Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

USING TYPE INFERENCE AND INDUCED RULES TO PROVIDE INTENSIONAL ANSWERS

Wesley W. Chu
Rei-Chi Lee

March 1990
CSD-900006

Using Type Inference and Induced Rules to Provide Intensional Answers*

Wesley W. Chu and Rei-Chi Lee

Computer Science Department
University of California, Los Angeles
Los Angeles, California 90024

ABSTRACT

An intensional answer provides characteristics rather than listing all the instances that satisfy a query. This paper presents a new approach that uses knowledge induction and type inference to provide intensional answers. Machine learning techniques are used to analyze database contents and induce a set of If-then rules. Database type hierarchies are used to derive the intensional answers to query. A prototype intensional query processing system that uses the proposed approach has been implemented. Using a ship database as a test bed, we demonstrate the use of type inference and induced rules to derive specific intensional answers.

* This research is supported by DARPA Contract F29601-87-C-0072.

Using Type Inference and Induced Rules to Provide Intensional Answerers

1. Introduction

Conventional database systems provide answers in the form of an enumeration of database instances retrieved from the database. Although such an answer conveys information to the users, a general description of the answer or summarized or approximate answers are often more useful. Meta-data of the database such as integrity constraints and semantic rules can be used to infer information hidden within the database. For example, integrity constraints were used to improve query processing performance [KING81, HAMM80] and to derive intensional answers [MOTR89].

Type hierarchies specify the subtype and supertype relationships in a database application domain and can be used to improve query processing [CHU90] and also to provide an aggregate response to queries [SHUM88]. Using type hierarchy without database intensional knowledge can only generate very limited forms of intensional answers. To remedy this problem, we propose to use knowledge induction to analyze database contents to derive a set of If-then rules. Based on the type hierarchy, these generated rules can be used to derive much more specific intensional answers. We shall refer to this process as *type inference*.

In this paper, we first present the knowledge-based entity relationship data model to represent *type/subtype* and *with constraint* information. Next, we propose a methodology that uses knowledge induction techniques to extract useful meta-data from the database. Then, we present the use of type inference to derive intensional answers. We then describe a framework and the implementation of our proposed intensional query processing system. Finally, we present examples that use type inference to derive intensional answers from an example ship database.

2. The Knowledge-based (KER) Data Model

To enhance modeling of such capabilities as type hierarchy and knowledge specification, we introduce a Knowledge-based E-R (KER) model, an extension of the Entity-Relationship Model [Chen76]. KER provides the following three generic constructs of data modeling:

1. **has/with** (*aggregation*) which links an object with another object and specifies a certain property of the object (e.g., a CLASS has an instructor);

2. **isa/with** or **contains/with** (*generalization/specialization*) which links an object type with another object type and specifies an object as a subtype of another object (e.g., PROFESSOR is-a subtype of PERSON or PERSON contains PROFESSOR, STUDENT, and STAFF);

3. **has-instance** (*classification*) which links a type to an object that is an instance of that type (e.g., "John Smith" is an instance of PROFESSOR).

Note that in addition to the semantic constructs provided by most semantic data models, KER also provides knowledge specification which is represented by the *with-constraint* information. Such knowledge specification associated with each database definition is useful for knowledge-based data processing.

In KER, an entity is a distinctly identified object, for example, a specific person, a department, or a course. An entity set is a collection of entities. Each of these entities is distinguished by a unique identifier. The set of unique identifiers is called the primary key of the entity set. A relationship specifies the connections between different entities. Conceptually, both entity type and relationship type can be considered as object type and can be modeled using the **has/with** construct. For example, Figure 1 shows an object type SUBMARINE represented in KER.

object type SUBMARINE

has key:	ShipId	**domain:**	**char[10]**
has:	ShipName	**domain:**	**char[20]**
has:	ShipType	**domain:**	**char[4]**
has:	ShipClass	**domain:**	**char[4]**
has:	Displacement	**domain:**	**integer**

with Displacement **in** [2000..30000]

Figure 1. The KER representation of an object type SUBMARINE.

The object type can also be represented mathematically as:

$$\{ [a_1, a_2, ..., a_n] \mid a_1 \in D_1, a_2 \in D_2, a_n \in D_n \text{ with } \Psi \}$$

where each tuple $[a_1, a_2, ..., a_n]$ is an instance of such a type. Note that each a_i defines an attribute of the object type, and D_i specifies its attribute domain while Ψ states constraints on the allowable values the tuple can have. An attribute domain can also be an entity type. The system provides a set of basic domains such as **integer**, **real**, **string**, and **date**. A more complex domain can be constructed from these basic domains. For example, we can define a domain AGE on the basic domain INTEGER with the range [0..200]. A BNF description of the KER model is given in the Appendix A.

A type hierarchy uses specialization/generalization constructs (**isa** or **contains** relationships) to define the subtype and supertype relationships. For example, SSBN (Ballistic Nuclear Missile Submarine) is a subtype of SUBMARINE, and CLASS-0101 is a subtype of SSBN, and therefore, a type hierarchy consisting of SUBMARINE, SSBN, and CLASS-0101 is formed (see Figure 2).

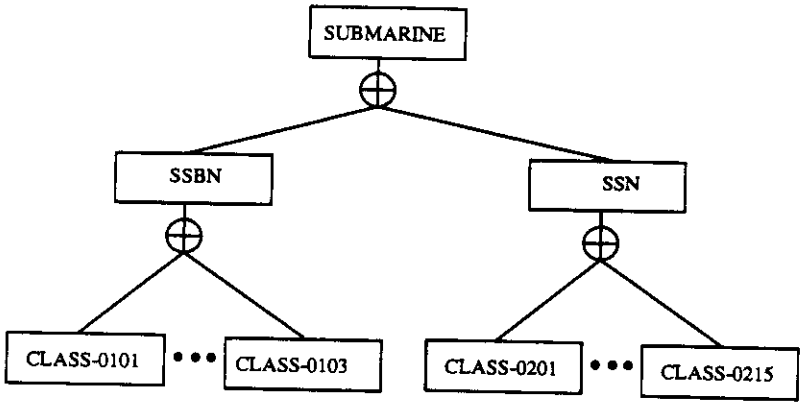

Figure 2. A Type Hierarchy SUBMARINE

A subtype inherits all the properties of its supertypes, unless some of the properties have been redefined in the subtype. For example, type SUBMARINE has attributes ShipID ,and ShipName, and type SSBN has attribute TypeID and TypeName; subtype CLASS-0101 will automatically inherit properties ShipID and ShipName from supertype SUBMARINE, and inherit properties TypeID and TypeName from another supertype SSBN.

A subtype can also be derived from another type by providing a *derivation specification.* For example, one can define a subtype SSBN (all the ships with ship type SSBN) of type SUBMARINE by specifying:

SSBN **isa** SUBMARINE **with** ShipType = "SSBN"

The with-clause defines the derivation specification of the subtype SSBN. It can also be considered as associating a constraint with this subtype.

The type hierarchy is represented in KER as:

E_1 **isa** E with Ψ_1

E_2 **isa** E with Ψ_2

...

E_n **isa** E with Ψ_n

or alternatively, it can also be represented as:

E **contains** $E_1, E_2, ..., E_n$ **with** Ψ.

This definition states that the instances of E can be divided into n disjoint subsets $E_1, E_2, ..., E_n$, with the constraint Ψ. Each E_i is a subtype of E.

To provide a graphical representation of the inter-relationships among the entity types/subtypes, relationship types, and derivation specification, we can extend the ER diagram by adding the type hierarchy with constraint representation as shown in Figure 3. A representation of a ship database schema by the KER Diagram is shown in Figure 4.

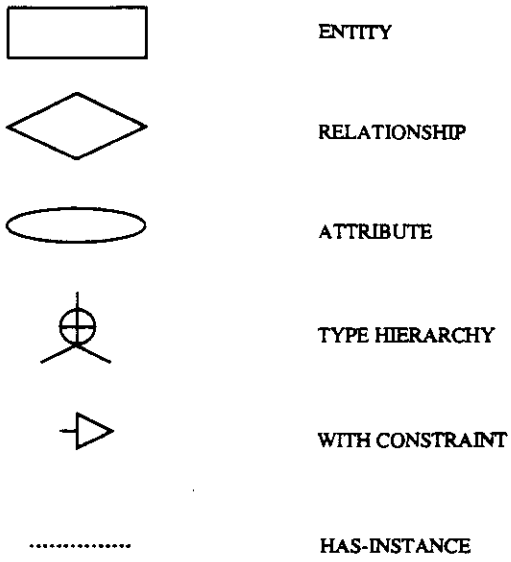

Figure 3. Components of the KER Diagram

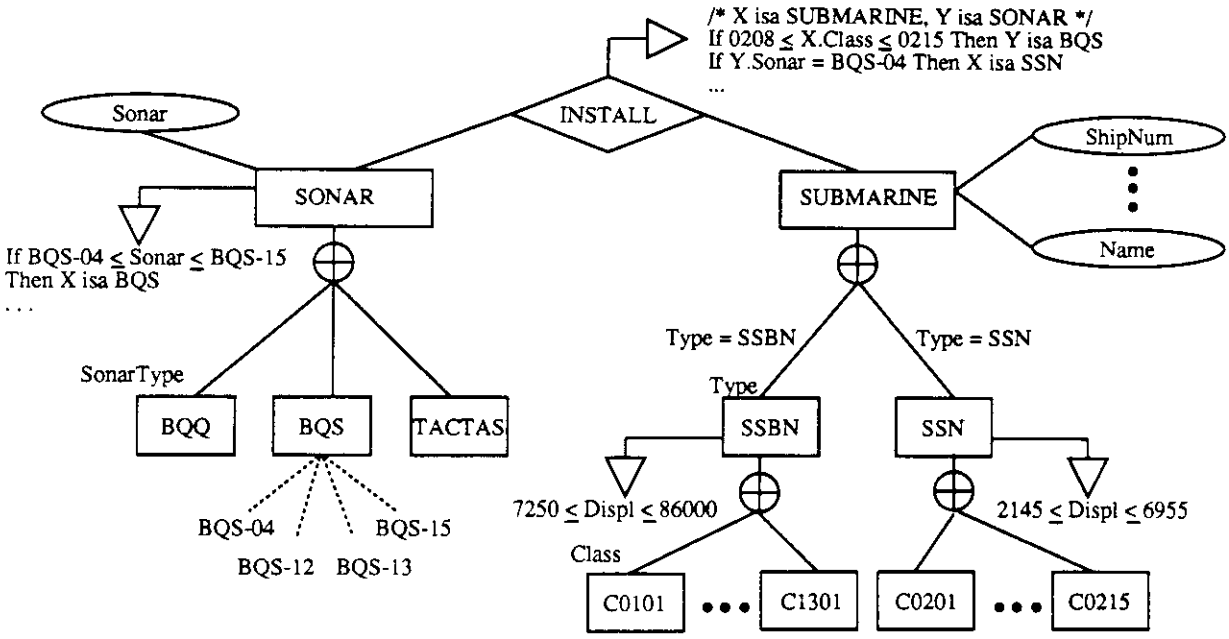

Figure 4. Representing the Ship Database Schema in KER Diagram

3. Knowledge Induction

3.1 Database Semantics

To construct the database schema, objects with similar characteristics or properties are grouped into object types and subtypes. These semantics, referred to as *classification semantics* or *classification characteristics*, are useful in knowledge-based data processing. Table 1 presents an example of the navy battleship characteristics that classify ships into ship types with different displacement ranges.

Category	Type	Type Name	Displacement (in tons)		
Subsurface	SSBN	Ballistic Nuclear Missile Submarine	7250	-	16600
	SSN	Nuclear Submarine	1720	-	6000
Surface	CVN	Attack Aircraft Carrier	75700	-	81600
	CV	Aircraft Carrier	41900	-	61000
	BB	Battleship	45000	-	45000
	CGN	Guided Nuclear Missile Crusier	7600	-	14200
	CG	Guided Missile Crusier	5670	-	13700
	CA	Gun Cruiser	17000	-	17000
	DDG	Guided Missile Destroyer	3370	-	8300
	DD	Destroyer	2425	-	7810
	FFG	Guided Missile Frigate	3605	-	3605
	FF	Frigate	2360	-	3011

Table 1. Classification Characteristics of Navy Battleships

Such characteristics, if maintained in the database, can be useful to provide intensional answers to queries. These characteristics knowledge are database semantics which are followed by instances of the database. Using these database instances as training examples, these characteristics are the candidate knowledge that can be derived from the database. To induced this knowledge, we propose to use a model-based learning methodology.

3.2 Model-based Learning Methodology

The acquisition of knowledge is one of the most difficult problems in the development of a knowledge-based system. Currently, knowledge acquisition is still largely a manual process which is very time-consuming. Further, it is often not possible for domain experts to describe their expertise to others. To remedy this problem, machine learning techniques can be used to

construct the knowledge base. Inductive learning [QUIN79, MICH83] is a machine learning technique that has been used in AI research. For a given concept and a set of training examples representing the concept, it finds a description for the concept such that all positive examples satisfy and all negative examples contradict the description. One approach is to examine the training examples to determine which descriptors are most significant in identifying the concept from other related concepts. This approach recursively determines a set of descriptors that classify each example and selects the best descriptor from a set of examples based on a statistical estimation or a theoretical information content. The set of examples is then partitioned into subsets $S_1, S_2, ..., S_n$ according to the values of the descriptor for each example. This technique is recursively applied to each S_i until each subset contains only positive examples so that the set of descriptors describes the example set. Therefore, using the database contents as the set of training examples, object classification characteristics embedded within the database can be induced using the rule induction technique. Although the automated approach speeds up the knowledge acquisition process, it has been used mainly in applications where the size of training examples is small. For a database that consists a very large volume of data, we need to identify a set of candidates for rule induction. Since a database schema is created by the designer based on the semantic characteristics of the application, such semantic characteristics can be used as the candidates for rule induction. Therefore, we propose to use machine learning to acquire database characteristics and use the database schema to guide the rule induction process.

The Knowledge-based Entity Relationship (KER) model will be used to facilitate rule induction. The KER model consists of entity sets and relationship sets. The semantic knowledge associated with each database are: *intra-object knowledge* and *inter-object knowledge*. Intra-object knowledge defines specific properties of each entity set such as the attribute domains, value ranges, relationships between attributes, etc., and restricts the allowable instances of an entity set. For example, the displacement of an Attack Aircraft Carrier is in the range of 75,700 tons - 81,600 tons.

The inter-object knowledge specifies the constraints that the instances of a relationship set must satisfy. For example, the relationship VISIT involves entities of SHIP and PORT and satisfies the constraint that the draft of the ship must be less than the depth of the port. The inter-object knowledge can be induced from the interrelationship between SHIP and PORT linked by the VISIT relationship.

4. Deriving Intensional Answers by Type Inference

A relational database is made up of the extension database (EDB) and the intension database (IDB) [GALL78, NICO78]. The EDB is the set of tuples contained in the relations. It is expressed in *relations* over domain values. The IDB is the set of general laws (i.e., meta-data) about data stored in the EDB. It is expressed in closed well-formed formulas in the first-order predicate calculus.

An answer to a query is the set of data values that satisfy a qualification specified in the query. Generally, query answers are retrieved from the EDB. An *intensional answer* to a query provides the characterizations of the set of data values that satisfies the query [MOTR89]. In many applications, users are satisfied with or prefer to obtain summarized answers rather than the answers from the EDB. Such summarized or abstract answers can be represented as intensional answers. In the following, we shall show that the rules that we induced from the database contents can be used for deriving intensional answers.

The intra- and inter-object knowledge specifies the inter-relationships between the database objects which are the essential components of the intensional database. This knowledge can be induced by our model-based knowledge acquisition methodology. Using these induced rules and based on the database schema, the condition, and object types specified in the query, the inference processor derives the intensional answers by traversing the type hierarchies of the object types as specified in the query. We call this technique *type inference*. For example, the entities SUBMARINE, SSN (Nuclear Missile Submarines), and SSBN (Ballistic Nuclear Missile Submarines) forms a type hierarchy where the set of SUBMARINES can be divided into two

disjoint subsets: SSBN and SSN. Representing this type hierarchy associated with the induced rules in KER, we have the intensional knowledge as shown in Figure 5.

```
SSBN isa SUBMARINE with ShipType = "SSBN"
SSN isa SUBMARINE with ShipType = "SSN"

object type SUBMARINE

    has key: ShipId        domain: char[20]
    ...
    has:     Displacement  domain: integer

    with /* x isa SUBMARINE */

        if x.Displacement ≥ 7250 then x isa SSBN
        if x.Displacement ≤ 6955 then x isa SSN
```

Figure 5. A Type Hierarchy with Induced Rules for Submarine.

This knowledge can be used to provide intensional answers to queries that involve SUBMARINE or SSBN type ships.

Intensional answers can be derived by *forward inference* (Modus Ponens) and *backward inference*. Forward inference uses the known facts to derive more facts, i.e., given a rule "if X then Y", and a fact "X is true", we can conclude "Y" is true. Using forward inference, we can traverse the type hierarchies of the object types specified in the query based on the query condition and the *with* constraints to derive intensional answers. For example, consider a query asking the submarines with displacement greater than 8,000. Using the intensional knowledge of Figure 5, We can traverse down from the submarine hierarchy (Figure 4) to derive an intensional answer "SSBN" since the condition "Displacement > 8000" is subsumed by "Displacement ≥ 7250".

Backward inference uses the known facts to infer what must be true according to the induced rules. For example, given a rule "if x *isa* SUBMARINE and x.Displacement ≥ 7250, then x *isa* SSBN", and a fact "x *isa* SSBN", we can conclude that "x.Displacement ≥ 7250" must be true, otherwise if "x.Displacement" is not true and "x *isa* SSBN" is true, then we will not have

such induced rule in the knowledge base. The backward inference described here is different from the backward chaining in logic programming such as PROLOG which uses backward reasoning to prove goals. Using backward inference, we can only derive descriptions of a *subset* of the extensional answers. For example, there might have some SSBN ships with displacements less than 7250.

Using forward inference, the intensional answer gives a description of a set of instances that includes the answers. Therefore, the intensional answers derived from forward inference characterize a set of instances *containing* the extensional answer. Using backward inference, the intensional answer gives only a description of partial answers. There may be other extensional answers that satisfy the query condition but are not included in the intensional answer derived from the backward inference. Therefore, the intensional answer derived from backward inference characterizes a set of answers *contained in* the extensional answer. The forward and backward type inference can be combined to derive a more specific intensional answer.

5. Intensional Query Processing System

5.1 System Architecture

Let us now describe an intensional query processing system which consists of three components: a traditional query processor, an intelligent data dictionary, an inductive learning subsystem, and an inference processor as shown in Figure 6.

The intelligent data dictionary is a knowledge base containing meta-data which includes database schema and semantic knowledge. The database schema describes the interrelationships among database objects in terms of entities and relationships as specified in the KER model. Semantic knowledge is the semantic characteristics of the database objects such as domain ranges, induced semantic rules, etc. Using the rule induction techniques, the inductive learning subsystem induces semantic rules by analyzing database schema and contents. Based on the database knowledge stored in the intelligent data dictionary, the inference processor

derives intensional answers from the given query.

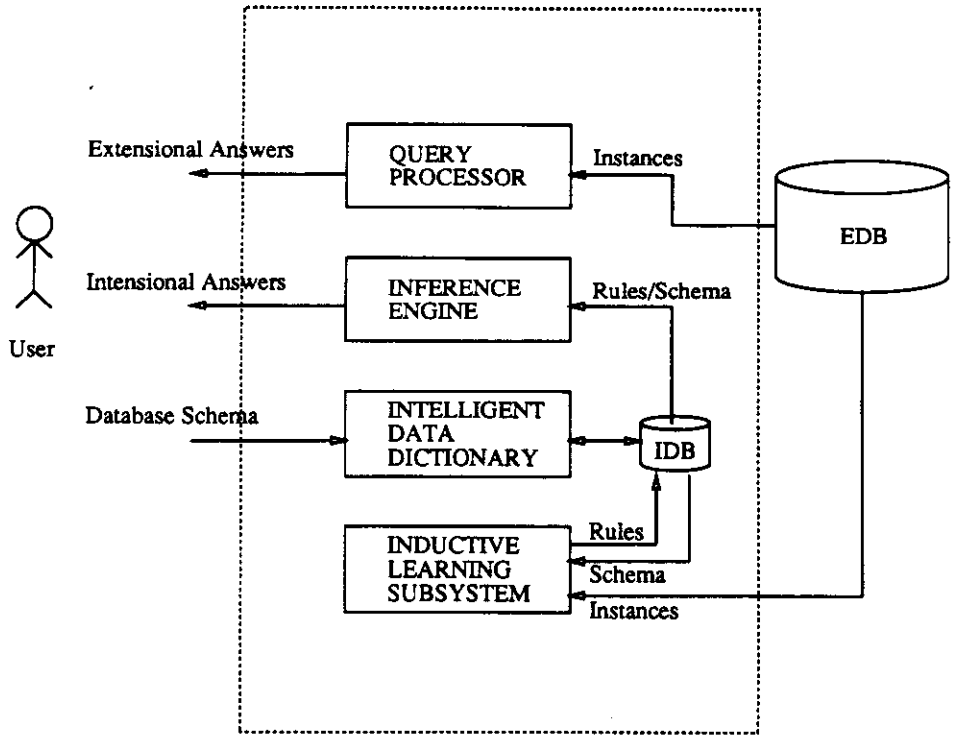

Figure 6. An Intensional Query Processing System

5.2 The Inductive Learning Subsystem (ILS)

Inductive learning involves determining a set of characterizations to classify objects into similar classes. In this section, we describe our Model-based Inductive Learning Subsystem (ILS), detailing the use of database schema to form a classification. The inductive learning approach uses database schema information with the following inputs:

- object instances,

- schema describing object types and object hierarchy, and

- criteria to evaluate the quality of a classification.

The output is the characterization of classes. In our implementation, the set of object instances is represented as relations and the database schema is represented in the KER model. The ILS uses the object hierarchy to generate the classification characteristics for each class.

5.2.1 The Rule Induction Algorithm

We have implemented a prototype of the proposed intensional query processing system in EQUEL (Embedded QUEL) and C on top of the INGRES system. Rule induction is performed in the ILS which uses the relational operations to generate semantic rules for pairwise attributes. Let us now present the algorithm that induces correlated relationships of the rule scheme X --> Y for attribute pair (X, Y).

Rule Induction Algorithm.

1. *Retrieving (X, Y) value pairs*

   Retrieve into S the instances of the (X, Y) pair from the database. The corresponding QUEL statement is:

   ```
   range of r is relation
   retrieve into S unique (r.Y, r.X)
   sort by r.Y
   ```

2. *Removing inconsistent (X, Y) value pairs*

   Retrieve all the (X, Y) pairs that for the same value of X has multiple values of Y. Let T be the result of this relation.

   ```
   range of r is relation
   range of s is S
   retrieve into T unique (s.Y, s.X)
   where (r.X = s.X and r.Y != s.Y)
   ```

   Then, remove all the (X, Y) pairs that have different Y values for the same X value from S.

range of s is S
range of t is T
delete s
where (s.X = t.X and s.Y = t.Y)

3. *Constructing Rules*

For each distinct value of Y in S, say y, determine the *value range* x of X and create a rule in the form of

if $x_1 \leq X \leq x_2$ **then** $Y = y$.

A value range is defined as a consecutive sequence of X values that occur in the database. The rules generated for the the same attribute pair (X, Y) consist of the *rule set* designated by the rule scheme X --> Y. Note that when $x_1 = x_2$, then the rule reduces to

if $X = x$ **then** $Y = y$.

4. *Pruning the Rule Set*

Although storing more rules in the knowledge base provides more opportunities for inference, it also increases the overhead for storing and searching these rules. Therefore, when the number of rules generated becomes too large, the system must reduce the size of rule set. In general, we keep the rules that are satisfied by many database instances and drop those rules that are satisfied by only a few database instances. We remove these rules from the knowledge base when the number of instances satisfied is less than a prespecified number N_c, which can be a percentage of the total number of instances of a relation. N_c provides a tradeoff between the applicability of the rules and the overhead of storing and searching these rules for providing intensional answers.

5.2.2 Rule Relations

Since knowledge is induced from the database, it is necessary that knowledge be vound to the data in some way. Therefore, in our implementation, rules are represented in relations referred to as *rule relations*. A database and its associated rule relations can be relocated together. When the database is used in a location, the associated schema and rules are loaded into the system. The rule relations are then converted into the KER representation and stored in the intelligent data dictionary. Rule relations are added to the database as meta-relations. In our representation, each rule consists of the *Left-Hand-Side* (LHS, also called *premise*) and the *Right-Hand-Side* (RHS, also called *consequence*) as follows:

LHS --> RHS.

or

if LHS **then** RHS

Each portion is represented as a *conjunctive* form which contains one or more *clauses* as:

if C_{L_1} **and** ... **and** C_{L_n} **then** C_R

where each C's is a clause. In our implementation, we only deal with Horn Clause, that is, the RHS portion contains at most one clause. However, the LHS portion can contain many clauses. Each clause defines an attribute value range and is represented as an expression in the form:

(*lvalue*, *attribute*, *uvalue*)

where *lvalue* is the lower value limit (inclusive) and *uvalue* is the upper value limit (inclusive) of the attribute value range which is equivalent to "*lvalue* ≤ *attribute* ≤ *uvalue*". For example,

(18, Employee.Age, 65)

means that the Employee.Age is in the range 18 to 65; and

("ENGINEER", Employee.Position, "ENGINEER")

means that the Employee.Position is equal to "ENGINEER". To store the clauses in relation, both *lvalue*, *rvalue*, and the *attribute* are encoded as integers and a mapping between the encoded numbers and the real values are provided in an attribute value mapping relation and a system table provided by INGRES.

Therefore, each rule is represented as a set of clauses $\{C_R, C_{L_1}, ..., C_{L_n}\}$. The relational schema of R' is defined as:

$$R' = (RuleNo, Role, Lvalue, AttributeNo, Uvalue).$$

where *RuleNo* is an index to the rule relation. Each rule has a unique rule number. *Role* indicates whether the clause is in LHS (L) or RHS (R). *Lvalue* and *Uvalue* are the lower value limit and upper value limit of the attribute value range of the clause.

For example, given the following rule:

if $a1 \leq R.A \leq a2$ **then** $R.B = b1$

where R is a relation and A and B are two attributes of R. The rule relation for this rule is

RuleNo	Role	Lvalue	Att_no	Uvalue
1	L	1.00	0	2.00
1	R	1.00	1	1.00

and the attribute value mapping relation is

Att_no	Value	RealValue
0	1.00	a1
0	2.00	a2
1	1.00	b1

Using the rule relation representation, the knowledge can be relocated to different locations with the database. The knowledge is stored in the extended data dictionary and used by the inference engine to derive intensional answers.

5.3 The Extended Data Dictionary

The extended data dictionary is a knowledge-based data dictionary which includes database schema and semantic knowledge represented in KER. The knowledge representation combines both frame-based and rule-based knowledge representation. The database schema, describing the inter-relationship between database objects, is represented by frame-based knowledge representation. Each object type is represented as a frame and the object hierarchy is represented as a hierarchy of frames. The semantic rules are translated into rule-based knowledge representation for inference.

6. Intensional Answers Derived From a Ship Test Bed

We shall now use type inference to derive intensional answers. The ship database was created by the System Development Corporation (now UNISYS) to provide a generic naval database based on [JANE81]. The database is currently running on INGRES on a Sun 3/60 machine. These examples use the nuclear submarine portion of the database which consists the following relations (a sample database instance is given in the Appendix C):

SUBMARINE = (*Id*, *Name*, *Class*)

CLASS = (*Class*, *ClassName*, *Type*, *Displacement*)

TYPE = (*Type*, *TypeName*)

SONAR = (*Sonar*, *SonarType*)

INSTALL = (*Ship*, *Sonar*)

The database consists of five entity types: SUBMARINE, CLASS, TYPE, SONAR, SONAR_TYPE and one relationship type: INSTALL. The three entity types SUBMARINE, TYPE, and CLASS form a ship hierarchy and the entities SONAR and SONAR TYPE form another hierarchy as shown in Figure 4. Each submarine type contains a set of submarine classes and each submarine class contains a set of submarine instances. For example, Submarines are divided into two types: SSBN (Ballistic Nuclear Missile Submarine) and SSN (Nuclear Subma-

rine). The SSBN ships contain three classes of ships: 0101 (Ohio), 0102 (Benjamin Franklin), and 0103 (Lafayette), and there are three ships that belong to the ship class 0103 (Lafayette).

Each ship class has its specific characteristics such as displacement, length, beam, etc. For tactical or strategic reasons, different sonars are installed on different ships. The relationship INSTALL indicates the sonars installed on the different ships. A textual representation of the database schema is given in Appendix B.

Applying our knowledge acquisition technique to the ship database generates 17 rules as shown below (rules are grouped by object types):

(1) SUBMARINE

R_1: **if** SSN623 ≤ *Id* ≤ SSN635 **then** x **isa** C0103
R_2: **if** SSN648 ≤ *Id* ≤ SSN666 **then** x **isa** C0204
R_3: **if** SSN673 ≤ *Id* ≤ SSN686 **then** x **isa** C0204
R_4: **if** SSN692 ≤ *Id* ≤ SSN704 **then** x **isa** C0201

(2) CLASS

R_5: **if** 0101 ≤ *Class* ≤ 0103 **then** x **isa** SSBN
R_6: **if** 0201 ≤ *Class* ≤ 0215 **then** x **isa** SSN
R_7: **if** Skate ≤ *ClassName* ≤ Thresher **then** x **isa** SSN
R_8: **if** 2145 ≤ *Displacement* ≤ 6955 **then** x **isa** SSN
R_9: **if** 7250 ≤ *Displacement* ≤ 30000 **then** x **isa** SSBN

(3) SONAR

R_{10}: **if** BQQ-2 ≤ *Sonar* ≤ BQQ-8 **then** x **isa** BQQ
R_{11}: **if** BQS-04 ≤ *Sonar* ≤ BQS-15 **then** x **isa** BQS

(4) INSTALL (x isa SUBMARINE and y isa SONAR)

R_{12}: **if** SSN582 ≤ x.*Id* = SSN601 **then** y **isa** BQS
R_{13}: **if** SSN604 ≤ x.*Id* = SSN671 **then** y **isa** BQQ
R_{14}: **if** x.*Class* = 0203 **then** y **isa** BQQ
R_{15}: **if** 0205 ≤ x.*Class* ≤ 0207 **then** y **isa** BQQ
R_{16}: **if** 0208 ≤ x.*Class* ≤ 0215 **then** y **isa** BQS
R_{17}: **if** y.*Sonar* = BQS-04 **then** x **isa** SSN

These induced rules are used by the inference engine to derive intensional answers.

Let us consider the following examples:

Example 1: Find the Ids, Names, Classes, and Types of the SUBMARINE with Displacement greater than 8000.

```
SELECT      SUBMARINE.ID, SUBMARINE.NAME
            SUBMARINE.CLASS, CLASS.TYPE
FROM        SUBMARINE, CLASS
WHERE       SUBMARINE.CLASS = CLASS.CLASS
AND         CLASS.DISPLACEMENT > 8000
```

The extensional answer of the above query is:

id	name	class	type
SSBN730	Rhode Island	0101	SSBN
SSBN130	Typhoon	1301	SSBN

Using forward inference with the induced rule R_9 and the definition of SSBN in the database schema, we derive the following intensional answer:

A_I = "*Ship type SSBN has displacement greater than 8000*"

which provides a summarized answer for the query.

Example 2: Find the names and classes of the SSBN ships.

```
SELECT  SUBMARINE.NAME, SUBMARINE.CLASS
FROM    SUBMARINE, CLASS
WHERE   SUBMARINE.CLASS = CLASS.CLASS
AND     CLASS.TYPE = "SSBN"
```

The following is the extensional answer to the above query:

name	class
Nathaniel Hale	0103
Daniel Boone	0103
Sam Rayburn	0103
Lewis and Clark	0102
Mariano G. Vallejo	0102
Rhode Island	0101
Typhoon	1301

Using backward inference with the induced rule R_5, the following intensional answer can be derived for this query:

A_I = "*Ship Classes in the range of 0101 to 0103 are SSBN.*"

Note that ship class 1301 is also a SSBN (see Appendix C), but is not included in the answer. This is because backward inference is used to derive the intensional answer which yields only a partial answer. As a result, the answer is incomplete. Note the following rule

R_{new}: **if** x.*Class* = 1301 **then** x **isa** SSBN.

is satisfied only by a single instance. For efficiency reasons, R_{new} is not maintained in the knowledge base. However, if this rule is maintained by the system, then the derived intensional answer will be complete.

Example 3: List the names, classes and types of SUBMARINEs equipped with sonar BQS-04.

```
SELECT    SUBMARINE.NAME, SUBMARINE.CLASS, CLASS.TYPE
FROM      SUBMARINE, CLASS, INSTALL
WHERE     SUBMARINE.CLASS = CLASS.CLASS
AND       SUBMARINE.ID = INSTALL.SHIP
AND       INSTALL.SONAR = "BQS-04"
```

The extensional answer of the above query is:

name	class	type
Bonefish	0215	SSN
Seadragon	0212	SSN
Snook	0209	SSN
Robert E. Lee	0208	SSN

Using forward inference, from rule R_{17}, we know the ship type must be SSN; and from rule R_{11}, we know the sonar type is BQS. Next, using backward inference with the rule R_{16}, we conclude that the answers must contain ships with class from 0208 to 0215 (See Figure 4). We therefore have the following intensional answer:

A_I = "*Ship type SSN with class 0208 to 0215 is equipped with sonar BQS-04.*"

In this example, we combine forward and backward inferences to derive the specific intensional answer from two object types (SUBMARINE and SONAR) that are related by the INSTALL relation.

7. Conclusions

In this paper, we present an approach using type inference and induced rules to provide intensional answers to queries. An inductive learning technique is developed to induce knowledge from the database contents. Using the induced knowledge, inference can be performed on the type hierarchies to derive intensional answers for queries. An architectural framework of an intensional query processing system consists of an intelligent data dictionary, an inductive learning system, and an inference engine is presented.

A prototype system has been implemented on top of INGRES using a naval ship database as a test bed. A machine learning technique is used to acquire the rules from database contents. These rules are stored in rule relations. Forward and backward type inferences can be used individually or combined to derive intensional answers.

Our experiments reveal that induced rules can play an important role in type inference in providing intensional answers. Further, type inference with induced rules is a more effective technique to derive intensional answers than using integrity constraints when the database schema has strong type hierarchy and semantic knowledge.

REFERENCES

[BROD84a] Brodie, M., Mylopoulos, J., and Schmidt, J. W., (eds.) *On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Springer, New York, 1984.

[CHEN76] Chen, P.P.S., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactio on Database Systems*, Vo. 1, No. 1, March 1976.

[CHU90] Chu, W. W., Lee, R., "Semantic Query Optimization via Database Restructuring," to appear in 8th International Congress of Cybernetics and Systems, New York, 1990.

[HAMM75] Hammer, M., and Mcleod, D., "Semantic integrity in a relational data base system," In *Proceedings of the First International Conference on Very Large Data Bases*, IEEE, New York, pp. 25-47, 1975.

[HAMM80] Hammer, M. and Zdonik, S. B., Jr., "Knowledge-based query processing," In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1-3). IEEE, New York, pp. 137-147, 1980.

[HAMM81] Hammer, M., and McLeod, D., "Database Description with SDM: A Semantic Database Model," *ACM Transactions on Database Systems*, Vol. 6, No. 3, September 1981.

[JANE81] "*Jane's Fighting Ships*", Jane's Publishing Co., 1981.

[KING81] King, J. J., "QUIST: A system for semantic query optimization in relational databases," In *proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, Sept. 9-11). IEEE, New York, pp. 510-517.

[MCKE82] McLeod, D., and Smith, J. M., "Abstraction in Database," *Proc. Workshop on Data Abstraction, Databases, and Conceptual Modelling, SIGMOD Record*, Vol. 11, No. 2, February 1981.

[MICH83] Michalski, R. S., et al, (eds.) *Machine Learning: An Artificial Intelligence Approach*, Tioga Press, Palo Alto, 1983.

[MOTR89] Motro, A., "Using Integrity Constraints to Provide Intensional Answers to Relational Queries," In *Proc. Fifteenth Intl. Conf. on Very Large Data Bases*, August 1989.

[QUIN79] Quinlan, J. R., "Induction Over Large Data Bases", STAN-CS-79-739, Stanford University, 1979.

[SHUM88] Shum, C. D., and R. Muntz, "An Information-Theoretic Study on Aggregate Responses," In *Proc. Fourteenth Intl. Conf. on Very Large Data Bases*, August 1988.

Appendix A. A BNF definition of the KER Model

We will use the following BNF conventions:

<...> non-terminal symbol

{ x } x appears 0 or more times

[x] x appears 0 or 1 time

x1 | x2 | ... | xn x1 or x2 or ... or xn

'1' literal symbol

A.1 Data Definition Statements

<KER definition> ::=

<domain definitions> |

<object type definitions> |

<type hierarchy definitions>

A.2 Domain Definition Statements

<domain definitions> ::=

<domain definition> {, <domain definition> }

<domain definition> ::=

domain <domain name> **is** <domain description>

[<domain sepcification>]

<domain name> ::= **identifier**

<domain description> ::= <standard domain> | <object domain>

<standard domain> ::= **string** | **integer** | **real** | **date**

<domain specification> ::=

<range specification> |

<set specification>

<range specification> ::=

range <lower boundary> <value> ',' <value> <upper boundary>

<lower boundary> ::= '[' | '('

<upper boundary> ::= ']' | ')'

<set specification> ::=

set of '{' <value> {, <value> } '}'

<value> ::= **identifier** | <integer> | <real>

<object domain> ::= <object type name>

<object type name> ::= **identifier**

A.3 Object Type Definition Statements

<object type definition> ::=

object type <object type name>

<attribute list>

<with constraints>

<attribute list> ::=

<attribute> {, <attribute> }

<attribute> ::=

has [**key**] ':' <attribute name> **domain** <domain name>

<with constraints> ::=

with <constraints>

A.4 Type Hierarchy Definition Statements

<type hierarchy definition> ::=

<object type name> **contains** <sub-type list>

[<attribute list>]

[<with constraints>]

<sub-type list> ::=

<object type name> {, <object type name> }

A.5 Constraint Definition Statements

<constraints> ::=

<constraint> {, <constraint> }

<constraint> ::=

<domain range constraint> |

<semantic rule>

<domain range constraint> ::=

<attribute name> **in** <domain sepcification>

<semantic rule> ::=

<constraint rule> |

<structure rule>

<constraint rule> ::=

if <premise> **then** <consequence>

<premise> ::=

<conjuntives>

<conjunctives> ::=

<clause> { **and** <clause> }

<clause> ::=

<attribute> <operator> **constant**

<consequence> ::=

<attribute> '=' **constant**

<structure rule> ::=

if <role definitions>

and <conjunctives>

then <variable> **isa** <object type name>

<role definitions> :: =

<role> { **and** <role> }

<role> ::= <variable> **isa** <object type name>

<variable> ::= **identifier**

Appendix B. A KER Representation of a Naval Ship Database Schema.

B.1 Domain Definitions

domain: NAME **isa** CHAR[20]

domain: CLASS_NAME **isa** NAME
domain: SHIP_NAME **isa** NAME

domain: TYPE_NAME **isa** CHAR[30]

domain: SONAR_NAME **isa** CHAR[8]

B.2 Object Type Definitions

object type CLASS

has key:	Class	**domain**: CHAR[4]
has:	Type	**domain**: type
has:	ClassName	**domain**: CLASS_NAME
has:	Displacement	**domain**: INTEGER

with /* constraint rules */

if "0101" ≤ Class ≤ "0103" **then** Type = "SSBN"
if "0201" ≤ Class ≤ "0216" **then** Type = "SSN"

CLASS **contains** SSBN, SSNs

Bwith /* x **isa** CLASS */

if 2145 ≤ x.Displacement ≤ 6955 **then** x **isa** SSN
if 7250 ≤ x.Displacement ≤ 30000 **then** x **isa** SSBN

object type SUBMARINE

has key:	Id	**domain**: CHAR[7]
has:	Name	**domain**: SHIP_NAME
has:	Class	**domain**: class

SUBMARINE **contains** C0101, ..., C1301

object type TYPE

has key:	Type	**domain**: CHAR[4]
has:	TypeName	**domain**: TYPE_NAME

object type SONAR

has key:	Sonar	**domain**: CHAR[8]
has:	SonarType	**domain**: SONAR-NAME

SONAR **contains** BQQ, BQS, TACTAS

with /* x **isa** SONAR */

if BQQ-2 ≤ x.Sonar ≤ BQQ-8 **then** x **isa** BQQ
if BQS-04 ≤ x.Sonar ≤ BQS-15 **then** x **isa** BQS
if x.Sonar = "TACTAS" **then** x **isa** TACTAS

object type INSTALL

has key:	Ship	**domain:** SUBMARINE
has:	Sonar	**domain:** SONAR

with /* x **isa** SUBMARINE and y **isa** SONAR */

if x.Class = 0203 **then** y **isa** BQQ
if 0205 ≤ x.Class ≤ 0207 **then** y **isa** BQQ
if 0208 ≤ x.Class ≤ 0215 **then** y **isa** BQS
if y.Sonar = BQS-04 **then** x **isa** SSN

Appendix C. A Ship Database and Its Induced Rules

A Ship Database:

Relation SUBMARINE

Id	Name	Class
SSBN130	Typhoon	1301
SSBN623	Nathaniel Hale	0103
SSBN629	Daniel Boone	0103
SSBN635	Sam Rayburn	0103
SSBN644	Lewis and Clark	0102
SSBN658	Mariano G. Vallejo	0102
SSBN730	Rhode Island	0101
SSN582	Bonefish	0215
SSN584	Seadragon	0212
SSN592	Snook	0209
SSN601	Robert E. Lee	0208
SSN604	Haddo	0205
SSN610	Thomas A. Edison	0207
SSN614	Greenling	0205
SSN648	Aspro	0204
SSN660	Sand Lance	0204
SSN666	Hawkbill	0204
SSN671	Narwhal	0203
SSN673	Flying Fish	0204
SSN679	Silversides	0204
SSN686	L. Mendel Rivers	0204
SSN692	Omaha	0201
SSN698	Bremerton	0201
SSN704	Baltimore	0201

Relation INSTALL

Ship	Sonar
SSBN130	BQQ-2
SSBN623	BQQ-5
SSBN629	BQQ-5
SSBN635	BQS-12
SSBN644	BQQ-5
SSBN658	BQS-12
SSBN730	BQQ-5
SSN582	BQS-04
SSN584	BQS-04
SSN592	BQS-04
SSN601	BQS-04
SSN604	BQQ-2
SSN610	BQQ-5
SSN614	BQQ-2
SSN648	BQQ-2
SSN660	BQQ-5
SSN666	BQQ-8
SSN671	BQQ-2
SSN673	BQS-12
SSN679	BQS-13
SSN686	BQQ-2
SSN692	BQS-15
SSN698	TACTAS
SSN704	BQQ-5

Relation TYPE

Type	TypeName
SSBN	ballistic nuclear missile sub
SSN	nuclear submarine

Relation SONAR

Sonar	SonarType
BQQ-2	BQQ
BQQ-5	BQQ
BQQ-8	BQQ
BQS-04	BQS
BQS-12	BQS
BQS-13	BQS
BQS-15	BQS
TACTAS	TACTAS

Relation CLASS

Class	ClassName	Type	Displacement
0101	Ohio	SSBN	16600
0102	Benjamin Franklin	SSBN	7250
0103	Lafayette	SSBN	7250
0201	LosAngeles	SSN	6000
0203	Narwhal	SSN	4450
0204	Sturgeon	SSN	3640
0205	Thresher	SSN	3750
0207	Ethan Allen	SSN	6955
0208	George Washington	SSN	6019
0209	Skipjack	SSN	3075
0212	Skate	SSN	2360
0215	Barbel	SSN	2145
1301	Typhoon	SSBN	30000