction. In the figure the labels of the leaves, such as opv12, indicate the real types of faults those instances belong to. However, this information is not input to the induction system. We let the induction system group those instance into meaningful clusters. Comparing the two concept hierarchies, we can see that in the first case, many instances of different types of faults were grouped together on the second and third levels of clusters, while in the second case, no faults were grouped with different types of faults on each level. On the second level of the hierarchy, the clusters in the second case show more regularity than the first case. In Figure 4.3, Cluster246 corresponds to the MFV blockage fault, cluster253 corresponds to the CCV blockage fault, cluster254 corresponds to the MOV blockage fault, cluster249 corresponds to the OPV blockage fault and cluster248 to FPOV blockage faults. In Figure 4.2, the correspondence between clusters and faults is not as clear because none of the clusters contains all the instances of any single fault. Furthermore, some clusters, like cluster422, do not contain

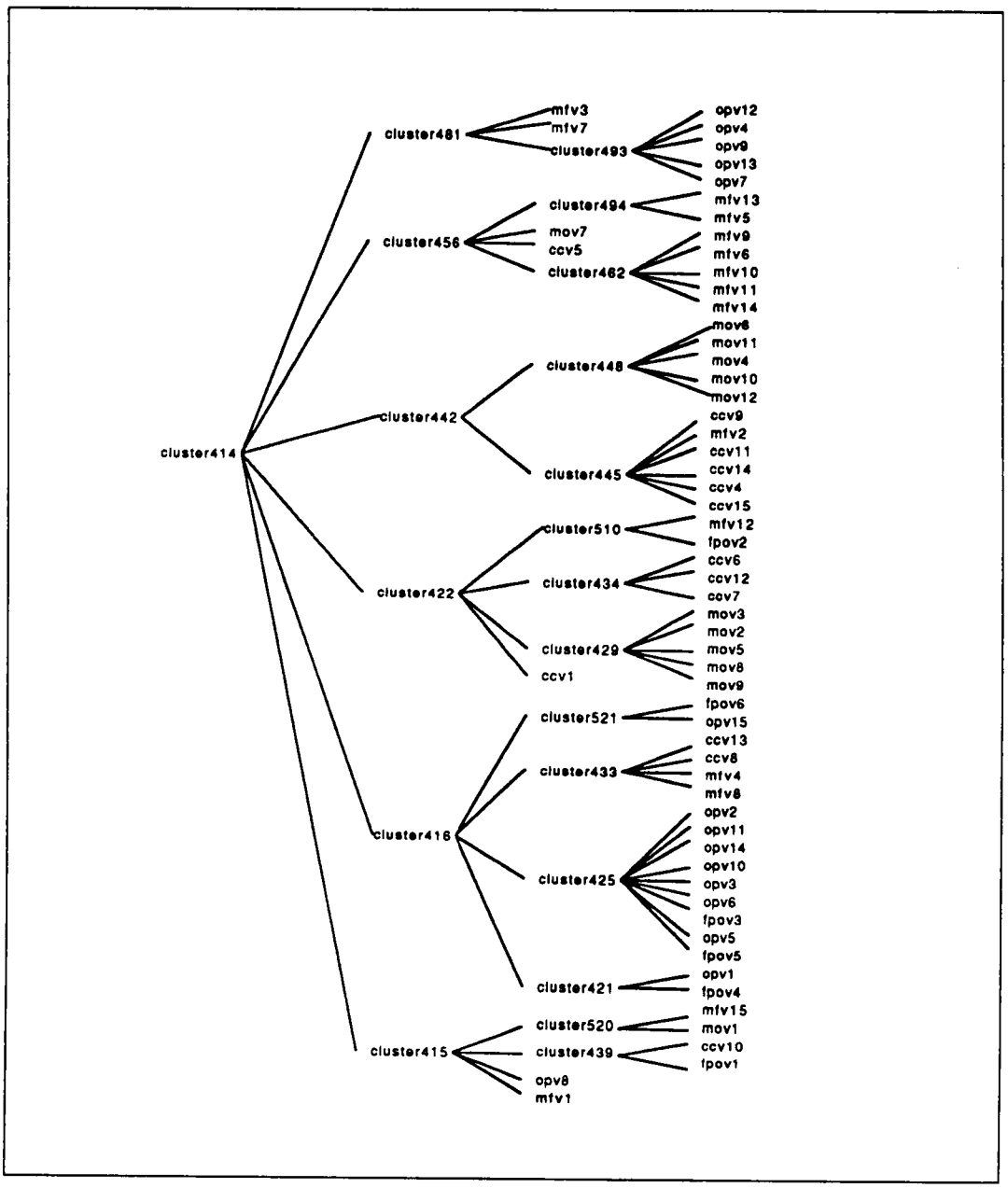

Figure 4.2. Hierarchy of Induction without Domain Knowledge

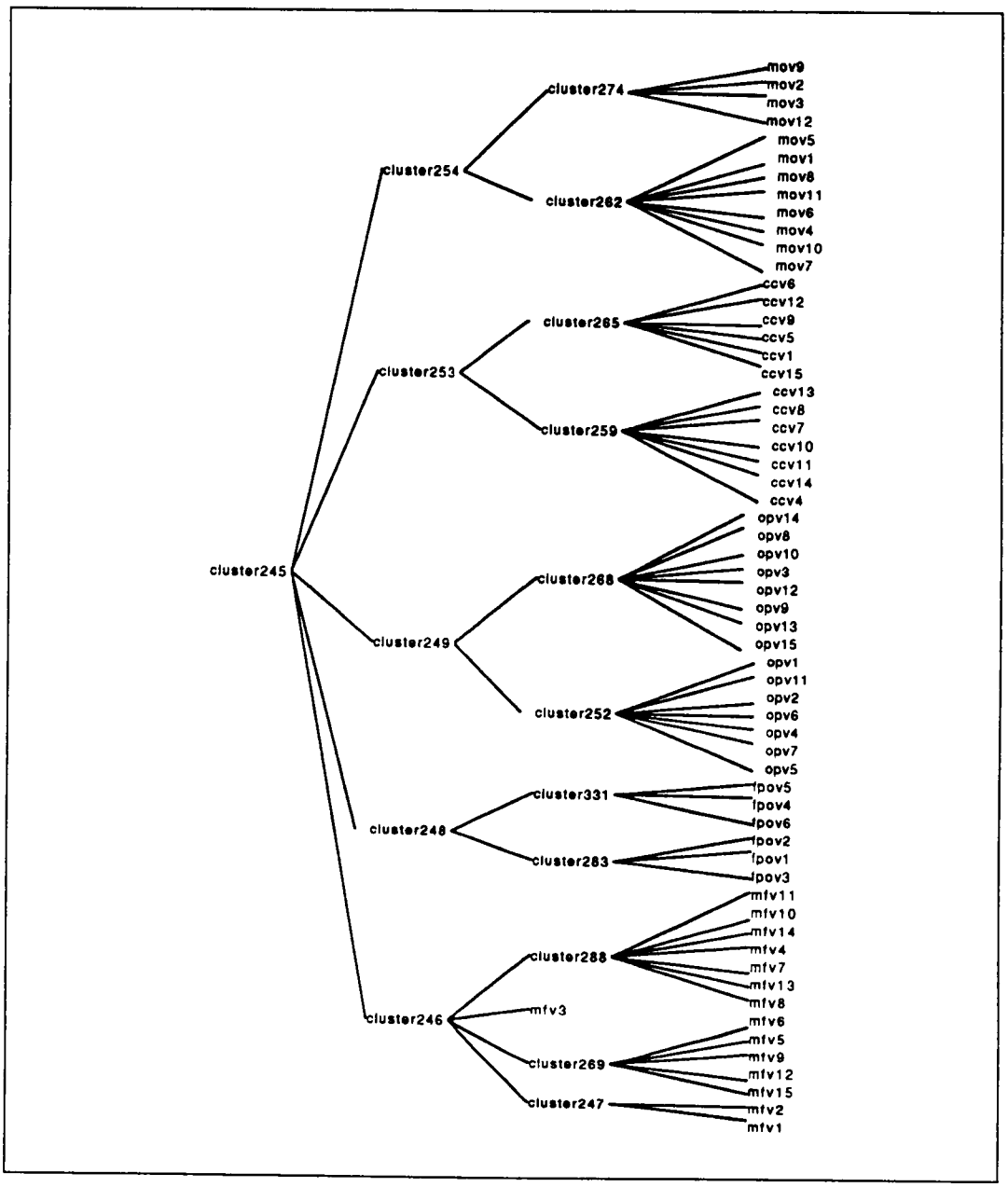

Figure 4.3. Hierarchy of Induction with Domain Knowledge

the majority of instances of any fault. For example, cluster442 contains the instances of MOV and CCV blockage faults which have close percentages; both cluster442 and cluster422 contain the highest percentage instances of the same MOV fault. Each cluster is associated with the type of fault which has the highest percentage of instances in the cluster.

The comparison of error rates of the two cases is shown in Table 4.6. In the first case, 33 instances were misgrouped for the second level clusters, and the error rate is 54.1 percent; 11 instances were misgrouped for the third level clusters, and the error rate is 20.4 percent. By using the domain heuristic knowledge associated with attribute worth and component relevance, these error rates are reduced to zero.

Table 4.6. Comparison of Clustering Error Rates

	Without Domain Knowledge	With Domain Knowledge
Error Rate on The Second Level	54.1 %	0 %
Error Rate on The Third Level	20.4 %	0 %

4.2. Chemical Process Monitoring

In this section we apply IDB1 in a chemical process monitoring domain to test the applicability of the KNOWD methodology and analyze its properties. The application is automated process diagnosis. We first describe the application domain. Then, we discuss several experiments of IDB1 performed on this domain.

4.2.1. Domain Background

Inductive inference programs can be applied to a historical database of a continuous stirred tank reactor (CSTR) which carries out an exothermic reaction of material A reversibly going to

material B [Wang et al, 1992]. The results of the induction programs are diagnostic information which can be used to improve the control of the chemical process.

The diagnostic information is used for online monitoring of process operation and generating accurate reports of events to the operations personnel, so that they can take rapid and correct intervention in case of serious perturbations, such as a stuck valve, heat transfer surface fouling, inlet concentration change, inlet temperature change and the presence of inhibitory factors.

The operating objective is to maximize the rate of production of material B. The reaction is controlled to maximize the concentration of B. The status of the CSTR system can described by several variables:

C_A the concentration of material A

V the reactor volume

T reactor temperature

T_j jacket coolant water temperature

T_A the ambient temperature

F the reactor flow rate

F_j the jacket coolant flow rate

In order to get a high concentration of material B, the reactor temperature T must be controlled at a proper point. The excess heat generated from the reaction is removed by a jacket through which cooling water flows. The reactor volume V also must be controlled. This is accomplished through control of the reactor flow rate, F. Thus, the reactor flow rate F and the jacket coolant flow rate are the two manipulated variables.

The raw process data is derived from simulation of a CSTR. The original data set is reduced to a smaller dimensional space of data consisting of mutually independent variables through a statistical data reduction procedure and yields a database of attribute values versus time. There are four variables denoted as p1, p2, p3 and p4. Each instance of the CSTR status is represented by the values of the four variables. Since the failure models are programmed into the

simulation, known class values are available for each entry of the database. The objective is to construct descriptions of the objects in each class using automatic procedures.

In the induction system, preprocessing is performed to divide the value range of each variable into ten equal subranges. There are no nominal and hierarchical attributes. All variables are considered linear attributes, and each subrange is represented by an integer.

Three states of the operating process are considered:

(1) nominal operation,

(2) insulation failure, and

(3) concentration change.

The three states are the concepts to be learned by the induction system. IDB1 generates the concept descriptions for each of the states.

The quality of the automatically generated descriptions can be evaluated in two ways. First, the available data can be divided into training and validation sets of independent measurements. Descriptions derived from the training set can be used to classify elements of the validation set, and a score can be derived based upon the percent of correct classification. This score can be compared with scores for classification derived using neural network and decision tree technology, which is available from independent research [Wang et al. 1992].

A second approach to validation is to attempt to determine if the descriptions make sense in terms of the physics of the process. One of the failure modes corresponds to failure of a portion of the thermal insulation blanket which surrounds the reactor. This causes the reactor to lose thermal energy due to heat transfer to the environment more rapidly than normal, which causes a greater than normal consumption of fuel for heating. The other failure mode is a large variation in the composition of the raw material entering the CSTR. This would affect product composition were it not for the action of the closed loop control system. The effect of this failure mode should be most noticeable in the control actions (value positions) applied to the reactor.

Correct identification of these physically-based shifts in the data by manual examination of the automatically generated descriptions would be a significant validation of this approach

toward intelligent database design. The identification problem is somewhat complicated by the initial statistical reduction step; however, that is essentially a linear transformation of the data into a lower-dimensional space and can be relatively easily un-done by examination of the regression coefficients.

Further work could be done using this example by varying the separation of the failure modes, This requires modification of the simulation used to generate the database and was deemed beyond the scope of this research. Such work would, however, provide an indication of the robustness of the technology to class separability. We address this issue in the power electronics application, where there is an easier mechanism for control of error rates. Although not quite the same properties are evaluated, intuitively, a relationship should exist. We leave further study of robustness issues for future research.

4.2.2. Experiments and Tests

In this application, little domain knowledge is available. We do not differentiate the attributes' importance. There are no new constructive attributes and concept constraint rules.

One experiment is performed to test the classification capability of the learned concept descriptions by IDB1. In this experiment, 2580 instances of the CSTR system states are provided to IDB1 for generating concept descriptions. Then another 2580 unseen instances are classified based on the learned descriptions. The results are shown in Table 4.7. In all three cases, all the instances are correctly classified.

Table 4.7. Result of Classification

CASES	# OF OBSERVATIONS	% CORRECT
Nominal	1680	100
Insulation Failure	450	100
Concentration Change	450	100

The ability to generate symbolic description about the operation states is a highly desired feature for an induction program in the chemical process monitoring application. The concept descriptions generated are shown in Figure 4.4. Three types of descriptions are shown in the figure: characteristic descriptions, discriminant descriptions and confident descriptions. Features in the confident description may not satisfy completeness or consistency conditions, but they have high completeness and consistency factors.

From the generated concept descriptions, people can easily understand features for each case. Discriminant descriptions are use for classification of unseen instances. Whenever this cannot be done, confident descriptions are used to compute similarities between concepts and an instance for classification.

One of the primary requirements for the classifier is that it must be modifiable online as a consequence of reception of new data. It is important for an induction method to be incremental in this application. IDB1 can incrementally modify its concept descriptions and improve its classification performance, which is shown in the following experiment.

In this experiment instances of the chemical process states are inserted in the database in an incremental way. After incorporating 20 instances, the concept descriptions generated by IDB1 are used to classify 2580 unseen instances. The average rate of correct classification is 53.9 percent. Then 20 more instances are incorporated. Again the modified concept descriptions are used for classification. This time the average rate of correct classification is 76.5 percent. We can see that the concept descriptions are improved. More experiment steps are performed and the results are shown in Figure 4.5. From the diagram, we can see that the concept descriptions are improved incrementally with more instances incorporated. After incorporating 480 instances, all 2580 unseen instances can be correctly classified.

From the above experiments, we can see that IDB1 can be successfully applied to the CSTR monitoring domain to generate useful concept descriptions for classification of unseen instances. IDB1 can improve the concept descriptions through incremental induction. It can be

CONCEPTS	Characteristic Descriptions	Discriminant Descriptions	Confident Descriptions
NOMINAL		(P4 = 0..4) (P1 = 0V4V5)	(P1 = 1..3) (P4 = 1..3) (P2 = 6..8) (P1 = 4..5)
INSULATION FAILURE	(P4 = 8..10) (P1 = 1..3)	(P4 = 10)	(P4 = 8..10) (P1 = 1..3)
CONCENTRATION CHANGE	(P1 = 8..10)	(P3 = 10) (P2 = 0..1) (P4 = 6..7) (P1 = 8..10)	(P1 = 8..10) (P4 = 6)

Figure 4.4. Concept Descriptions

applied to a domain with little domain knowledge. However, in the absence of domain knowledge the quality of concept description would be lower in comparison with using domain knowledge. This point will be shown in next section.

4.3. Power Electronic Equipment Control

In this section we apply IDB1 to the domain of power electronic equipment control. In this domain data has little regularity. We try to test the induction capability of IDB1 in this domain. Noisy data was used to test the robustness of IDB1's algorithm.

4.3.1. Domain Concepts and Data

Inductive inference methods have been applied to the control of power electronic equipment [Birdwell et al 1992]. A decision tree method was used to synthesize the control sequence in real-time. The control signal is a fixed period binary value. Their research results have demonstrated success in term of improvement of control performance.

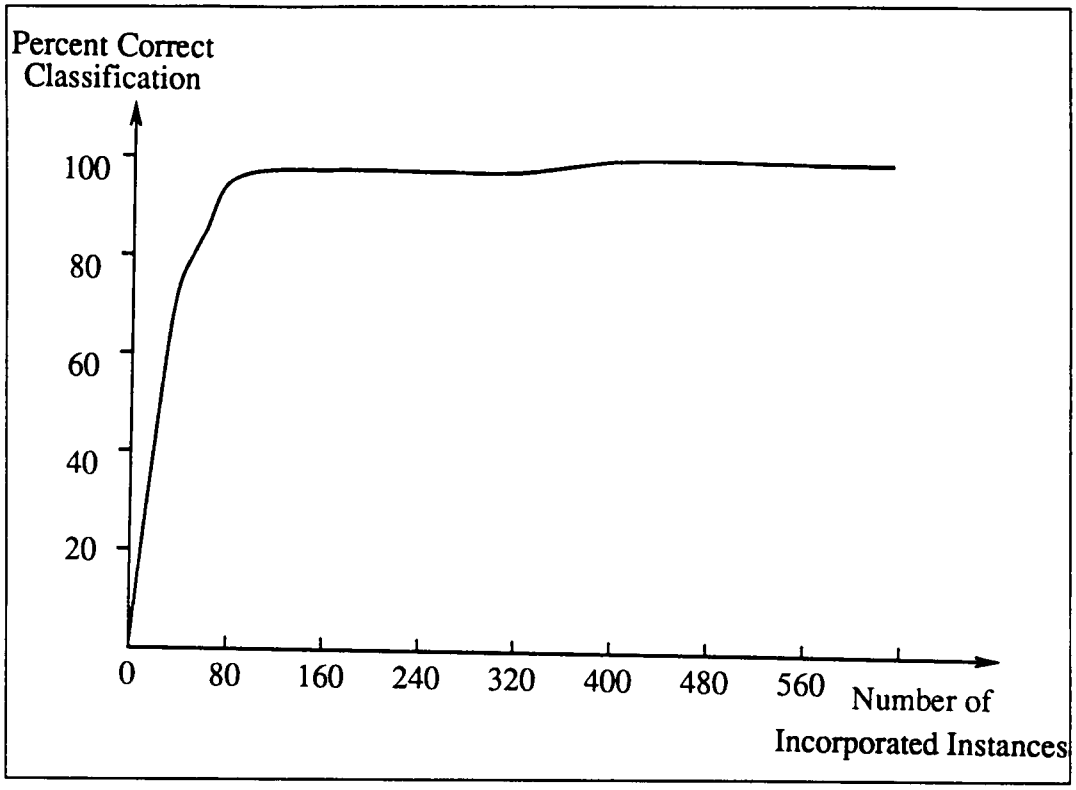

Figure 4.5. Incremental Improvement of Classification

In the domain of power electronic equipment control, a modulator is used to generate binary control signals (0's and 1's) to drive a motor. Input to the modulator includes:

measured motor currents	I_a, I_b, I_c
current references	I_a^*, I_b^*, I_c^*
measured motor speed	w

The output from the modulator is three binary variables (b_1, b_2, b_3). The three binary variables may take one of the seven combinations: (000, 001, 010, 011, 100, 101, 110). A classifier procedure based on the concept descriptions generated from the induction system can be used as a modulator for variable speed control of a motor.

The task of induction in this domain is to generalize the samples of the input variables for each of the seven output values. Therefore, the basic concepts to learn correspond to the seven

output values; we can name them case0, ..., case6. An instance of a concept contains values of I_a, I_b, I_c, I_a^*, I_b^*, I_c^*, and w.

4.3.2. Tests and Discussion

IDB1 has been tested in the domain of power electronics control. One experiment has been performed using a training data set of 2000 instances and another 2000 instances for classification. We examined the descriptions generated and found that there are no features in the characteristic description of any concept of the seven cases; there are very few features (zero, one or two) with low completeness factor in the discriminant descriptions. All features in the confident descriptions have low consistency factors (less than 0.6). Therefore, this is a typical domain where data has little regularity. The accuracy is compared with the decision tree method, and the result is shown in Table 4.8.

From this experiment we can see that in domain with data of little regularity, IDB1 does not have a good classification capability compared with a decision tree method. There is another aspect where a decision tree is better. Concept descriptions in the form of a decision tree can be used for fast classification because the classification consists only of comparison of attributes along the path from the tree root to its leaf. Concept descriptions generated by IDB1 are in symbolic form. While they are convenient for humans to read, they usually cannot be used for fast real-time classification. This is because the classification is efficient only in case discriminant features are found with high completeness factors. Otherwise, the classification has to depend on the computation of similarity between concepts and instances which will take much longer time.

A second experiment is performed to see the robustness of the IDB1's algorithm to noise. In this experiment, we used a different set of attributes. In this problem, the control signal to be generated depends on the accumulated current errors and also the output control signal of the last period. There are three attributes: $I_{err}(k)$ is the difference of the measured current and the current reference in period k, $S_{err}(k)$ is the accumulated current error, and B(k) is the output binary control signal. The concepts correspond to the two possible output control signal values (0 or 1). This problem has a special property in its attributes. There is a relationship between $S_{err}(k)$ and B(k);

Table 4.8. Comparison of Classification Accuracies

	IDB1	Decision Tree
Rate of Correct Classification	78.6 %	92.7 %

however, the relationship does not exist in all instances. Its occurrence depends on the value of $S_{err}(k)$. Sometimes, the concept instances are described by the value of $S_{err}(k)$ and at other times concept instances are described by the combination of $S_{err}(k)$ and B(k).

At the first step of this experiment 4095 noise free instances are inserted in IDB1. The concept descriptions are shown in Figure 4.6. These concept descriptions are used to classify 300 unseen instances. In the second step, 1024 instances with 5 percent noise are used to generate concept descriptions. The resulting descriptions are shown in Figure 4.7, and are used to classify 300 unseen instances. This step is repeated for data sets with different noise levels, and the accuracies of classification for each level are shown in Table 4.9.

From the table, we can see that, measured in terms of classification accuracy, IDB1 is quite robust to noise. There is no sudden failure of classification capability in the presence of noise. The classification accuracy only decreases gradually with increasing noise. However, the concept descriptions are quite sensitive to the noise. For characteristic descriptions, the features have to cover all the positive instances; if one instance contains noise the affected features will be excluded from the description. Similarly, for discriminant descriptions, the features cannot cover any negative instances. Any noise would affect the descriptions.

In order to test the effect of domain knowledge on the results of induction, we performed another experiment. In this experiment we compare the results of two runs: one run without any domain knowledge, and a second run with domain knowledge that $S_{err}(k)$ is likely to have relationship with B(t). The concept descriptions are shown in Figure 4.8, and the classification accuracy is shown in Table 4.10.

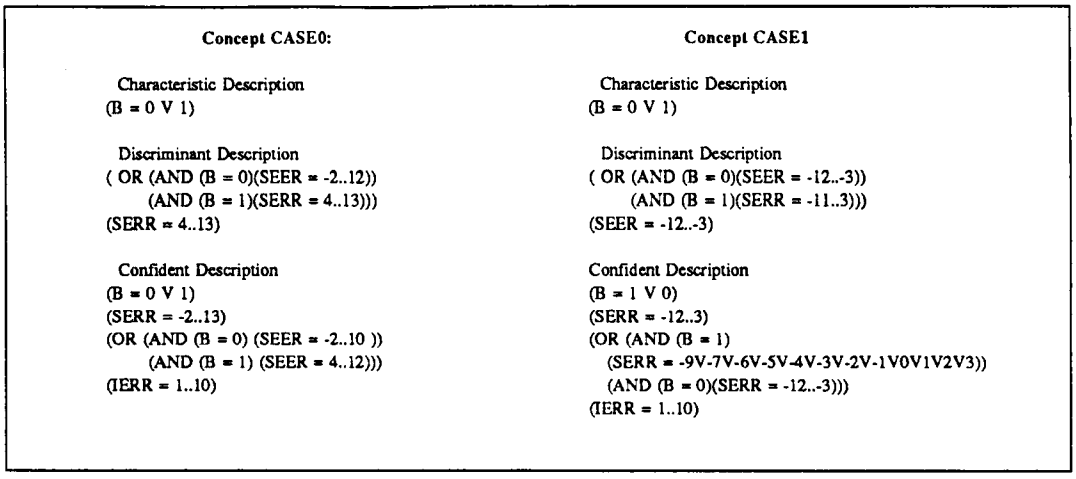

Figure 4.6. Concept Descriptions for Noise Free Data

From the comparison we can see that with domain knowledge the concept descriptions have a higher rate of correct classification. Without the domain knowledge the relationship between the attributes S_{err} and B cannot be found. This relationship provides an important discriminating information for the concepts. Missing this information, the classification accuracy should be affected. From this experiment we can see that in situations where domain knowledge is available IDB1 can utilize it to improve its performance. Because of efficiency consideration, IDB1 does not try to investigate all the possible relationships automatically.

From the above experiments we can see that IDB1 has a high robustness to noise in the respect of classification. However, the concept description are too sensitive to the noise. A small change in the input instances may cause large changes in the concept descriptions. One of the reasons is the definitions of the description types (characteristic and discriminant). Another reason is that the algorithm is incremental. It is impossible for the inductive algorithm to anticipate good descriptions in the future, and keeping all the description templates in the system is prohibitively inefficient in both processing space and time.

Further research work needs to be performed for description robustness and to develop an efficient induction method that can find all possible relationships automatically.

INPUT DATA WITH 5 % NOISE

Concept CASE0:

Characteristic Description
(B = 0 V 1)

Discriminant Description
(OR (AND (B = 0)
(SEER = 2 V 7 V 8 V 9 V 10 V 11 V 12))
(AND (B = 1)(SERR = 5 V 8 V 9)))
(SERR = 8..9)

Confident Description
(B = 0 V 1)
(SERR = -2 V 4 V 5 V 6 V 7 V 8)
(OR (AND (B = 0)
(SEER = -2 V -1 V 0 V 1 V 3))
(AND (B = 1) (SEER = 4..6))
(IERR = 5 V 6 V 7 V 9)

Concept CASE1

Characteristic Description
(B = 0 V 1)

Discriminant Description

(OR (AND (B = 0)(SEER = -11 V -10))
(AND (B = 1)(SERR = -10 V -9 V -8 V -7 V -3)))
(SEER = -10 V -11)

Confident Description
(B = 0 V 1)
(SERR = -7..-3)
(OR (AND (B = 0)(SERR = -4 V -3 V -6))
(AND (B = 1)(SERR = -2..3)
(IERR = 2 V 3 V 5 V 7)

INPUT DATA WITH 10 % NOISE

Concept CASE0

Characteristic Description
(B = 0 V 1)

Discriminant Description
(SERR = 9 V 13)
(OR (AND (B = 0)(SERR = 12 V 9 V 7 V 6))
(AND (B = 1)(SERR = 13 V 9 V 8)))

Confident Description
(B = 1 V 0)
(SERR = -1 V -2 V 5 V 6 V 4)
(IERR = 1 V 6 V 9 V 10 V 8)
(OR (AND (B = 0)(SERR = -2..2))
(AND (B = 1)(SERR = 4..6)))

Concept CASE1

Characteristic Description
(B = 0 V 1)

Discriminant Description
(SERR = -12)
(OR (AND (B = 0)(SERR = -8 V -12))
(AND (B = 1)(SERR = -11 V -9 V -12)

Confident Description
(B = 1 V 0)
(SERR = -6 V -8 V -7 V -4 V -5 V -3 V 3)
(IERR = 1 V 2 V 3 V 5 V 7)
(OR (AND (B = 1)(SERR = -2 V -1 V 2 V 1 V 3))
(AND (B = 0)(SERR = -3)))

Figure 4.7. Concept Descriptions for Noisy Data

Table 4.9. Classification Accuracies in Case of Noisy Data

NOISE LEVEL (%)	CLASSIFICATION ACCURACY (%)
0	100
5	100
10	95.7
15	90.3
20	88.9
25	81.9
30	74.3
35	63.6

4.4. Evaluation of the IDB1 System

In this section we summarize the experiments performed in previous sections and give an evaluation of the IDB1 system. We evaluate the IDB1 system in three aspects: the representation capability, the quality of conceptual induction, and the computational efficiency. The evaluation is performed with the following methods (1) comparison with other systems, (2) performing experiments on the system, and (3) analyzing the algorithms.

4.4.1. Representation Capability of IDB1

The description language employed for expressing input descriptions of concept instances and the output descriptions of concepts partly determines the possible generalization operations that an induction system can perform, and to a large extent determines the quality and utility of the output concept descriptions [Michalski 1983b].

The criterion of representation capability is based on the following aspects of the description language: (1) representation of structural concepts, (2) representation of disjunction or internal disjunction, (3) representation of relationships between concept components, and (4) types of attributes.

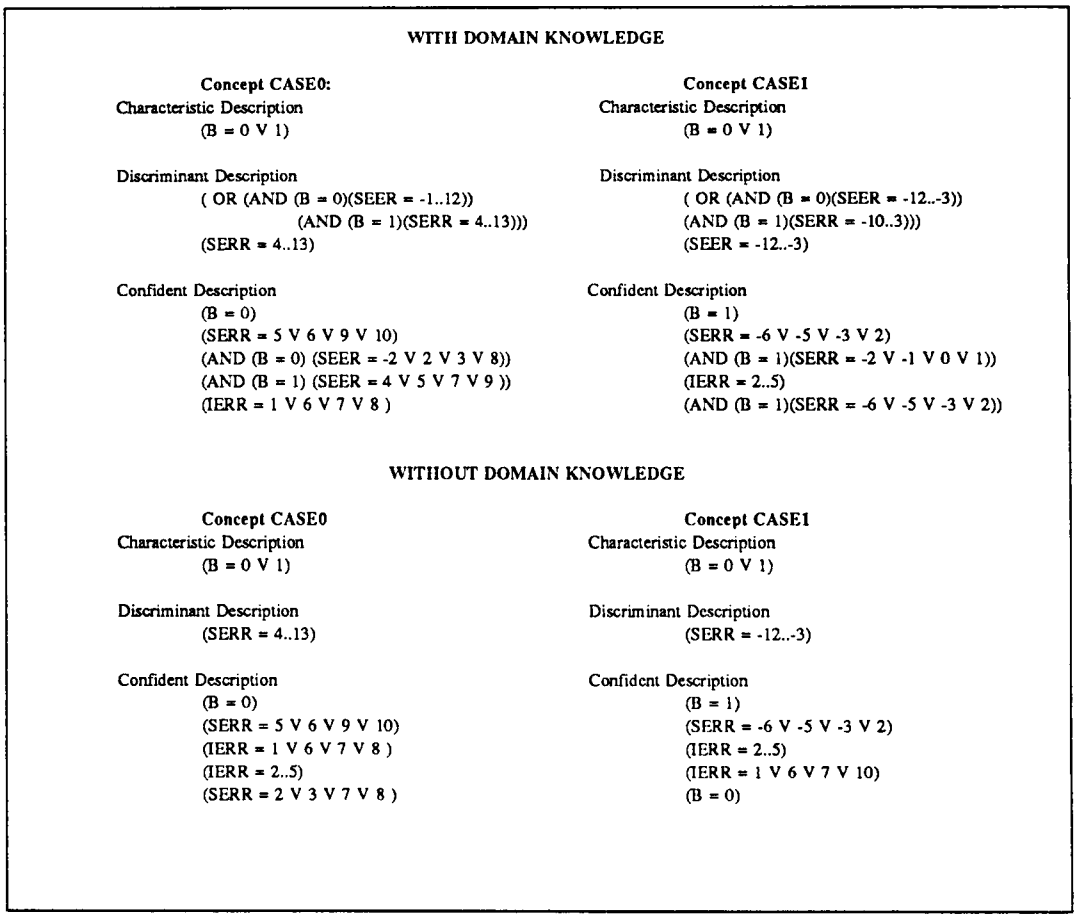

Figure 4.8. Comparison of Concept Descriptions with and without Domain Knowledge

Table 4.10. Comparison of Classification Accuracies

WITH DOMAIN KNOWLEDGE	WITHOUT DOMAIN KNOWLEDGE
100 %	87.8 %

IDB1 is an incremental inductive system with an expressive representation language which is described in Section 3.1.2. Combining a rich representation language with incremental induction is difficult. Most incremental inductive systems use a simple representation language, such as attribute-value pairs.

In many application domains, a physical object is composed of several parts called components, and a description of the object must consist of the features of its components and relationships among them. For those domains, an induction system needs the capability to represent structural concepts. In IDB1, the structural concepts of SSME domain can be represented, where the descriptions are composed of many features describing sensor behavior. Sensors are components of a basic concept. Some complex feature values can be represented, such as in the expression:

(AND (end-time-diff (s23-s55) = -10 ∨ 1)

(start-time-diff (s23-255) = 1..3))

where "-10 ∨ 1" is an internal disjunction, and "1..3" is a range-from value. In the same example, we can see the relationship between two attributes "end-time-diff" and "start-time-diff" is represented by a conjunction.

In many situations, a conjunction form of description is not sufficient to describe a concept because one conjunction only describes part of the instances. In order to describe the whole set of instances of a concept, a disjunction of several conjunctions is needed. Internal disjunction is necessary to make the description more simplified.

The ability to represent relationships of components is important because in some situations the important characteristics of a concept lie in the relationship of its components. For example, in the space shuttle engine test analysis, some important characteristics of engine faults are shown in the relationships of sensors. In different domains, different types of attributes are needed for describing concepts. In order to apply an induction system to more types of application problems, the induction system should be able to represent and process more types of attributes.

In Figure 4.6 we can see the disjunctive descriptions, for example:

(OR (AND (B = 0)(SERR = -2..13))

(AND (B = 1)(SERR = 4..13)))

The results of conceptual induction are the characteristic descriptions, discriminant descriptions and confident descriptions.

The representation language allows matching variables and functions to appear in an expression. This is a very important feature because it makes it possible for the deductive mechanism to perform deduction directly on the concept descriptions. It also facilitates the representation of general deductive rules. For example, if there is a relationship between two attributes of all sensors, then only one rule is needed to represent the relationships with a matching variable in place of a specific sensor. The following rule deduces the relationship "same-trend" for all sensor pairs:

IF ((direction (?sensor1) = ?v) ∧

(direction (?sensor2) = ?v))

THEN ADD (same-trend ?sensor1 ?sensor2)

Representation capability of a conceptual inductive system is an important criterion which determines the scope of types of problems and domain areas the system can be applied to. As discussed in several examples in Chapter 2, some problems cannot be solved in a set-based model, which has less capability of representation. Systems with expressive representation languages usually have a high complexity of induction process. In a database environment an incremental processing is necessary for an inductive system to have high computation efficiency.

In summary, the representation language in IDB1 has the following features:

(1) It can represent structural concepts;

(2) It allows many types of attributes including nominal, linear and hierarchical types;

(3) It handles complex feature values, such as internal disjunctions and range form;

(4) It can represent relationships between attributes and between components;

(5) It allows logical expressions with connectives of conjunctions and disjunctions; and

(6) It has a close connection with and supports the deductive mechanism.

Now let us compare the representation capability of IDB1 with other incremental systems of conceptual induction. Lebowitz's UNIMEM [1986] represents descriptions of instances and concepts with a set of attribute-value pairs. For concepts, the system associates each value with an integer called confidence which is computed independently. UNIMEM's representation is less general since it cannot easily handle structural concepts. It has limited generalization rules since the value forms are limited; for example, it cannot handle range-form values and hierarchical attributes. It cannot represent relationships between attributes and components and support constructive induction. The internal disjunctions are represented by repeating features of different values of an associated attribute. The following is an example of a concept description D in UNIMEM[1]:

D: (pressure-of-sensor1 pre3:3
pressure-of-sensor2 pre1:3
changing-rate-of-sensor1 rat4:5
changing-rate-of-sensor2 rat2:5)

Fisher's COBWEB [1987] also uses attribute-value pairs to represent descriptions of instances and concepts, and has limitations on types of attributes because only nominal attributes are allowed (numeric attributes cannot be handled). In COBWEB a concept description includes

[1] An attribute derived from numeric data has a value indicating its category and number of categories. For example, the value pre3:3 indicates the pressure falls in the third of three categories.

every attribute observed; and associated with each attribute are all possible values for that attribute. Each such value has two associated probabilities[2]. An example of concept description is shown below:

D: (pressure-of-sensor1 high 1, 1, medium 0, 0, low 0, 0

pressure-of-sensor2 high 1, 1, medium 0, 0, low 0, 0

changing-rate-of-sensor1 high .5, 1, medium .5, 1, low 0, 0

changing-rate-of-sensor2 high 0, 0, medium .5, 1, low .5, 1)

COBWEB cannot handle structural concepts and relationships between attributes. For example, let us classify apples with two attributes: size and color. Size has two possible values: "big" and "small" and color has two possible values: "red" and "green". If a half of the input instances are "big red apples", and the other half are "small green apples". The probabilities of features [color red], [size big], [color green], and [size small] in the concept description are adjusted individually, which will result in a concept description describing "big green apples" with the same probability as "big red apples", even though there is not a single "big green apple" that was input.

This comparative analysis with the two incremental systems indicates that IDB is more expressive in its representation language.

4.4.2. Quality of Conceptual Induction in IDB1

The quality of the induction is determined by the quality of the resulting concept descriptions and the quality of the concept hierarchy.

The criteria we used for evaluating the quality of concept descriptions are the classification capability, the relative matching factor, number of confident features in each concept description, and the average feature confidence value for every concept.

---

[2] One is called predictiveness of a value v for category c and is defined as the conditional probability that an instance i will be a member of c, given that i has value v. The other is called predictability of a value v for category c and is defined as the conditional probability that an instance i will have value v, given that i is a member of c.

To examine the quality of the descriptions of a group of basic concepts in a specific domain, we employ the relative matching factors for each pair of different basic concepts and the average relative matching factor among pairs of concepts in the group.

The average relative matching factor RMF_a is the average of the relative matching factors of all the pairs of the different basic concepts:

$$RMF_a = \frac{1}{M} \sum_{j \neq i}^{u} RMF(D_i, D_j)$$

where M= (u-1)u/2, and u is the total number of basic concepts.

The larger matching factor of a basic concept with itself indicates that more features of high confidence are found for the concept, and a smaller matching factor of a basic concept with another basic concept indicates that the concept descriptions have a higher discriminating quality. The discriminating quality of descriptions of a set of concepts is defined as negation of the average matching factors of different concepts:

$$-\frac{1}{M} \sum_{j \neq i}^{u} MF(D_i, D_j).$$

Therefore, smaller relative matching factors of basic concepts indicate a higher quality of descriptions of basic concepts.

A concept description with more confident features is considered to be of higher quality. A higher average confidence shows a higher quality of concept description.

The quality of the concept hierarchy is measured by the error rate which is the ratio of the number of wrongly classified instances to the total number of instances incorporated into the concept hierarchy. A lower error rate indicates a higher quality of the concept hierarchy.

We have performed experiments on the quality of conceptual induction in IDB1. In subsection 4.1.2, one experiment was performed to show that with using domain knowledge IDB1 improves the quality of concept descriptions and the concept hierarchy. One experiment on

chemical process domain (in subsection 4.2.2) has shown good classification capability of concept descriptions generated by IDB1. Comparing with the neural network and decision tree methods [Wang et al. 1992], IDB1 can perform as well in the CSTR domain. Another experiment has shown that IDB1's incremental induction can improve the quality of concept descriptions by gradually incorporating more input instances.

Comparison in classification accuracy with a decision tree method was performed in the application domain of power electronic equipment control (subsection 4.3.2). The result shows that IDB1 does not perform as well as the decision tree method in the domain with little regularity. Another experiment in subsection 4.3.2 has been performed to test IDB1's robustness to noise. The result shows that IDB1 has a high robustness to noise.

From the experiments discussed in the previous sections, we know that both the quality of concept descriptions and the quality of conceptual clustering can be improved significantly by using domain related knowledge. However, the previous inductive systems UNIMEM [Lebowitz 1987] and COBWEB [Fisher 1987] do not employ any domain knowledge to guide the inductive process. In each system, the inductive process is based only on the input data. Comparing with these two systems, IDB1 has a strong advantage in utilizing various types of knowledge to improve the quality of conceptual induction.

4.4.3. Computational Efficiency of IDB1

Computational efficiency is an important criterion to evaluate an algorithm of inductive processing, especially in a database environment. Since the execution time of the inductive inference is affected by data patterns, inductive biases, and many other factors, it is very difficult to make a thorough theoretical analysis of the time efficiency of IDB1. To examine the performance of IDB1, a general analysis as well as experiments are described below.

4.4.3.1. Cost Analysis of the Basic Abstraction

In this subsection, we analyze the processing time of the basic abstraction for incorporating an instance into the inductive system with respect to n, the number of instances already incorporated into the system. The time for transforming, deducing, and processing

characteristic descriptions is not related to the number of instances already incorporated. The time for processing discriminant descriptions changes with the size of the uniqueness-table, which is the number of different features of all incorporated instances. The uniqueness-table is a tree structure in which a feature expression is used as its index. In the KNOWD model, each expression in the description is processed independently of other expressions. Suppose the average number of expressions in a description of an instance is a constant n_e. Processing each expression of the instance description could be performed in one of three cases: (1) in the first case, the expression has no matching expression in the uniqueness-table, and the processing time includes T_f, time for fetching matching expressions from the table, and T_a, time for adding an entry into the table; (2) in the second case, the expression has a conflict or compatible matching expression in the uniqueness-table, and the processing time includes T_f and T_c, time for specialization, generalization and table-entry modification; and (3) in the third case, the expression has matching expressions which are neither conflicting nor compatible, and the processing time includes T_f and T_a. Let r be the average number of instances in which an expression appears, B be the average branching factor of the uniqueness-table, and m be the average number of matching expressions for each input expression. Since an expression is stored only once in the uniqueness-table, the total number of entries in the table is $n \cdot n_e / r$. In general, the average depth of an entry in the table is approximately $\log_B(n \cdot n_e / r)$, so

$$T_f = O(m \cdot \log_B(\frac{n \cdot n_e}{r})) \tag{4.1}$$

$$T_a = O(\log_B(\frac{n \cdot n_e}{r})) \tag{4.2}$$

T_c is a constant with respect to n. In the instance description, the number of expressions in case one is n_e/r because only the first time an expression is input is it in case one. The number of

expressions in case two and three is $n_e(r-1)/r$. Then the cost for processing expressions in case one is:

$$C_1=\frac{n_e}{r}(T_f+T_a)=O(\frac{n_e}{r}(m+1)\log_B(n\cdot\frac{n_e}{r}))$$

The ratio of the number of expressions in case two to the number of expressions in case three is roughly 1:m, because only one expression is possible conflicting or compatible of the m matching expressions. Ignoring the cost T_c, the cost for processing expressions in case two is:

$$C_2=n_e\cdot\frac{r-1}{r}\cdot\frac{1}{m}T_f=O(n_e\cdot\frac{r-1}{r}\log_B(n\cdot\frac{n_e}{r})) \tag{4.4}$$

The cost for processing expressions in case three is:

$$\begin{aligned} C_3&=n_e\cdot\frac{r-1}{r}\cdot\frac{m-1}{m}\cdot(T_f+T_a) \\ &=O(n_e\cdot\frac{r-1}{r}\cdot\frac{m-1}{m}\cdot(m+1)\log_B(n\cdot\frac{n_e}{r})) \end{aligned} \tag{4.5}$$

Therefore, the cost for incorporating an instance is

$$C=C_1+C_2+C_3=O(n_e\cdot(m+1+\frac{1}{mr}-\frac{1}{m})\log_B(n\cdot\frac{n_e}{r})) \tag{4.6}$$

From the equation (4.6) we can see that the cost of basic abstraction is of a logarithmic order with respect to the number of instances incorporated. Therefore, the processing of the basic abstraction is efficient.

4.4.3.2. Cost Analysis of the Advanced Abstraction

When computing the cost of incorporating an instance into the concept hierarchy, we need to consider two cases: (1) the instance is the first instance of a basic concept, and (2) the instance

is one of the basic concepts already existing in the hierarchy. Let n be the number of instances already incorporated in the hierarchy, B_1 the average branching factor of a cluster node in the hierarchy, and B_2 the average branching factor of a basic concept node in the hierarchy, then the number of basic concepts in the hierarchy will be: $N = n / B_2$. Let N_e be the average number of expressions for each node, and L_e be the average number of atoms in each expression, then the cost of computing the matching factor of two nodes will be $O(N_e^2 L_e)$.

For the first case a top-down extension of the hierarchy is performed. The new basic concept instance can be incorporated with a cluster in three possible ways: i) adding the instance as a leaf of the cluster; ii) combining the instance and an existing leaf to form a new cluster; and iii) recursively trying to perform the process with one of the subclusters. At each cluster node, the computation to generalize the cluster description has a cost $O(N_e^2 L_e)$; the computation to find all the close nodes has a cost $O(N_e^2 L_e B_1)$; the computation to find the best node has a cost $O(N_e^2 L_e B_1 B_3)$ where B_3 is the average number of close nodes under a cluster node ($B_3 \leq B_1$); and the operation to combine two basic concepts to form a new cluster has a cost $O(N_e^2 L_e)$. All the computations are performed at each node all the way from the top to the right cluster node and the last operation can be performed at most once. Let us assume that the average depth of a cluster node is $\log_{B1} N = \log_{B1}(n / B_2)$, then the total cost for incorporating an instance in case one is:

$$C_{t_1} = O(N_e^2 L_e (1 + B_1 + B_1 B_3) \log_{B_1}(\frac{n}{B_2})) \tag{4.7}$$

The cost of the last operation is ignored here and further simplification makes

$$C_{t_1} = O(N_e^2 L_e B_1 B_3 \log_{B_1}(\frac{n}{B_2})) \tag{4.8}$$

For the second case a bottom up hierarchy modification is performed. The process is only performed on all the ancestors of an existing basic concept node. For each expression in the basic concept node, the possible operations are: 1) finding a matching expression in a cluster node which has a cost $O(N_eL_e)$, 2) specializing an expression which has a cost $O(L_e)$, 3) generalizing an expression which has a cost $O(L_e)$. After modifying the description of a cluster, the operation for trying to create a new cluster which has a cost $O(N_e^2L_e\ B_1 + N_e^2L_e)$ is performed. Then the total cost for incorporating an instance in case two is:

$$\begin{aligned} C_{t_2} &= O((N_eL_e+2L_e)N_e\log_{B_1}(\frac{n}{B_2})+N_e^2L_eB_1+N_e^2L_e) \\ &= O(N_e^2L_e\log_{B_1}(\frac{n}{B_2})+N_e^2L_eB_1) \end{aligned} \tag{4.9}$$

The n instances are incorporated in either case one or case two. Since the ratio of case one to case two can be taken as B_2, we can combine the two costs C_{t1} and C_{t2} to give an expected cost:

$$\begin{aligned} C_t &= \frac{C_{t_1}}{B_2}+\frac{(B_2-1)C_{t_2}}{B_2} \\ &= O(N_e^2L_e(\frac{B_1B_3}{B_2}+1)\log_{B_1}(\frac{n}{B_2})) \end{aligned} \tag{4.10}$$

In the KNOWD model not all attributes are required to appear in the description, so N_eL_e is less than the number of all possible attributes. Also in general, $B_2 > B_1$, the cost is roughly the same as in the COBWEB system [13] which is an incremental conceptual clustering system based on the set-based model. From the above equation, we can see that the expected cost C_t is on the logarithmic order with respect to n, the number of instances incorporated. Therefore, the KNOWD method is an efficient inductive method.

4.4.3.3. Experimental Cost Analysis of Computational Efficiency

EXPERIMENT A:

First, let us examine how the execution time of the inductive process is affected by the number of features in each concept instance. Let T be the execution time for incorporating an instance, and N be the number of features in the input instance.

The execution time T includes:

(1) the time for simple transformation, which is O(N);

(2) the time for deduction, which is determined by the size of the fact base (i.e., N), the number of enabled deductive rules, and the execution time of activated deductive rules;

(3) the time for generalization, which includes T_1, the time for processing characteristic description, T_2, the time for processing discriminant description, and T_3, the time for processing advanced abstraction, where:

T_1 is determined by the number of matching features of the concept and its instance and the execution time of generalization rules. T_1 is O(N), assuming the number of features in a concept description does not depend on N;

T_2 is determined by N and the number of different features of input instances so far;

T_3 is affected by the concept hierarchy and N.

The purpose of this experiment is to examine the relationship between T and N. In this experiment we use a different number of sensors in the instances, so that the number of features is changed. SSME simulation data of sixty-one instances in five types of valve-blockage faults are incorporated into the concept hierarchy. The time for incorporating one instance is computed as the average time of all the instances.

The diagram in Figure 4.9 shows the result of the experiment. From the diagram we can see that the execution time for incorporating an instance has roughly a linear relationship with the number of features in the input instances.

EXPERIMENT B:

Now, let us examine how the execution time of the inductive process is affected by the number of instances already incorporated in the concept hierarchy. Let T be the execution time for incorporating an instance, and M be the number of incorporated instances.

From the previous analysis, the time for basic abstraction is O(log(M)), and the time for

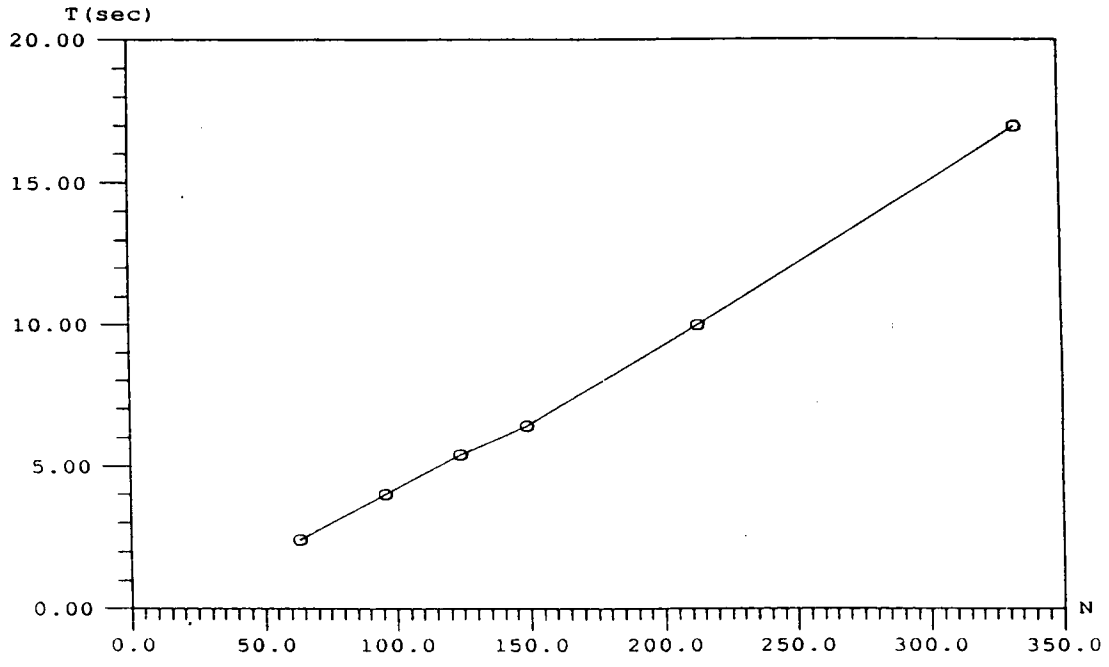

Figure 4.9. Result of Experiment A

advanced abstraction is also O(log(M)), assuming other parameters are constants. So the time T is O(log(M)).

The purpose of this experiment is to examine the relationship between T and M. In this experiment we use SSME simulation data of sixty-one instances in five types of valve-blockage faults. The T value at each point is computed as an average of the T values on five neighboring points. This average is used to smooth the fluctuation of T, since for different cases (whether a

new concept node is created or modified, whether a new cluster node is created or not), T can have an uneven change.

The diagram in Figure 4.10 shows the result of the experiment. From the diagram we can see that the execution time for incorporating an instance has a low rate of increase with respect to the number of incorporated instances, even though the form of the curve does not exactly coincide with a logarithmic curve.

4.4.3.4. Comparison of Computational Efficiency with Other Systems

Lebowitz performed an experiment on the time efficiency of UNIMEM [1987]. The growth of time to incorporate new instances appears to be a logarithmic increase with the number of instances already incorporated. The experiment result indicates that after incorporating certain number of instances, the processing time becomes nearly a constant.

Fisher's COBWEB is computationally economical. His analysis of computation cost [1987] indicates that the time for incorporating a new instance is:

$$C_t = O(A \cdot V \cdot B_1^2 \cdot \log_{B_1} n)$$

where A is the total number of attributes, V is the average number of values per attribute, B_1 is the average branch factor, and n is the number of instances already incorporated into the system.

From the analysis and experiments we performed on IDB1, we can see that our system has the same computation cost as the above two incremental conceptual inductive systems. Considering that our system deals with more complicated representation, we can claim that IDB1 is an efficient system.

CLUSTER/2 [Michalski 1983a] is a nonincremental conceptual inductive system which requires rebuilding a generalization hierarchy when incorporating a new instance. As indicated by Fisher that polynomial or exponential cost is associated with any nonincremental inductive method. So comparing with CLUSTER/2, IDB1 is significantly less expensive.

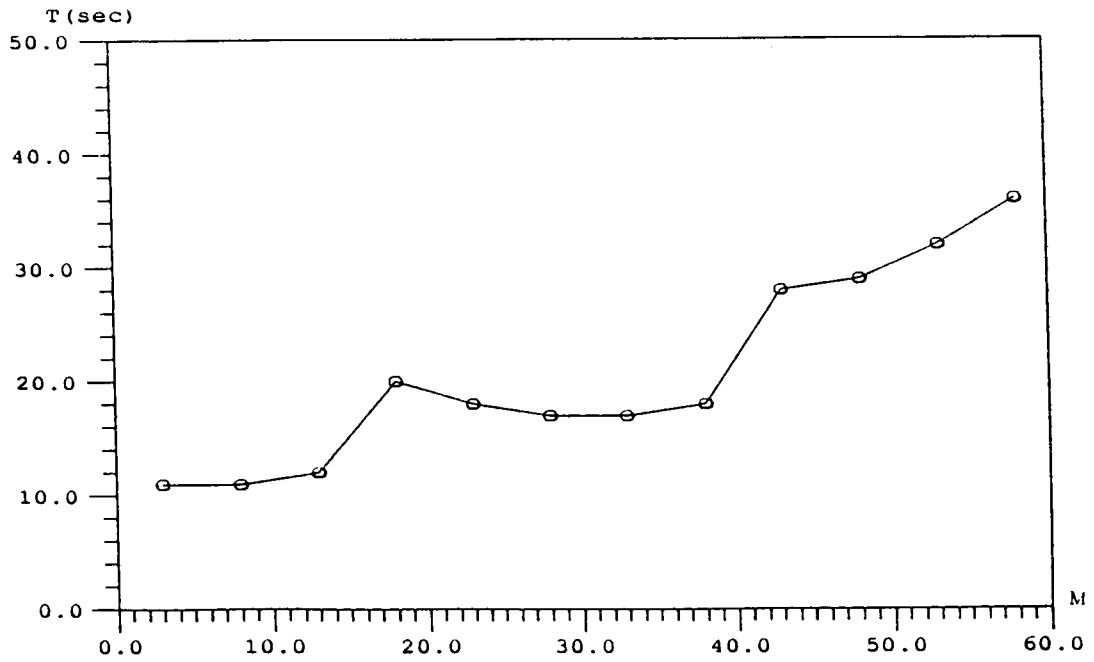

Figure 4.10. Result of Experiment B

4.5. Summary

In this chapter, we have applied IDB1, the inductive database system, to three different domains. We performed various experiments to test the performance of IDB1. The experiments and their purposes are summarized in Table 4.11.

We also evaluated IDB1 in three aspects: the representation capability, the quality of conceptual induction, and the computational efficiency. Next chapter will summarize the research work and good features of our model and methodology.

Table 4.11. Summary of Experiments on IDB1

No.	SECTION	PURPOSE OF EXPERIMENT
1	4.1.2	To show that using domain knowledge can improve the quality of concept descriptions generated by conceptual induction.
2	4.1.2	To show that using domain knowledge can improve the quality of the concept hierarchy.
3	4.2.2	To test the classification capability of the learned concept descriptions by IDB1.
4	4.2.2	To test incremental improvement of concept descriptions through incorporating new instances.
5	4.3.2	To test the classification accuracy of the learned concept descriptions in a domain with little regularity.
6	4.3.2	To test the robustness of IDB1 to noise.
7	4.3.2	To test the effect of domain knowledge on the classification accuracy.
8	4.4.3.3	To test the execution time of IDB1 affected by the number of features presented in each instance.
9	4.4.3.3	To test the execution time of IDB1 affected by the number of instances already incorporated in the database.

CHAPTER 5

CONCLUSION

5.1. Summary of the Research Work

In a database environment a conceptual induction system should satisfy several requirements: incremental control strategy, expressive representation, efficient processing, knowledge utilization and general methodology. In order to design and implement a conceptual induction system for an inductive database that possesses all the required properties, we have performed following research work in this dissertation:

(1) Formalization of conceptual induction systems and database systems;

(2) Analysis of different conceptual induction models;

(3) Development of the Knowledge-Directed conceptual induction model (KNOWD model);

(4) Development of the Knowledge-Directed conceptual induction methodology (KNOWD methodology);

(5) Development of the Generalization Data Model (GDM) and implementation of the inductive database IDB1; and

(6) Application of the KNOWD methodology in three areas.

Two principal contributions have been made in the research of this dissertation:

(1) The Knowledge-Directed conceptual inductive model (KNOWD model); and

(2) The Knowledge-Directed conceptual inductive methodology (KNOWD methodology).

5.2. Importance of the Research Work

For making decisions and solving problems in daily life or business we need knowledge. Because of their growing number and size, databases are becoming important sources of knowledge. Many new applications demand a database management system (DBMS) to perform knowledge-based inference on various types of data. New knowledge can be obtained by inferentially analyzing this data in order to discover the unexpected relationships, unknown characteristics, hidden patterns, and regularities. The knowledge derived from data in databases can also be used for building expert systems.

Conventional DBMSs lack inferential capabilities. They can only answer precise queries about rigidly formatted data stored in the database. Induction is one of the main inferential capabilities required by an intelligent DBMS. The purpose of this dissertation research is to develop a model and methodology of conceptual induction for use in DBMSs so that knowledge can be automatically generated from data.

An inductive database system has an inductive inference capability that conventional database systems do not have. It performs induction on data so that new features of a concept, and new concepts can be discovered. Users can query about general features of a concept which include characteristic and discriminant descriptions, and query about common features of a set of concepts. Knowledge acquisition has become the bottleneck of building expert systems. An inductive database can facilitate knowledge acquisition in many ways. An inductive database system can be applied to many application areas. For example, it can be applied to system failure data by a manufacturer for discovering unknown reasons for product failures, to crime data by a government agency, to financial data by a financial analyst, to patient data by a medical researcher, and so on.

5.3. Features of the KNOWD model and Methodology

The important features of the KNOWD model and methodology as well as their implementation and application aspects are summarized below.

The KNOWD model has the following main features:

A. *Relationships of features are derived from domain knowledge.* In previous incremental conceptual inductive systems, attributes of a concept are regarded as independent, and the confidence or the estimate of the probability of a feature is adjusted individually. But in a real world domain, regarding attributes as independent may result in incorrect descriptions of concepts. Attributes often have relationships which cannot be represented by previous incremental conceptual inductive systems. The KNOWD model describes a concept by a set of logical expressions, so that an attribute relationship can be represented by a compound feature in which attributes are related by logical connectives. Possible attribute relationships are also derived by using deductive rules in the KNOWD model.

B. *Expressive representation is integrated with incremental induction.* Previous incremental conceptual inductive systems only employ simple formalisms such as attribute-value pairs with occurrence count or confidence, making it impossible to represent logical connectives and internal structures. Significant improvements have been made in the KNOWD model in which many value types, such as nominal, linear and hierarchical types, can be represented, so that more types of generalization than only selective generalization can be performed, and inexact feature matching can be employed. Conjunctions and disjunctions are allowed to represent relationships of features. Structured concepts, which have component features and component relationships, can also be represented.

C. *Use of inductive biases facilitates the inductive inference.* In some application domains, people know that certain attributes are more important than other attributes, that some components are more relevant to a concept than other components, and that some components are irrelevant. This kind of knowledge is very useful and is employed in the KNOWD model in guiding the inductive process, but in previous incremental conceptual inductive systems, it was not utilized.

The KNOWD methodology has the following unique features:

A. *It is based on the KNOWD inductive model.* A rich representation combined with incremental induction is the dominant characteristic of this methodology.

B. *It employs a deductive mechanism for utilizing domain knowledge.* Deductive rules are used for concept constraints which check the consistency and errors of instance data. Constructive rules are used to derive a new vocabulary for describing concepts. Deductive rules are also used to derive promising relationships.

C. *It consists of multilevel abstraction.* Multilevel abstraction is a unique control strategy of induction which integrates the concept description process and the concept formation process into a single system and allows the inductive database to be applied to a larger range of domains.

D. *It explicitly contains a comprehensive inductive bias management system (BMS).* Various inductive biases are managed by BMS, so that it is easier to build up and maintain necessary biases for a better performance of the inductive inference.

We have developed and implemented an inductive database system IDB1, which is based on the KNOWD inductive model and methodology. An object-oriented database system is employed as the underlying database system. Objects are the basic constructs to store and organize data and knowledge. The Generalization Data Model (GDM) has been developed to support inductive semantics. BMS (an inductive bias management system) is implemented as an important part of IDB1. IDB1 has been applied to several areas including fault analysis of the space shuttle main engine, chemical process monitoring and power electronic equipment control. Experiments were performed to show that the quality of conceptual induction can be improved by using domain knowledge. IDB1 is quite robust to noise in its capability of classification. Analysis and experiments for examining the cost of the inductive process of IDB1 were also performed, and these show that IDB1 is time efficient.

5.3. Further Research and Development

Further research work needs to be done to improve the performance of the inductive process in a database system. IDB1's classification capability in domains where data has less regularity needs improvement. In any logic-base induction system if the main goal is to generate concise concept descriptions, then generalization must be performed. This generalization usually

results in inaccurate concept descriptions for classification in a less regular domain. For set-based models a similar difficulty exists. If there is no confident feature for the concept, then the classification will be inaccurate. To combine a high classification accuracy and concise concept descriptions in a conceptual induction system for a less regular domain is a challenging task.

The deductive supporting mechanism appears to be time-consuming for processing a large number of instances. The duplication of the deductive process on instances of the same concept needs to be eliminated because most features in these instances are the same and only a small number of features in the descriptions are different. Selection of system parameters needs to be monitored and directed by the system so that a developer can easily build up an application of the inductive system.

Known knowledge about concepts, such as known features or known relationships, from human experts needs to be added to the concept hierarchy directly. Redundant induction for this kind of knowledge from data needs to be avoided. Knowledge with uncertainty should also be incorporated directly from human experts. This kind of knowledge needs to be verified by data. Partial knowledge about the possible classes of basic concepts should be used as a guideline for the advanced abstraction. For example, in SSME four subsystems can be identified: MCC (Main Combustion Chamber), FUEL, OXIDIZER, and CONTROL. Faults occurring in each subsystem may have a similar set of critical sensors. Within each subsystem, faults can be classified into several classes, such as duct faults, valve faults and pump faults. Therefore, instead of starting with an empty hierarchy, the user may initialize a concept hierarchy with partial knowledge about a possible framework. Some concept constraints may also be added to the clusters in the initial hierarchy. The initial clusters may be confirmed or deleted and new clusters may be created by the inductive system through the incorporation of concept instances. With the above extension, the inductive database will be able to acquire knowledge from both human experts and data.

Further research work is also need for combining efficient incremental processing with automatic identification of relationships between attributes. Trying to investigate all the

combinations is not feasible for domains with many attributes. Fully depending on the guidance of domain knowledge is not possible for domains with no knowledge.

Using a natural language in the user interface of the inductive database would be of great importance. A natural language interface allows the user to query the database in English (or other human languages), and presents the concept descriptions in English, so that the inductive database shows more intelligence and is more friendly to users. We leave this for future research.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Adiba, M., and Nguyen, G.T., "Knowledge Engineering for CAD/VLSI on a Generalized Data Management System", in *Knowledge Engineering in Computer-aided Design*, J.S. Gero (ed.), pp. 331-346, North-Holland, 1985.

[2] Anderberg, M.R., *Cluster Analysis for Applications*, Academic Press, New York, 1973.

[3] Andrews, T. and C. Harris, "Combining Language and Database Advances in an Object-oriented Development Environment," in *Proceedings of OOPSLA*, 1987.

[4] Bancilhon, F. and P. Buneman, *Advances in Database Programming Languages*, ACM Press, 1990.

[5] Bergadano, F. and A. Giorana, "A Knowledge intensive approach to concept induction", *Proceedings of the 5th Int'l Conf. on Machine Learning*, pp. 305-317, Morgan Kaufmann, Los Altos, Calif., Jun. 1988.

[6] Bert, M.N., M.L. Denarie, A.D. Leva and P. Giolito, "Rule Management for Heterogeneous Knowledge-based Systems", in *The Proc. of the 1st Int'l Conf. on Industrial & Engineering Applications of A.I. and ES*, pp. 823-832, 1988.

[7] Biermann, A.W., and R. Krishnaswamy, "Constructing Programs from Example Computations", *IEEE Trans, Software Engineering*, SE-2, pp. 141-153, 1976.

[8] Birdwell, J.D., J.S. Lawler, G. Qiu, R.E. Uhrig, Z. Duan, R. Horn, J. Ilic, S. Liang, S. Patek, P. Shanmugam, and Y. Zhu, "New Approaches to Real-Time Expert System Design for Power Electronics Applications", University of Tennessee, Dept. of Electronics and Computer Engineering, Final Report, contract no. TCRD05-CR89-D070 with Tennessee Center for Research and Development, EPRI Power Electronics Applications Center, January 1992.

[9] Booch, G., "Object-Oriented Development", *IEEE Transactions on Software Engineering*, Vol.SE-12, No.2, pp. 211-221, Feb.,1986.

[10] Brodie, M. L., and M. Jarke, "On Integrating Logic Programming and Database", in *Expert Database Systems*, L. Kerschberg (ed.), Proceedings, First Intl. Workshop on Expert Database Systems, pp. 191-207, Kiawah Island, S.C., Oct. 1986.

[11] Brodie, M.L., "Future Intelligent Information Systems: AI and Database Technologies Working Together", in *Artificial Intelligence & Databases*, J. Mylopoulos & M.L. Brodie (eds.), pp. 623-641, Morgan Kaufmann Publishers, Palo Alto, CA, 1989.

[12] Brodie, M.L., J. Mylopoulos and J.W. Schmidt (eds.), *On Conceptual Modelling:*

Perspectives from Artificial Intelligence, Databases, and Programming Languages, Springer-Verlag, New York, 1984a.

[13] Brodie, M.L. and J. Mylopoulos (eds), *On Knowledge Management System*, Springer-Verlag, New York, N.Y., 1986.

[14] Blum, R.L., *Discovery and Representation of Causal Relationships from a Large Time Oriented Clinical Database: The RX Project*, Springer-Verlag, 1982.

[15] Callan, J.P., "Knowledge-based Feature Generation", in *Proceedings of the Sixth International Workshop on Machine Learning*, Segre, A. M (Ed), pp.441-443, 1989.

[16] Carbonell, J.G., "Introduction: Paradigms for Machine Learning", *Artificial Intelligence, An Int'l Journal*, Vol.40, No.1-3, Sept., 1989.

[17] Carpineto, C., "An Approach Based on Integrated Learning to Generating Stories from Stories", in *Proc. of the 5th Int'l Conf. on Machine Learning*, pp. 298-304, Morgan Kaufmann, Los Altos, Calif. 1988.

[18] Chan, P.K., "Inductive Learning with BCT", in *Proceedings of the Sixth International Workshop on Machine Learning*, Segre, A. M (Ed), pp.104-108, 1989.

[19] Chang, C.L. & A. Walker, " PROSQL: A Prolog Programming Interface with SQL/DS " , in *Expert Database Systems*, L. Kerschberg (ed.), Proceedings, First Intl. Workshop on Expert Database Systems, pp. 233-246, Kiawah Island, S.C., Oct. 1984.

[20] Charniak, E., C.K. Riesbeck & D.V. McDermott, *Artificial Intelligence Programming*, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1980.

[21] Chen, P., "The Entity-relationship Model: Toward a Unified View of Data", *ACM Transaction on Database Systems*, 1, 1, Mar., 1976. Reprinted in *Artificial Intelligence & Databases*, J. Mylopoulos and M.L. Brodie (eds.), pp. 98-111, Morgan Kaufmann, 1989.

[22] Cheng, Y. and K.-S. Fu, "Conceptual Clustering in Knowledge Organization", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.PAMI-7, No.5, 1985.

[23] Clark, P. and T. Niblett, "The CN2 Induction Algorithms", *Machine Learning*, Vol.3, pp.261-283, Kluwer Academic Publishers, 1989.

[24] Clearwater, S.H., T.-P. Cheng, H. Hirsh, and B.G. Buchanan, "Incremental Batch Learning", in *Proceedings of the Sixth International Workshop on Machine Learning*, Segre, A. M (Ed), 1989.

[25] CODASYL, "CODASYL Data Base Task Group Report", Conference on Data System Languages, ACM, New York, 1971.

[26] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", *ACM Comm.*, 13, 6, pp.377-387, 1970.

[27] Copeland, G. and D. Maier, "Making Smalltalk a Database System", ACM SIGMOO, 1984.

[28] Dahl, O., B. Myrhaug, and K. Nygaard, "Simula67 Common Base Language", Norwegian Computing Center S-2, 1968.

[29] Danyluk, A.P., "The Use of Explanations for Similarity-based Learning", *Proceedings of the Int'l Joint Conf. on Artificial Intelligence*, pp.274-276, IJCAI, 1987.

[30] DATAPRO Research Corp., "SYSTEM 2000 - MRI Systems Corporation", Datapro 70, 1972.

[31] Date, C.J., *An Introduction to Database Systems*, 2nd Edition, Addison-Wesley, 1977.

[32] Deering, M. and J. Faletti, " Database Support for Storage of AI Reasoning Knowledge", in *Expert Database Systems*, L. Kerschberg (ed.), Proceedings, First Intl. Workshop on Expert Database Systems, pp.527-536, Kiawah Island, S.C., Oct. 1984.

[33] DeJong, G. and Mooney, R., "Explanation-Based Learning: An Alternative View", *Machine Learning*, Vol.1, pp.145-176, 1986.

[34] Delcambre, L.M.L., "RPL: An Expert System Language with Query Power", *IEEE EXPERT*, pp.51-61, Winter, 1988.

[35] Dietterich, T.G., B. London, K. Clarkson and G. Dromoy, "Learning and Inductive Inference", in *The Handbook of Artificial Intelligence*, P.R. Cohen and E.A. Feigenbaum (eds.), pp.323-512, William Kaufmann Inc., 1982.

[36] Dietterich, T.G. and R.S. Michalski, "A Comparative Review of Selected Methods for Learning from Examples", in *Machine Learning, An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), pp.41-81, Morgan Kaufmann, Los Altos, 1983.

[37] Eick, C.F., R. Kochhar and S. Kumar, " DALI - A Knowledge Base Management System",in *The Proc. of the 1st Int'l Conf. on Industrial & Engineering Applications of A.I. and ES*, pp.837-846, 1988.

[38] Elio, R. and L. Watanabe, "An Incremental Deductive Strategy for Controlling Constructive Induction in Learning From Examples", *Machine Learning*, Vol.7, pp.7-44, Kluwer Academic Publishers, 1991.

[39] Ellman, T., "Explanation-Based Learning: A Survey of Programs and Perspectives", *ACM Computing Surveys*, Vol.21, No.2, pp.163-221, June 1989.

[40] Elmasri, R. and S.B. Navathe, *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, Inc. 1989.

[41] Fisher, D.H., "Knowledge Acquisition Via Incremental Conceptual Clustering", *Machine Learning*, Vol.2, pp. 139-172, Kluwer Academic Publishers, Boston, 1987.

[42] Frank, M., and M.L. Brodie, "On Knowledge-Based System Architectures", in *On Knowledge Management System*, Brodie, M.L. and J. Mylopoulos (eds), pp.35-54, Springer-Verlag, New York, N.Y., 1986.

[43] Gallaire, H. and J. Minker (eds.), *Logic and Databases*, Plenum Press, New York, 1978.

[44] Gallaire, H. , J. Minker and J. Nicolas (eds.), " Logic and Databases: A deductive approach", *ACM Computing Surveys*, Vol.16, No.2, pp.153-185, June 1984.

[45] Gams, M., M. Drobnic, and M. Petkovsek, "Learning from Examples - A Uniform View", *International Journal on Man-Machine Studies*, 34, pp.49-68, 1991.

[46] Gardarin, G. and P. Valduriez, *Relational Databases and Knowledge Bases*, Addison Wesley, 1989

[47] Gennari, J.H., P. Langley and D. Fisher, "Model of Incremental Concept Formation", *Artificial Intelligence, An Int'l Journal*, Vol.40, No.1-3, pp. 11-61,Sept., 1989.

[48] Goldberg, A., D. Robson, *Smalltalk 80: The Language and Its Implementation*, Addison-Wesley, Reading, 1983.

[49] Hammer, M., and Mcleod, D. "Database description with SDM: A semantic database model", *ACM Transactions on Database Systems*, Vol.6, No.3(Sept.), pp.351-386, 1981.

[50] Harel, D., "Review of Logic and Databases", H. Gallaire and J. Minker, *Computing Reviews 21*, No.8, Aug. 1980.

[51] Haussler, D., "Quantifying the Inductive Bias in Concept Learning", *Proc. of The 5th Nat'l Conf. on A.I.*, pp. 485-489, Philadelphia, PA, Morgan Kaufmann, 1986.

[52] Hayes-Roth, F., "Collected Papers on the Learning and Recognition of Structured Patterns", Technical Report, Carnegie-Mellon, Dept. of Computer Sci., Pittsburgh, PA, 1976a.

[53] Hayes-Roth, F., " Patterns of Induction and Associated Knowledge Acquisition Algorithms ", Technical Report, Carnegie-Mellon, Dept. of Computer Sci., Pittsburgh, PA, 1976b.

[54] Hayes-Roth, F. and J. McDermott, "Knowledge Acquisition from Structural Descriptions", *Proc. of the 5th Int'l Joint Conf. on Artificial Intelligence*, IJCAI, Cambridge, Mass., 1977.

[55] Hayes-Roth, F. and J. McDermott, "An Interference Matching Technique for Inducing Abstractions", *Communications of the ACM*, Vol. 21, No. 5, pp.401-411, 1978.

[56] Hayes-Roth, F., D.A. Waterman and D.B. Lenat (eds.), *Building Expert Systems*, Addison Wesley, 1983.

[57] Heiler, S., U. Dayal, J. Orenstein and S. Radke-Sproull, "An Object-Oriented Approach to Data Management: Why Design Database Need It", *The 24th ACM/IEEE Design Automation Conf.*, pp.335-340, 1987

[58] Hillyer, B.K., *On Applying Heterogeneous Parallelism to Elements of Knowledge-Based Data Management*, Ph.D. Dissertation, Columbia University, 1986.

[59] Holland, J.H., *Induction: Processes of Inference, Learning, and Discovery*, MIT Press, 1986.

[60] Hull, R. and R. King, "Semantic Database Modelling: Survey, Applications and Research Issues", *ACM Computing Surveys*, Vol.19, No.3, pp.201-260, Sept. 1987.

[61] IBM Corporation, "Information Management System/ Virtual Storage General Information Manual", IBM Form No. GH20-1260-3.

[62] Jarke, M. and Y. Vassiliou, " Coupling Expert Systems with Database Management Systems", in *Artificial Intelligence Applications for Business*, Walter Reitman (ed), pp.65-85, Ablex Publishing Corporation, 1984.

[63] Karagiannis, D. and H. Schneider, " Data- and Knowledge- Base Management Systems for Decision Support", in *Decision Support Systems: Theory and Application*, C.W. Holsapple and A.B. Whinston (eds.), pp.55-89, Springer-Verlag, Berlin Heidelgerg, 1987.

[64] Ke, M., M. Ali, "A Knowledge-Directed Induction Methodology for Intelligent Database Systems", *The International Journal of Expert Systems*, Vol.4, No.1, pp.71-115, JAI Press, 1991.

[65] Ke, M., M. Ali, "MLS, a Machine Learning System for Engine Fault Diagnosis", in *Proceedings of the First International Conference on Industrial and Engineering Applications of AI and Expert Systems*, 1988.

[66] Ke, M., M. Ali, "A Learning, Representation and Diagnostic Methodology for Engine Fault Diagnosis", in *Proceedings of the Second International Conference on Industrial and Engineering Applications of AI and Expert Systems*, 1989.

[67] Kerschberg, L., (ed.), *Expert Database Systems*, Proceedings, First Intl. Workshop on Expert Database Systems, Kiawah Island, S.C., Oct. 1984.

[68] Kolodner, J.L. , " Reconstructive Memory: A Computer Model ", *Cognitive Science*, Vol.7, pp. 281-328, 1983.

[69] Kubat, M., "Conceptual Inductive Learning: the Case of Unreliable Teachers", *Artificial Intelligence*, 52(2), pp.169-182, 1991.

[70] Larson, J., "Inductive Inference in the Variable-valued Predicate Logic System VL21: Methodology and Computer Implementation," Ph.D. dissertation, University of Illinois, Urbana, Illinois, May 1977.

[71] Larson, J. and R.S. Michalski, "Inductive Inference of VL decision rules," *Proceedings of the Workshop on Pattern Directed Inference System*, SIGART Newsletter 63, June 1977.

[72] Lebowitz, M., " Correcting Erroneous Generalizations", *Cognition and Brain Theory*, Vol.5, pp.367-381 1982.

[73] Lebowitz, M., "Concept Learning in a Rich Input Domain: Generalization-base Memory", in *Machine Learning, An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Vol.II, pp. 193-214, Morgan Kaufmann, Los Altos, CA, 1986a.

[74] Lebowitz, M., "Generalization and Memory in an Integrated Understanding System", Technical Report 186, Dept. of Computer Sci., Yale Univ., 1980 (Ph.D. diss., 1980).

[75] Lebowitz, M., "Integrated Learning: Controlling Explanation", *Cognitive Science*, pp.219-240, 10. 1986b.

[76] Lebowitz, M., "Experiments with Incremental Concept Formation: UNIMEM", *Machine Learning*, 2. (1987), pp. 103-138.

[77] Lecluse, C., P. Richard and F. Velez, "O2 An Object-Oriented Data Model", in *Advances in Database Programming Languages*, Bancilhon, F. and P. Buneman, ACM Press, 1990.

[78] Lenat, D.B., "The Role of Heuristics in Learning by Discovery: Three Case Studies", in *Machine Learning, An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), pp.243-306, Morgan Kaufmann, Los Altos, 1983.

[79] Leung, Y. Y. and D.L. Lee, "Logic Approaches for Deductive Databases", *IEEE EXPERT*, pp.64-75, Winter, 1988.

[80] Levene, M. and A. Poulovassilis, "An Object-Oriented Data Model Formalized through Hypergraphs", in *Data and Knowledge Engineering*, 6, pp.205-224, 1991.

[81] Lloyd, J. and R. Topor, "A Basis for Deductive Database", *Journal of Logic Programming*, Vol.2, No.2, July 1985.

[82] Levesque, H.J., "Knowledge Representation and Reasoning", Ann. Rev. Comput. Sci. 1986, Reprinted in *Artificial Intelligence & Databases* , J. Mylopoulos and M.L. Brodie (eds.) pp.35-51, Morgan Kaufmann, 1989.

[83] Macdonald, B.A. and I.H. Witten, "A Framework for Knowledge Acquisition through Techniques of Concept Learning", *IEEE Trans. on System, Man, and Cybernetics*, vol.19, No.3, pp.499-512, May/June, 1989.

[84] Maier, D., J. Stein, A. Otis & A. Purdy, "Development of an Object-oriented DBMS", in *Proc. of OOPSLA*, pp.472-482, Sept., 1986.

[85] Medin, D.L., W.D. Wattenmaker & R.S. Michalski, "Constraints and Preferences in Inductive Learning: An Experimental Study of Human and Machine Performance," *Cognitive Science* 11, pp.299-339, 1987.

[86] Meyer, B., *Object-Oriented Software Construction*, Prentice Hall, Hertfordshire, UK, 1988.

[87] Michalski, R.S. , " Pattern Recognition as Rule-guided Inductive Inference ", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.PAMI-2, No.4, pp. 349-361, 1980a.

[88] Michalski, R.S., "Knowledge Acquisition through Conceptual Clustering: A Theoretical Framework and Algorithm for Partitioning Data into Conjunctive Concepts", *Int'l Journal of Policy Analysis and Information Systems*, Vol4, pp.219-243 (A special issue on Knowledge Acquisition and Induction), 1980b.

[89] Michalski, R.S. and R.E. Stepp, "Concept-based Clustering versus Numerical Taxonomy", Technical Report 1073, Dept. of Computer Sci., Univ. of Illinois, 1981.

[90] Michalski, R.S. and R.E. Stepp, "Learning from Observation: Conceptual Clustering", in *Machine Learning, An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), pp.331-364, Morgan Kaufmann, Los Altos, 1983a.

[91] Michalski, R.S., "A Theory and Methodology of Inductive Learning", in *Machine Learning, An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), pp.83-134, Morgan Kaufmann, Los Altos, 1983b.

[92] Michalski, R.S. and R.E. Stepp, "Automated Construction of Classifications: Conceptual Clustering Versus Numerical Taxonomy", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No.4, pp. 396-410, July, 1983c.

[93] Michalski, R.S., I. Mozetic, J. Hong, and N. Lavrac, "The Multi-purpose Incremental Learning System AQ15 and its Testing to Three Medical Domains". in *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp.1041-1045, Morgan Kaufmann, 1986a.

[94] Michalski, R.S., "Understanding the Nature of Learning: Issues and Research Directions", in in *Machine Learning, An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Vol.II, pp.3-25, Morgan Kaufmann, 1986b.

[95] Michalski, R.S., "How to Learn Imprecise Concepts: A Method for Employing a Two-Tiered Knowledge Representation in Learning", in *Proceedings of the Fourth International Machine Learning Workshop*, pp.50-58, 1987.

[96] Miller, R.K., CMfgE, T.C. Walker, *Artificial Intelligence Application in Research and Development*, Prentice Hall, 1988.

[97] Missikoff, M. & G. Wiederhold , " Towards a Unified Approach for Expert and Database Systems " , in *Expert Database Systems* , L. Kerschberg (ed.), Proceedings, First Int'l. Workshop on Expert Database Systems, pp.383-399, Kiawah Island, S.C., Oct. 1984.

[98] Mitchell, T.M., "The Need for Biases in Learning Generalizations", Technical Report CBM-TR-117, New Brunswich, NJ, Rutgers Univ., Dept. of Computer Sci., 1980.

[99] Mitchell, T.M., "Goal Directed Learning", *Proceedings of the 2nd Int'l Workshop on Machine Learning*, pp.117-118, 1983.

[100] Mitchell, T.M., R.M. Keller and S.J. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View", *Machine Learning*, Vol.1, pp.47-80, 1986.

[101] Modesitt, K.L., "Experience with Commercial Tools Involving Induction on Large Databases for Space Shuttle Main Engine Testing", *The 4th Int'l Expert Systems Conf.*, pp. 1-9, London, UK, June, 1988.

[102] Mortimer, H., *The Logic of Induction*, Ellis Horwood, Chichester, 1988.

[103] Mylopoulos, J. and M.L. Brodie (eds.), *Artificial Intelligence & Databases*, Morgan Kaufmann, 1989.

[104] Nunez, M., "The Use of Background Knowledge in Decision Tree Induction", *Machine Learning*, Vol.6, pp.231-250, Kluwer Academic Publishers, 1991.

[105] Parsaye, K., M. Chignell, S. Khoshafian, and H. Wong, *Intelligent Databases*, John Wiley & Sons, Inc., 1989.

[106] Peckham, J., and F. Maryanski, "Semantic Data Models", *ACM Computing Surveys*, Vol.20, No.3, pp.153-189, Sept. 1988

[107] Quinlan, J.R., "Learning Efficient Classification Procedures and Their Application to Chess end Games", in *Machine Learning, An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), pp.463-482, Morgan Kaufmann, Los Altos, 1983.

[108] Quinlan, J.R., "Induction of Decision Trees", *Machine Learning*, Vol.1, pp.81-106, Kluwer Academic Publishers, Boston, 1986.

[109] Quinlan, J.R., "Generating Production Rules from Decision Trees", in *Proceedings of the*

Tenth International Joint Conference on Artificial intelligence, pp.304-307, 1987.

[110] Quinlan, J.R., "Learning Logical Definitions from Relations", *Machine Learning*, Vol.5, pp.239-266, Kluwer Academic Publishers, 1990.

[111] Reinke, R., and R.S. Michalski, "Incremental Learning of Concept Descriptions", *Machine Intelligence*, Vol.11, Oxford Univ. Press, 1986.

[112] Reiter, R., "On Closed World Databases", in *Logic and Data Bases*, Gallaire, H. and Minker, J. (eds.), New York, Plenum Press, 1978. Reprinted in *Artificial Intelligence & Databases*, J. Mylopoulos and M.L. Brodie (eds.) pp.248-258, Morgan Kaufmann, 1989.

[113] Rendell, L., "A General Framework for Induction and a Study of Selective Induction", *Machine Learning*, Vol.1, 177-226, 1986.

[114] Rendell, L.A., R.M. Seshu, and D.K. Tcheng, "Dynamically-variable Bias Management for Robust Concept Learning", in *Proc. of the 10th Int'l Joint Conf. on A.I.*, Milan, Italy, Morgan Kaufmann, 1987a.

[115] Rendell, L. A., R.M. Seshu, and D.K. Tcheng, "More Robust Concept Learning Using Dynamically-variable Bias ", in *Proc. of the 4th Int'l Workshop on Machine Learning*, pp. 66-78, 1987b.

[116] Rine, C.D., "Improving Software Productivity: An Expert Database Approach", *IEEE EXPERT*, pp.16-32, Summer, 1988.

[117] Risch, T., R. Reboh, P. Hart, and R, Duda, "A Functional Approach to Integrating Database and Expert Systems", *Communications of the ACM*, Vol.31, No. 12, pp.1424-1437, Dec. 1988.

[118] Russell. S.J. and B.N. Grosof, "A Declarative Approach to Bias in Concept Learning," Proceedings of The AAAI, pp.505-510, 1987.

[119] Shaw, M. J. and J.A. Gentry, "Inductive learning for Risk Classification", *IEEE EXPERT*, pp.47-53, Feb. 1990.

[120] Schlimmer, J.C. and D.H. Fisher, "A Case Study of Incremental Concept Induction", in *Proc. of The 5th Nat'l Conf. on A.I.*, pp. 496-501, Philadelphia, PA, Morgan Kaufmann, 1986.

[121] Schlimmer, J.C. and R.H. Granger Jr., " Incremental Learning from Noisy Data", *Machine Learning*, Vol.1, pp. 317-354, 1986.

[122] Schlimmer, J.C. , "Incremental Adjustment of Representations for Learning", *Proc. of The 4th Int'l Workshop on Machine Learning*, pp. 79-90, Univ. of Calif. Irvine, 1987.

[123] Schmidt, J.W. & C. Thanos (eds.), *Fundamentals of Knowledge Base Management*

Systems, Springer-Verlag, New York, 1988.

[124] Shin, D. and P.B. Berra, "An Architecture for Very Large Rule Bases Based on Surrogate Files", in *Database Machines and Knowledge Base Machines*, M. Kitsuregawa and H. Tanaka (eds), pp.557-570, Klumer Academic Publishers, 1988.

[125] Shipman, D.W., "The Functional Data Model and The Data Language DAPLEX", *ACM Transactions on Database Systems*, Vol.6, No.1(Mar.) pp.140-173, 1981.

[126] Smith, J.M., "Expert Database Systems: A Database Perspective", in *Expert Database Systems*, L. Kerschberg (ed.), Proceedings, First Intl. Workshop on Expert Database Systems, pp.3-14, Kiawah Island, S.C., Oct. 1984.

[127] Smith, J.M., and D.S.P. Smith, "Database Abstractions: Aggregation and Generalization",in *Artificial Intelligence & Databases* , J. Mylopoulos and M.L. Brodie (eds.) pp.138-153, Morgan Kaufmann, 1989.

[128] Spangler, S., U.M. Fayyad, and R. Uthurusamy, "Induction of Decision Trees from Inconclusive Data", in *Proceedings of the Sixth International Workshop on Machine Learning*, Segre, A. M (Ed), pp.146-150, 1989.

[129] Stepp, R.E., "Conjunctive Conceptual Clustering: A Methodology and Experimentation", Ph.D. Dissertation, Department of Computer Sci., Univ. of Illinois at Urbana-Champaign, 1984.

[130] Symbolics Inc., *Statice*, Aug., 1988.

[131] Thompson, K., and P. Langley, "Incremental Concept Formation with Composite Objects", in *Proceedings of the Sixth International Workshop on Machine Learning*, Segre, A. M (Ed), 1989.

[132] Tsichritzis, D.C. and F.H. Lochovsky, "Hierarchical Data Base Management: A Survey", *ACM Computing Survey*, Vol.8, No.1, 1976.

[133] Tsur, S. and C. Zaniolo, "LDS: A Logic-based Data-Language", Proc. 12th Int'l Conf. *Very Large Databases*, Morgan Kaufmann, Los Altos, Calif., 1986.

[134] Tsur, S., "LDS - A Technology for the Realization of Tightly Coupled Expert Database Systems", *IEEE EXPERT*, pp.41-51, Fall, 1988.

[135] Ullman, J.D., *Principles of Database Systems*, 2nd Edition, Computer Science Press, Inc., 1982.

[136] Ullman, J.D., *Principles of Database and Knowledge-Base Systems* , Vol.I, Computer Science Press, Inc., Potomac, MD, 1988.

[137] Unland, R. and G. Schlageter, "Object-Oriented Database Systems: Concepts and Perspectives", in *Database Systems of the 90s*, A. Blaser (Ed.), Springer-Verlay, 1990.

[138] Utgoff, P.E., "Adjusting Bias in Concept Learning", in *Proc. of the 8th Int'l Joint Conf. on A.I.*, Karlsruhe, West Germany, pp. 105-109, Morgan kaufmann, 1983.

[139] Utgoff, P.E., *Machine Learning of Inductive Bias*, Kluwer Academic Publishers, 1986a.

[140] Utgoff, P.E., "Shift of Bias for Inductive Concept Learning", in *Machine Learning, An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Vol.II, pp. 107-147, Morgan Kaufmann, Los Altos, CA, 1986b.

[141] Utgoff, P., "ID5: An Incremental ID3", in *Proceedings of the Fifth International Conference on Machine Learning*, pp.107-120, Morgan Kaufmann, 1988.

[142] Vere, S.A., "Induction of Concepts in the Predicate Calculus", *Proc. of the 4th Int'l Joint Conf. on Artificial Intelligence*, IJCAI, pp.281-287, Tbilisi, USSR, 1975.

[143] Vere, S.A., "Induction of Relational Productions in the Presence of Background Information", *Proc. of The 5th Int'l Joint Conf. on A.I.*, Cambridge, Mass., pp.349-355, 1977.

[144] Vere, S.A., "Inductive Learning of Relational Productions", *Pattern-Directed Inference Systems*, Waterman, D.A. and Hayes-Roth, F. (eds.), Academic Press, New York, 1978.

[145] Vere, S.A., "Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions", *Artificial Intelligence*, Vol.14, No.2, pp. 139-164, Sept. 1980.

[146] Wang, T.-W., and J.D. Birdwell, "Intelligent Monitoring, Analysis, and Control of Process Operations Using Multivariate Statistical Tools", Research Proposal, submitted to the Intelligent Control Program Initiative, NSF/Electric Power Research Institute, April, 1992.

[147] Winston, P.H., "Learning Structural Descriptions from Examples", Technical Report, AI-TR-231, MIT, Cambridge, Mass., Sept. 1970.

[148] Winston, P.H., "Learning Structural Descriptions from Examples", *The Psychology of Computer Vision*, Winston, P.H. (ed.), McGraw Hill, New York, 1975.

[149] Yang, C.-C., *Relational Databases*, Prentice-Hall, 1986.

[150] Zdonik, S.B. and D. Maier, *Readings in Object-Oriented Database Systems*, Morgan Kaufmann Publishers, Inc. 1990.

APPENDIXES

APPENDIX A.

DEDUCTIVE MECHANISM

The deductive subsystem of the KNOWD methodology supports a declarative representation of the domain knowledge. The basic unit need to represent knowledge is a production rule. Using a production rule to represent domain knowledge has the following advantages: (1) production rules are in a form close to natural language, so knowledge can be easily formulated and added; (2) knowledge is not embedded in the program so it is easily examined and modified. The knowledge base (containing domain related facts and rules) is kept separated from other modules, so that greater domain independence is achieved. Knowledge management facilities, such as viewing, creation, deletion and modification, have also been developed.

A constructive rule is in the following format:

IF <ANT> THEN <KEYWD> <CONS>

IF <ANT> ACTION <FUNCTIONS>

where ANT is the antecedent, an expression of features of an instance, CONS is the consequent, an expression of new features, KEYWD can be ADD or REPLACE, ADD means to add the CONS into the feature list of the instance, and REPLACE means that ANT is to be replaced by CONS. FUNCTIONS is a list of executable functions which are defined by the system or users and which generate new features. The second form of constructive rules is necessary because there are situations in which it is very awkward to represent the semantics of a rule in a declarative way. For example, in the execution of the generating-chain-property rule:

IF ((start-time (?sensor) = ?value))

ACTION ((add-minmax 'first))

a procedure **add-minmax** is called, which finds the first started sensor S1 in the instance description. The procedure involves finding the smallest starting time value of all the sensors and generating a new feature:

(first-start-time = S1)

and a new attribute **first-start-time** is also created.

A constraint rule is in the format:

IF <ANT> THEN <CONST>

where ANT is the same as above, and CONST is the constraint expression of predicates that must be true when ANT is true. If CONST is not true, the induction system will take proper action to deal with the inconsistent instance. Unlike constructive rules, which can be applied to all concepts, the constraint rules are related to a specific concept. The concept object contains a slot representing its constraints. When accepted by the system, a constraint rule is translated into the form:

IF <ANT> & ¬<CONST>

ACTION <CONSTRAINT-PROCESSOR>

which is of the second form of constructive rules, so the same deductive mechanism can be used for both constructing new features and enforcing the concept constraints.

A forward-chaining mechanism is selected over the backward-chaining mechanism for the deduction control. Backward-chaining is a goal-directed reasoning. In the task of generating new features for a concept, a large number of possible goal statements may be generated with a set of deductive rules. If we try to verify all these possible goal statements with backward-chaining, it would be very inefficient because only a small number of these goal statements can be verified to be true. Forward-chaining, however, always generates true statements which can be included in the description of a concept.

For each concept instance with the transformed description d_t, all enabled deductive rules are used as candidate rules, and the Known-Fact Base (KFB) is initialized to contain all the features in d_t. The deduction procedure repeats the following steps until the list of candidate rules

become empty:

(1) Fetch a candidate rule from the list;

(2) Try to execute the rule.

Each fetched deductive rule is processed as follows:

(1) Try to match the if-condition part with the features in KFB in various ways.

(2) If there is no match, then return.

(3) If there are matches, and rule has a then-part, then for each match perform the following operations:

a. Using the substitution to replace variables in the then-part;

b. Evaluating any functions in the then-part;

c. Adding the new feature (i.e., the processed then-part) into KFB;

d. Using the new feature as indices to fetch more deductive rules as candidate rules.

(4) If there are some matches and the rule has an action-part, then execute all the actions.

For the efficiency of the deduction process, each deductive rule is indexed into a discrimination net by the features in its if-condition part. When new features are deduced, the new facts can be used as indices to fetch further deductive rules as candidate rules. The KFB is also organized into a discrimination net which facilitates the matching process.

APPENDIX B.

THE INDUCTIVE BIAS MANAGEMENT SYSTEM

Inductive bias management is an important part of the KNOWD methodology. In this appendix we first introduce the concept of inductive bias; second, we explain that in an inductive database system a bias management system is indispensable; third, we describe various biases incorporated in the KNOWD inductive model; and finally, we describe the management strategy and organization of the Bias Management System (BMS).

B.1. Introduction to the Inductive Biases

For a given set of concept instances, too many consistent concept descriptions might be generated by an inductive process that has no constraints. For example, given

{ 1, 3, 5 } as a positive instance

{ 6, 8, 10 } as a negative instance

The consistent hypotheses might be

1) A set of numbers which are less than 5.5 .

2) A set of integers which are less than 6 .

3) A set of positive integers which are less than 6 .

4) A set of odd integers which are less than 6 .

5) A set of odd integers which are greater than zero.

6) A set of integers for which the sum is 9.

7) A set of integers for which the sum is less than 10.

8) A set of integers for which the sum is an odd number.

9) A set of integers in which no one has two different prime factors.

The inductive process can be viewed as a search process through the hypothesis space. Since the hypothesis space is usually very large, it is intractable to find the correct inductive

results without any heuristics. For efficient searching, the biases are required to make the induction process intelligent and feasible. It has even been said that without bias there is no induction [Utgoff 1986a].

An inductive bias is a heuristic that can be used to guide the inductive process. When performing an induction, human beings usually depend not only on the data of available instances but also on some background knowledge and past experience. These kinds of knowledge can be treated as inductive biases. With those biases, an inductive system can concentrate on promising hypotheses and eliminate unpromising hypotheses. For instance, in the above example if we know in a domain that all the elements in the sets are integers, then many useless hypotheses can be avoided. There are many types of inductive biases which have different functions in the inductive process. Examples of various types of inductive biases are listed in Table B.1. The control types are discussed in Section B.4. and the functions of biases are discussed below.

Inductive biases have three functions. First, biases can be used for forming hypotheses. For example, biases can be used to choose generalization rules. Different rules are given different worth values and the worth values are determined by the relevancies of attributes, characteristics of attribute values related to a domain, the usage pattern of the rules and the formats of the resultant descriptions. Biases can also be used to determine how to apply a rule, such as choosing an expression to drop in executing the dropping-condition rule (discussed in Chapter 3).

Second, biases can be used for choosing hypotheses. For example, preference can be based on the syntax of the hypothesis (such as the length of a disjunct and the number of disjuncts) and the weights of the attributes involved.

Third, biases can be used to change themselves. For example, a bias can be used to change the relevancy value of an attribute, to change the preferred length of the descriptions, or to change the worth value of a heuristic rule based on certain statistics.

Table B.1. Biases in the Inductive Database System

INDUCTIVE BIAS	FUNCTION	CONTROL TYPE
worth of attributes	1, 2	s, v
types of attributes	1	f
available attributes	1	s, v
transformation rules of attributes	1	s
value domains of attributes	1, 2	s
hierarchies on attribute values	1, 2	s
query count of attributes	3	v
forms of atomic expressions	1, 2	f
types of internal operations	1	f
value types	1, 2	f
matching criteria	1, 2	f
component-concept relevancies	1, 2	s, v
query count on component-concept relevancies	3	v
computing worth of expression	1, 2	f, s
attribute-attribute relationships	1, 2	s, v
component-component relationships	1, 2	s, v
computing atom worth	1, 2	f, s
threshold to choose expressions	1, 2	s
available deductive rules	1	s
available generalization rules	1	f, s
initial worth of rules	1	f, s
computing worth of rules	1	f, s, v
states of rules	1	f, s
usage count of rules	3	v
clustering control algorithm	1, 2	f, s
clustering criteria	1, 2	f
computing matching factors	1, 2	f
concept constraints	1	s
formalism of description language	1	f
preference of descriptions	2	f, s, v

FUNCTION:
1 -- Forming hypotheses
2 -- Choosing hypotheses
3 -- Changing heuristics

CONTROL TYPE:
f -- Fixed
s -- Settable
v -- Variable

B.2. Inductive Bias Management

Inductive biases are the knowledge, including domain specific knowledge and general inductive knowledge, utilized by an inductive system. In order to utilize this kind of knowledge, people have to specify it and transform it into a form that can be processed by a program. A bias management system can help the bias acquisition process and bias transformation. A bias management system also supports the representation of bias and automatic bias shifting in case the initial biases are not suitable.

Limited research work has been done in the area of inductive bias management. Utgoff [1986a] presented an inductive learning program, STABB, that shifts its bias during the course of learning from examples. The only bias included in STABB is that of using a restricted language within a fixed formalism. The shifting is in a fixed order from a strong bias to a weaker bias. Utgoff proposed a framework for shifting to a weaker bias, but domain related knowledge is not considered in his method, and the range of biases is limited.

Rendell et al. [1987] described a variable bias management system (VBMS) which learns the best bias for different types of induction problems. VBMS dynamically alters the concept representation language and learning algorithms by monitoring progress and selecting biases based on the characteristics of the particular induction problems presented. The implementation has limited capabilities: (1) problem-space features considered are only the number of instances and the number of attributes, (2) bias includes only three different coarse algorithms, (3) speed is the only performance measure, (4) there is no guarantee that the dissimilarity of problems based on a measure of utility (which is a vector with each element the speed of an algorithm) corresponds to problem characteristics, (5) the process of searching for the best bias is not performed at the same time as the induction process. In order to get statistics on performance of bias, many runs of the inductive system are required (for example, 3 problem types and 3 algorithms require 9 runs).

Many inductive learning systems embed the inductive biases in their program and have no explicit bias management capabilities. However, conceptual induction needs a different set of

biases even for closely related domains. The same instance of an entity can be used to form different concepts according to the goal of induction. Even in the same domain, different tasks need a different set of biases. For example, in the domain of classifying countries in the world, we may have different domain tasks: classify them geographically, economically, or politically. For different classifications, features should be of different importance, which is a kind of bias. For an inductive database system a relatively larger range of domains is expected, which makes it impossible to embed all biases in the induction system for different applications. Even for a specific domain application users often find it difficult to specify clearly and correctly all the necessary biases. Much burden in testing an induction program lies in identifying the right set of biases.

For the above reasons we advocate that an explicit bias management system should be an essential component of an integrated inductive database system as shown in Figure B.1. For an inductive database, both instance data and biases are required inputs. In order to assist users in bias acquisition, bias transformation and automatic bias shifting, a bias management system is needed to manage a large range of biases. The functions of a bias management system should include:

(1) *Bias acquisition.* It facilitates the specification of biases by users.

(2) *Bias generation.* It automatically generates certain biases through other system parameters.

(3) *Bias modification.* It facilitates the modification of biases by users.

(4) *Bias explanation.* It explains biases to users and helps users to understand the effects of biases.

(5) *Automatic bias shifting.* It automatically changes biases for a better performance of induction.

(6) *Bias enforcement.* It enforces biases in the induction process.

The objective of a bias management system is to help users build and maintain necessary

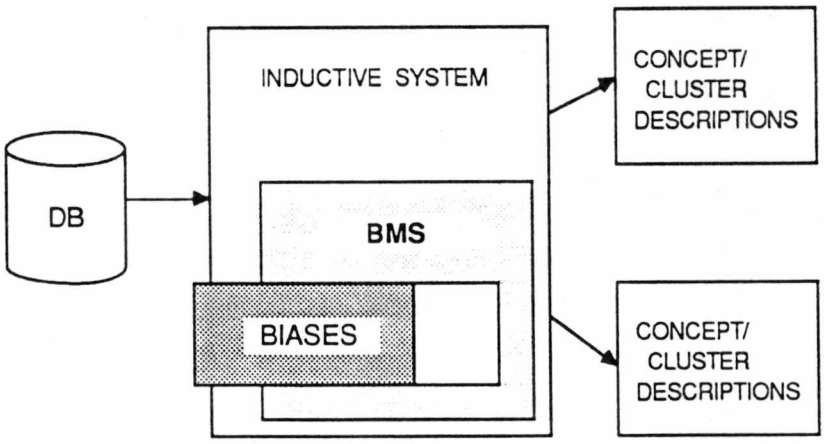

Figure B.1. BMS, an essential component of an inductive database system

biases to achieve higher performance in an inductive system. We have developed an inductive bias management system (BMS) which provides the functions given above, and we have incorporated it into the KNOWD methodology. The next section will discuss various types of biases needed by an inductive system.

B.3. Types of Inductive Biases

As heuristic knowledge, biases have three functions: (1) heuristics used to form hypotheses, (2) heuristics used to choose hypotheses, and (3) meta heuristics used to change heuristics (Table B.1). Each function is fulfilled by biases of many types. For example, in order to form promising hypotheses efficiently, we need to know what attributes are useful, what potential relationships among attributes exist, what operations are allowable, how to choose an operation, how to execute an operation, and when to execute an operation. This section describes biases in many types of our inductive system.

B.3.1. Representation Formalism

Every induction system has a representation formalism for describing the concepts. The representation formalism can be an attribute vector space, attribute sets, first-order logic, or a limited form of first-order logic. Biases in representation formalism constrain the manner in which the allowed vocabulary can be used to construct concept descriptions. Generally speaking, the more general and the more flexible the representation formalism is, the more types of hypotheses it will be able to represent. In this case, more time will be spent on unpromising hypotheses. Most inductive systems have restrictions on their representation formalisms, such as using only conjunctions, limiting forms of attribute values, or using simple forms of atomic expressions, which aim at making the inductive process more efficient. The best trade-off is to preserve representation capability and promote efficiency as much as possible.

The bias on the representation formalism in the KNOWD model is:

(1) Only logically related features are combined into an expression which is an element of the description of a concept.

(2) The maximum level of embedding in a logical expression is two.[1]

The descriptions have limited logical expressions. At most, two levels of embedding expression can be represented, processed and generated, so processing is more efficient and the resultant descriptions are more easily understood. Descriptions can be conjunctions and disjunctions. Internal conjunctions and disjunctions are also allowed.

The KNOWD model allows only logically related attributes connected by logical operations (conjunctions and disjunctions). The hypothesis space is reduced drastically because incorrect combinations of attributes are avoided. This bias makes high efficiency possible for incremental induction with a rich representation capability.

---

[1] An Example of two-level embedding in a logical expression is given below:

$$((A1 \wedge A2) \vee (A3 \wedge A4 \wedge A5))$$

where Ai is an atomic expression.

B.3.2. Vocabulary

The vocabulary of a representation language is domain related. Different domains have different concepts and attributes to describe them. Vocabulary, including concept names, attribute names, attribute values, component names and relationships, is the set of symbols which appear in instance and concept descriptions.

Besides the choice of the vocabulary, the KNOWD methodology allows bias on the vocabulary as follows:

(1) Not all attributes are of the same importance; they may be given different worth values to represent different importance.

(2) Components have different relevance to different concepts.

(3) Attributes or components may have relationships.

(4) Attributes' values may have logical relationships.

(5) New terms for attributes or relationships may be generated.

The worth of attributes and relevancies of components are used to compute the worth of expressions and matching factors of expressions. (The computation method is discussed in Chapter 3.) A human usually classifies objects based on a few important attributes of the object instead of the whole set of attributes of the object. In many domains concept descriptions contain features of the concept components. Some component features are more important to a concept than others because of the different relevancies of the components to a concept.

Types of attributes are fixed in the KNOWD methodology because the algorithms to process the different attribute types are assumed to be embedded in the program implementation.

B.3.3. Hypotheses

Efficiency is not only an implementation problem but also a problem of how intelligent a system is in selecting hypotheses at each level of the induction process. The induction process comprises many levels of abstraction and generalization. At each level, many possible intermediate hypotheses may be generated. These hypotheses present different properties of concepts in various logical expressions. Human beings always have certain preferences about the concept

descriptions induced from a set of instances. The inductive database system is used to help a human to understand and comprehend the concepts implied by the data. Therefore, the inductive system needs some criteria to choose the promising intermediate hypotheses for further processing. If many final descriptions about a concept are generated by an inductive system, there is also a need to rank the resultant descriptions for users to consider.

In the KNOWD methodology, hypothesis selection criteria are similar to Michalski's preference criteria [1983a], which are based on the syntactic and semantic features of the descriptions, such as (1) fitness of the description calculated by the total sparseness[2], relative sparseness and projected sparseness, (2) simplicity of the description calculated by the number of disjuncts, average length of disjuncts and number of attributes, (3) importance of attributes and relationships (of attributes and objects), (4) completeness of the description calculated by the number of instances covered by the description, and (5) consistency of the description determined by the number of instances covered by other classes. In the KNOWD methodology, some examples of criteria based on syntax can be given as follows:

(1) Conjunctions get higher preference than disjunctions.

(2) Higher preference is given to simpler descriptions.

For example, in the KNOWD methodology atomic descriptions get higher preference than compound descriptions. Since conjunctions are more easily comprehended than disjunctions [Medin et al. 1987], they are given higher preference. Shorter descriptions are preferred to longer ones, and internal disjunctions are preferred to external disjunctions. A single-type value is more desirable than a range-type value, and the or-type value is the least desirable.

Preference on hypotheses depends not only on the syntactic aspects of the descriptions but also on semantic aspects of the concept descriptions. There are two semantic aspects: the extent and the intent of a concept. The extent of a concept is the inclusion of instances, and the

[2] The total sparseness is the sum of the sparseness of all clusters. The sparseness of a cluster is the number of unobserved events covered by the description of the cluster. The relative sparseness of a cluster is the ratio of the sparseness of the cluster to the total number of events covered by the description of the cluster. The projected sparseness of cluster is the sparseness computed in a subspace of the original event space defined by selected variables.

intent of a concept is the meaning stated in features of the concept. In the extent preference depends on the consistency and completeness of a concept description, that is, how many negative instances are excluded and how many positive instances are included. In the intent preference depends on the essential components, attributes and relationships of a concept. In the KNOWD methodology biases of hypothesis-preference on semantic aspects of concepts are as follows:

(1) The description with important attributes is preferred.

(2) The description with more relevant components is preferred.

(3) The description with more important relationships is preferred.

(4) The characteristic description with higher consistency factor is preferred.

(5) The discriminant description with higher completeness factor is preferred.

B.3.4. Clustering

Instances of objects are clustered together based not only on the number of common attributes but also on the attributes themselves and the global improvement of clustering. A measure called matching factor is used in the KNOWD methodology to represent the similarity of two instances. The matching factor of two instances is determined by

(1) common features,

(2) worth of expressions, and

(3) closeness of the values of the matching expressions,

where the worth of an expression is determined by the worth of attributes and components in the expression, attribute types, attribute domains, length of the expression, and logical forms of the expression.

When selecting a cluster into which a new instance is going to be incorporated, the algorithm depends on the global improvement of the clustering's capability of discrimination and prediction by maximizing three factors which include:

(1) discrimination factor,

(2) coherence, and

(3) description simplicity.

B.3.5. Operations

Operations on concept descriptions include generalization rules, transformation rules, specialization rules, and constructive rules. Transformation rules are usually domain related, such as changing representation of a continuous process into discrete events, using aggregate value to replace many specific values, replacing numerical values by symbolic values, and extracting patterns from lower level data. The biases on operations include:

(1) Availability of operations.

(2) Activating conditions of operations.

(3) Profit of operations.

There are several levels of abstractions for the inductive process, and on each level many operations may be available. If we equally apply all available operations to the input instances, too many hypotheses will be generated, and many of them may be unimportant and useless. In order to generate only promising hypotheses, we need to choose better operations for a special situation. Heuristics for choosing induction rules are based on the worth value of each rule which is determined by three factors: strength, specificity and support [Holland et al. 1987].

In the KNOWD methodology, different priorities are assigned to different generalization rules, and each generalization rule has a state flag which allows a user to enable or disable the rule for execution. The activating conditions are set by the system for each generalization rule, and attached to each rule is a worth-computing function. For each situation, all the applicable generalization rules constitute a conflict set, in which each generalization rule's worth is computed. Then, the rule with the highest worth is found and executed. In the KNOWD methodology the worth value of a generalization rule is determined not only by the rule itself, but also by the descriptions to be generated. Factors determining the worth of a rule include:

(1) Strength of a rule is determined by the past experience of the inductive system, which can be calculated by statistics of the usage in the accepted hypotheses.

(2) Specificity of a rule is determined by the scope of the rule condition part and the extension of the generalization descriptions, which can be compared by the

abstraction level of attributes, the range of the attribute values, and the logical relationships of the predicates.

(3) Worth values of the resultant descriptions are determined by the syntax.

Transformation rules and deductive rules are created by users based on their knowledge about the domain. The states of deductive rules are also controlled by users.

B.3.6. Meta Heuristics

Our BMS can automatically change heuristics by using meta heuristics. This kind of automatic bias shifting is a learning capability which allows the inductive system to learn better biases for its own use. The learning techniques are based on a user's query pattern, a user's feedback judgments of the resultant descriptions, and the usage pattern of the induction rules. The following are examples of meta heuristics employed in the KNOWD methodology:

(1) The strength of an induction rule is increased when the rule is used for a new resultant description.

(2) The worth of an attribute is increased when the attribute appears in a user's query.

Besides changing the worth of attributes, users' queries may suggest changes to the worth of relationships, relevancies of components of a concept, and preference of some concept descriptions over others. New attributes and new relationships may be introduced by BMS.

B.4. The Bias Management Strategy and Organization

BMS (Bias Management System) is incorporated into the KNOWD inductive methodology as a fundamental component of our inductive database system. BMS is the first system that supports a comprehensive management of a large variety of inductive biases with many functions.

The management strategy of BMS employs three types of controls of bias management: fixed bias management, settable bias management and variable (automatic shifting) bias management, which correspond to the three types of bias management strategies discussed by Rendell et al. [1987]. But, as mentioned earlier, Rendell's system supports only very limited

types of biases and functions. Fixed bias management embeds different types of biases in the program, and the biases are fixed at the system design time. Settable bias management supplies facilities which allow the user to choose biases or change the parameter values of biases (this type of biases is called valued biases) at the beginning of each run. Variable bias management automatically changes some biases by using meta heuristics during the run time.

In the design of the inductive database system we cannot embed all biases in the system because many types of biases are domain related. What we need is a new strategy combining the above three strategies so that it is a full bias management system which provides facilities for bias definition, bias maintenance, bias viewing, bias explanation, bias selection and bias enforcement. BMS has been developed to fulfill these requirements.

BMS follows a declarative style for representing biases, so that biases can be easily evaluated, created, selected, modified or initialized by a user. We cannot expect the user to supply all the correct biases and domain problem characteristics. Therefore, BMS will incrementally shift its biases based on the users' query pattern, on statistics of the resultant descriptions, and on the inductive process.

Fixed biases (such as the available types of attributes, forms of atomic expressions, allowed types of internal operations, defined forms of attribute values, available generalization rules, computation of worth of generalization rules, clustering control algorithm, clustering criteria) mainly include general inductive heuristics which are domain independent and heuristics which are applicable to all the related domains concerned. Although users cannot create fixed biases, they can use them selectively. For example, the generalization rule can be enabled or disabled and algorithms can be controlled by setting their parameters. BMS gives explanations of fixed biases and helps developers to select and control fixed biases.

Settable biases are used for initializing the inductive system for a specific domain and for adjusting the valued biases for a better performance of the induction on the specific domain, such as the domain specific attributes, relationships, components, concepts, hierarchies on attribute values, attribute domains, worth of attributes, relevancy values of components to

concepts, transformation rules, deductive rules, concept constraints and many control thresholds. BMS provides primitives and interactive facilities to define the settable biases. BMS performs the function of a knowledge acquisition system which aids the developers in transferring their domain knowledge into the inductive system.

The automatic bias-shifting method applies meta heuristics to those biases whose values or choices are difficult for a developer to decide at the initialization of an inductive system, such as the worth of an attribute, all the characteristic and discriminant attributes, important attribute relationships and component relationships, and the worth of a generalization rule. BMS, based on its previous learned knowledge, automatically adjusts those valued biases for better induction performance, or dynamically expands the vocabulary of the concept description language. Even for automatic bias shifting, users can still get some control over the process. For example, the meta bias can be selected by developers or the speed of bias shifting can be adjusted.

In BMS the fixed biases are embedded in the representations and algorithms, such as the hypothesis representation language, the data model and the induction control strategies.

The settable biases are organized into object classes according to their characteristics. Attributes, components, relationships, concepts, and rules are represented by objects. Worths or other related valued biases are represented by slots in those objects; related computational functions are methods attached to the objects. This organization is more efficient in locating an applicable heuristic rule, more economical in representing similar heuristic rules, and easier for developers to manipulate. Developers can enable or disable a heuristic rule. Developers can also modify, add, or delete a heuristic rule.

The meta heuristics are defined by the system, but developers will still be able to view them, disable them, and change the computation methods and parameters.

VITA

Min Ke was born on September 17, 1955, in Hefei, China. He attended elementary and high schools in that city. In February of 1978, he entered Anhui University in Hefei. He graduated in January 1982 with a Bachelor of Science degree in Computer Science.

In the spring of 1982, he entered Nanjing Aeronautical Institute, Nanjing, China, as a graduate student, and he was awarded a Master of Science degree in Computer Science in September of 1984. After that he worked as an instructor in the same institute for two years.

In the fall of 1986, he accepted a graduate research assistantship at the University of Tennessee Space Institute, Tullahoma, USA and began study towards a doctoral degree in Computer Science.

In August, 1991, he accepted an appointment as an assistant professor at Wayne State College, Wayne, Nebraska, USA.