Lattice-structured Domains, Imperfect Data and Inductive Queries

Sally Rice[1] and John F. Roddick[2]

[1] School of Computer and Information Science, The Levels Campus,
University of South Australia, Mawson Lakes 5095, South Australia.
rice@cis.unisa.edu.au
[2] School of Informatics and Engineering, Flinders University of South Australia,
GPO Box 2100, Adelaide 5001, South Australia.
roddick@cs.flinders.edu.au

Abstract. The relational model, as proposed by Codd, contained the concept of relations as tables composed of tuples of single valued attributes taken from a domain. In most of the early literature this domain was assumed to consist of elementary items such as simple (atomic) values, defined complex data types or arbitrary length binary objects. Subsequent to that the nested relational or non-first normal form model allowing set-valued or relation-valued attributes was proposed. Within this model an attribute could take multiple values or complete relations as values. This paper presents a further extension to the relational model which allows domains to be defined as a hierarchy (specifically a lattice) of concepts, shows how different types of imperfect knowledge can be represented in attributes defined over such domains, and demonstrates how lattices allow the accommodation of some forms of inductive queries. While our model is applied to flat relations, many of the results given are applicable also to nested relations. Necessary extensions to the relational algebra and SQL, a justification for the extension in terms of application areas and future research areas are also discussed.

1 Introduction

Attribute values in relational databases which are not numeric, spatial or temporal are commonly interpreted as labels with no semantic connection between them beyond that of simple ordering. For example, consider the attribute domain {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, weekday, weekend}. As well as the order of the days of the week, there is also an inclusion relationship between Sunday and weekend (since Sunday is part of the weekend). This type of relationship can be expressed in a concept lattice, which can be represented in the database as a domain concept relation – a simple child-parent binary relation between pairs of labels from the domain [10, 11]. Kedad and Métais [4] have looked at hierarchical domains, but their approach is limited to trees.

Mannila [5] defines inductive databases as databases that contain inductive generalisations about data. The inductive queries discussed in this paper, although relevant to these inductive generalisations (for example, see [9–11]), are not restricted to inductive databases, but can be used in any database which has the ability to represent hierarchical domains and store imperfect data.

There are many terms used in the literature for different types of imperfect information. Parsons [6] offers a classification of imperfect information into five categories: uncertainty (lack of information), imprecision (lack of granularity), vagueness (the fuzziness implied by terms such as 'young' or 'short'), incompleteness (lack of relevant information) and inconsistency (contradictory information). We wish to consider three cases of imperfection which we present in detail in the next section of this paper. The type of imperfect information captured by the use of weekend instead of Sunday we term *incomplete*. Our first example using a lattice built on geographic locations gives rise to imperfect information of this type. Our second example uses a lattice of temporal intervals, with intervals at a higher level of the lattice containing those at lower levels. This type of information we term *imprecise*. Our third example is of *inconsistent* information, where one piece of data contradicts another. The term uncertainty in this paper is reserved for describing the effect of matching domain values at different levels of the hierarchy with the attribute value in an inductive query. Parson's uncertainty and vagueness are not addressed explicitly, but vagueness does not differ significantly from the incomplete information example, and uncertainty could be modelled using inconsistency if information with 70% certainty was interpreted as weights of 0.7 on the information and 0.3 on its negation.

This paper explores the use of lattices for describing the three examples, then introduces *inductive queries* (queries on attributes defined on lattice-structured domains) and discusses the uncertainty inherent in these queries. Necessary functions that operate on lattices, such as H and L [9] which retrieve all the concepts in the lattice above and below a given concept, are defined and an algorithm which uses these functions to divide the elements in the lattice into three sets corresponding to the truth values t (true), f (false) and u (uncertain) for a given inductive query is given. The implications of inductive queries for relational algebra operations is investigated, and some extensions to the algebra are proposed. These extensions, and other changes to the definition and data update facilities of query languages for inductive queries are discussed within the framework of an implemented query language (SQL).

The paper closes with a discussion of application areas and future research for this work.

2 Lattice-structured Domains

The structure of any hierarchical domain can be described by a domain concept relation DS = (element, concept), where each tuple describes a relation between two elements of the domain – the first element belongs to the concept that the second element describes. Consider a domain D with the twelve ele-

ments {Americas, North America, Central America, South America, USA, Canada, Mexico, Nicaragua, Brazil, Chile, northern hemisphere, southern hemisphere}. Figure 1 (a) shows a tree-structured domain built from ten of them, describing the geographical relationship between the elements. Each element at the lower levels of the structure has a single parent at the previous level, leading to unique paths from any node to the root of the structure to which it belongs.

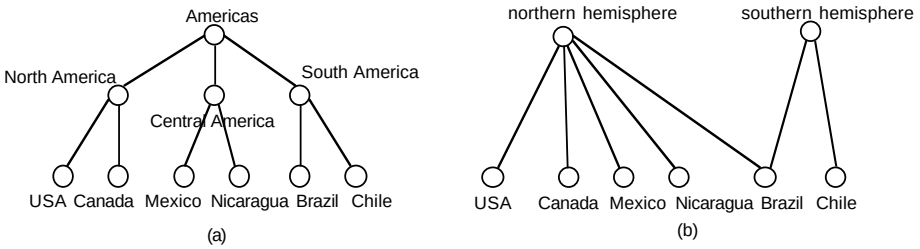

Fig. 1. (**a**) Tree-structured domain and (**b**) lattice-structured domain

Allowing children to have multiple parents in a structure changes the structure from a tree to a lattice: a simple example is that shown in Fig. 1 (b), where one child, Brazil, has two parents. Table 1 shows a domain concept relation combining the two hierarchical structures shown in Fig. 1. Top level concepts (Americas, northern hemisphere and southern hemisphere) are identified by never appearing in the element column, base level concepts (the names of the individual countries) by never appearing in the concept column. Middle level concepts appear in both columns.

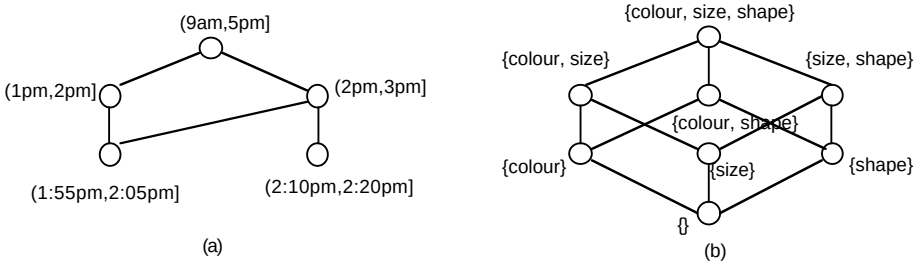

Fig. 2. (**a**)Lattice of intervals and (**b**) lattice of attribute sets

This example shows that information at different levels of granularity can be combined into a single lattice-structured domain, allowing the use of incomplete (i.e. coarser grained) information in relations defined on this domain. Lattices can also be used for imprecise information represented as intervals, by labelling

Table 1. Domain concept relation

element	concept
USA	North America
Canada	North America
Mexico	Central America
Nicaragua	Central America
Brazil	South America
Chile	South America
North America	Americas
Central America	Americas
South America	Americas
USA	northern hemisphere
Canada	northern hemisphere
Mexico	northern hemisphere
Nicaragua	northern hemisphere
Brazil	northern hemisphere
Brazil	southern hemisphere
Chile	southern hemisphere

each node in the lattice with an interval as shown in Fig. 2 (a). The idea of representing interval data in lattices is not new, see [2] for example. Similarly, a lattice whose nodes are sets of attributes can be defined for any relation and used to identify inconsistent information in tuples of the relation. If the same object is described as both red, small and square, and blue, small and square, we can represent this inconsistency by the node marked {colour} in Fig. 2 (b).

3 Uncertainty in Inductive Queries

For the domain shown in Fig. 1 (b), the semantic interpretation appears to be that Canada, USA, Nicaragua and Mexico are wholly within the northern hemisphere, Chile is wholly within the southern hemisphere, and Brazil is split between both hemispheres. However, the way the domain is used may change the meaning, so that uncertainty is inherent in relations using this domain. Consider relation R = (name, location) shown in Table 2, where *location* is defined on the lattice-structured domain described in Table 1. If the locations refer to a specific, if unknown, address, then this relation says that John lives somewhere in Brazil, and almost certainly either in the northern hemisphere or the southern hemisphere, but not both. This introduces a degree of uncertainty into whether or not conditions on these elements hold or not. Given any condition C on attribute

Table 2. Relations R, S and Q using a lattice-structured domain

Relation R		Relation S		Relation Q	
name	location	name	location	location	language
John	Brazil	Susan	South America	Canada	English
Susan	Chile	Peter	Brazil	Canada	French
Peter	southern hemisphere			USA	English
Betty	Nicaragua			Mexico	Spanish
				Nicaragua	Spanish
				Brazil	Portugese
				Chile	Spanish

A defined on lattice-structured domain D, that condition can be categorised as *known to be true*, *uncertain* (may or may not be true), or *known to be false*. For example, the condition `location = 'southern hemisphere'` is known true for Susan, known false for Betty, and uncertain for John. These categories create three mutually exclusive sets T, U and F for the elements $e \in D$, where $T = \{e|C \text{ true}\}$, $U = \{e|C \text{ uncertain}\}$ and $F = \{e|C \text{ false}\}$. In addition to these sets, we can define another three (useful) groupings by combining them in pairs. This leads to the definition of the six inductive conditions shown in Table 3. The symbol ? indicates that condition C may be true (i.e. true or uncertain), and the symbol ! that the value of C is not uncertain (i.e. true or false). The

Table 3. Inductive conditions

condition	meaning	included elements
C	C true	T
$\neg C$	C false	F
$\neg(!C)$	C uncertain	U
$?C$	C true or uncertain	$T \cup U$
$\neg(?C)$	C uncertain or false	$U \cup F$
$!C$	C true or false	$T \cup F$

inductive conditions degenerate as expected where no uncertainty exists. In the case of unstructured domains $U = \phi$, so $?C$ and $\neg(?C)$ are equivalent to C and $\neg C$ respectively, and $!C$ is always true and $\neg(!C)$ always false.

As well as the uncertainty introduced in relation R by saying that John lives somewhere in Brazil, there is another type of uncertainty in saying that Peter

lives in the southern hemisphere. This means that Peter might live in either Chile or Brazil, but not both. One type of uncertainty is introduced because of the use of an element in the tuple at a higher level than that in the query (type 1 uncertainty), and another because of the use of an element in the query that is a member of more than one higher level concept (type 2 uncertainty – this does not arise if structures are limited to trees).

Consider the conditions C_1 (`location = 'Brazil'`) and C_2 (`location = 'northern hemisphere'`), where C_1 leads to type 1 uncertainty, and C_2 to type 2. Table 4 shows the names in sets T, U and F for each condition. Only T

Table 4. Sets T, U and F for conditions C_1 and C_2

	C_1	C_2
T	{John}	{Betty}
U	{Peter}	{John}
F	{Susan, Betty}	{Susan, Peter}

and U need to be calculated by examining the lattice defined by relation DS, since $F = \Omega - T - U$. Once T and U have been determined, the type of uncertainty is no longer of importance. Consider a simple condition of the type $a = e$, where a is an attribute defined on a structured domain, and e is an element from that domain. For domains with type 2 uncertainty, $T = \{t | t \in R \wedge (a = e \vee a$ must be a descendent of $e\}$ and $U = \{t | t \in R \wedge (a$ is an ancestor of $e \vee a$ may be a descendent of $e)\}$. For domains with only type 1 uncertainty, if there are no lower level concepts with multiple parents, the algorithm needed to construct sets T and U can be simplified. Structurally, the difference between type 1 and type 2 uncertainty for domain value e is whether any descendents of the node e in the lattice have a path to the top of the lattice which does not go through e.

In the case of compound conditions such as $C_a \wedge C_b$ or $C_a \vee C_b$, the set to which a given tuple belongs can be determined by considering the 3-valued logic truth tables shown in Table 5. Here t, u and f represent the truth values true, uncertain and false respectively.

4 Determining Truth Values for Inductive Queries

In order to construct sets T and U, we need to be able to identify higher and lower level concepts for elements from a structured domain. Let e be an elemental value from a lattice-structured domain D whose structure is defined by domain concept relation DS. $H(e)$ is the set of higher-level concepts of e, i.e. $H(e) = \{c | c \in D \wedge c$ is an ancestor of $e\}$, and $L(e)$ is the set of lower-level concepts of

Table 5. 3-valued logic truth tables

C	$\neg C$
t	f
u	u
f	t

$C_a \vee C_b$	t	u	f
t	t	t	t
u	t	u	u
f	t	u	f

$C_a \wedge C_b$	t	u	f
t	t	u	f
u	u	u	f
f	f	f	f

e, i.e. $L(e) = \{c | c \in D \wedge c$ is a descendent of $e\}$. If e is a base concept, then $L(e) = \phi$, and if e is a top-level concept, then $H(e) = \phi$. If D is an unstructured domain, then $L(e) = H(e) = \phi$ for all e. For our example, H(South America) = {Americas}, L(South America) = {Brazil, Chile}, H(Brazil) = {South America, Americas, southern hemisphere, northern hemisphere} and L(Brazil) = ϕ.

Another useful hierarchical function $A(e_1, e_2)$ is defined as the set of all elements from D which are ancestors of e_2 which are not related to e_1. If $\{e_2 | e_2 \in L(e_1) \wedge A(e_1, e_2) \neq \phi\}$ is not empty, then type 2 uncertainty exists for a query using e_1. The function ancestor is used to formally define the set-valued functions H, L and A, and thus determine the sets T and U for a given domain element value e. These functions are defined below.

$$
\begin{aligned}
\text{ancestor}(x, y) &= (x, y) \in DS \vee ((x, z) \in DS \wedge \text{ancestor}(z, y)) \\
H(x) &= \{y | \text{ ancestor}(x, y)\} \\
L(x) &= \{y | \text{ ancestor}(y, x)\} \\
A(x, y) &= \{z | z \in H(y) \wedge z \notin \{x\} \cup H(x) \cup L(x)\} \\
T &= \{x | x = e \vee (x \in L(e) \wedge A(x, e) = \phi)\} \\
U &= \{x | x \in H(e) \vee (x \in L(e) \wedge A(x, e) \neq \phi)\}
\end{aligned}
$$

Evaluating functions H, L and A involves the recursive function ancestor. Determining the sets T and U involves evaluating $A(e, x)$ for all $x \in L(e)$. To determine T, U and F for a query with type 2 uncertainty involving domain element e:

1. Evaluate $H(e)$ and $L(e)$.
2. Evaluate $A(e, x) \forall x \in L(e)$.
3. Calculate $T = \{e\} \cup \{x | x \in L(x) \wedge A(e, x) = \phi\}$.
4. Calculate $U = H(e) \cup \{x | x \in L(x) \wedge A(e, x) \neq \phi\}$.
5. Calculate $F = \Omega - T - U$.

For a domain with only type 1 uncertainty, $A(e, x) = \phi$ for all $x \in L(e)$, which eliminates step 2 and simplifies steps 3 and 4. Algorithms whose complexity can be described in terms of the number of elements n in the domain have been developed for functions $H(e)$, $L(e)$ and $A(e, x)$ [8]. These algorithms are all polynomial – $O(n^4)$.

5 Relational Algebra Operations for Lattice-structured Domains

Introducing lattice-structured domains necessitates some extensions to the relational languages underpinning the relational model. Consider the five basic operations of the relational algebra: σ (selection), π (projection), $\times$ (cartesian product), $\cup$ (set union) and $-$ (set difference). Working with these domains will not affect π and $\times$, but will affect σ, $\cup$ and $-$. All other operations, such as $\cap$ (set intersection) and $\bowtie$ (join), can be constructed from the basic operations.

For simplicity, the relations in this section all contain exactly one attribute defined on a lattice-structured domain. The examples use the relations R and S (primary key `name`), and relation Q (primary key `(location, language)` defined in Table 2. As is usual for SQL, results of queries are allowed to include duplicate tuples. The changes required for σ are due to the effect of structured domains on conditions as previously described. This can be indicated in the algebra by using the inductive condition notation developed in the previous section. For example, $\sigma_{?C_2}(R)$ would return all tuples from R where C_2 is true or uncertain, i.e. {(Betty, Nicaragua), (John, Brazil)}.

The various join operations where join-columns are defined on lattice-structured domains can be expressed using inductive conditions in a similar way. Consider relation Q which shows the official languages for all six countries. The attribute `location` is defined on a flat subset of domain D which has been restricted to countries. $R \bowtie_{?(R.\text{location} = Q.\text{location})} Q$ returns all tuples from $R \times Q$ where the location values are or may be equal, whereas $R \bowtie_{R.\text{location} = Q.\text{location}} Q$ only includes the tuples where the location values are known to be equal. Table 6 shows some joins involving relations Q and R.

For the set operations, uncertainty can be introduced when relations with different levels of refinement for the same information are combined. Without further knowledge, it is reasonable to choose the information with the finest granularity when combining these relations, but sometimes this involves a loss of information. Susan has different locations in each relation, Chile and South America, only one of which can appear in $R \cup S$ and $R \cap S$. Chile is the most specific, so it is chosen. Peter on the other hand has the locations southern hemisphere and Brazil. Choosing either location causes the loss of some information, either the hemisphere, or the country. Without allowing attributes to take on multiple values, the best we can do is to make a decision about which information is the most useful. In cases such as this where one of the values is a descendent of the other in the lattice, the best method is likely to be to choose the descendent, and include or exclude the tuple depending on the set operation being performed. (Where neither of two different values representing the same information is the descendent of the other, they represent inconsistent information.) Table 7 shows $R \cup S$, $R \cap S$ and $R - S$.

Table 6. Joins involving relations R and Q

$R \bowtie_{R.\text{location} = Q.\text{location}} Q$

name	R.location	Q.location	language
Betty	Nicaragua	Nicaragua	Spanish
John	Brazil	Brazil	Portugese
Susan	Chile	Chile	Spanish

$R \bowtie_{?(R.\text{location} = Q.\text{location})} Q$

name	R.location	Q.location	language
Betty	Nicaragua	Nicaragua	Spanish
John	Brazil	Brazil	Portugese
Peter	southern hemisphere	Brazil	Portugese
Peter	southern hemisphere	Chile	Spanish
Susan	Chile	Chile	Spanish

$R \bowtie_{?(R.\text{location} = \text{'Brazil'})} Q$

name	R.location	Q.location	language
John	Brazil	Brazil	Portugese
Peter	southern hemisphere	Brazil	Portugese

$R \bowtie_{?(R.\text{location} = \text{'northern hemisphere'})} Q$

name	R.location	Q.location	language
Betty	Nicaragua	Nicaragua	Spanish
John	Brazil	Brazil	Portugese

6 Query Language Implications

Query languages such as SQL include statements for data definition and data update as well as querying the database. In this section describe an extension to the WHERE clause for inductive queries. See [8] for suggestions on defining lattice-structured domains and how the syntax of multi-table queries using the JOIN keyword might be adapted.

The WHERE clause needs to be adapted to incorporate inductive queries using the ? and ! symbols. The keyword MAYBE can be used as a condition modifier to represent ? with no confusion, since ? and $\neg$ are associative (i.e. $\neg(?C) = ?(\neg C)$). Since $!C = !(\neg C)$ care needs to be taken with choosing a keyword for !. TRUEORFALSE, although clumsy, has the advantage that the equivalence between TRUEORFALSE C and TRUEORFALSE NOT C is clear, and NOT TRUEORFALSE C can be used for $\neg(!C)$. UNCERTAIN C could be used instead of NOT TRUEORFALSE C, but this might lead to semantic confusion

Table 7. Results of set operations using R and S

$R \cup S$		$R \cap S$		$R - S$	
name	location	name	location	name	location
John	Brazil	Susan	Chile	John	Brazil
Susan	Chile	Peter	Brazil	Betty	Nicaragua
Peter	Brazil				
Betty	Nicaragua				

between UNCERTAIN and MAYBE. Using just MAYBE and TRUEORFALSE leads to WHERE clauses of the type shown below:

```
WHERE NOT TRUEORFALSE location = 'northern hemisphere'
WHERE MAYBE location = 'northern hemisphere'
WHERE MAYBE NOT location = 'northern hemisphere'
WHERE TRUEORFALSE location = 'northern hemisphere'
```

7 Conclusions and Further Research

Hierarchical domains are useful whenever there is a need to combine data from relations with different schemas, whether that difference arises from schema evolution over time in the same database or from relations from different sources. One important area of application is likely to be queries of web databases. Concept hierarchies used for web-based queries are discussed by Davulcu et al in [3].

Incomplete and imprecise information of the type that can be represented in concept and interval lattices can arise in many ways, such as by summarising or sampling data, from space restrictions imposed by mobile equipment or main memory databases, by needing to infer values for points not explicitly stored, and performance considerations in multimedia databases.

Allowing domain hierarchies to be lattice-structured rather than simple trees does not add much complexity to hierarchical domains: their definition and representation in the database is no more complex, and the additional (type 2) uncertainty introduced does not increase the complexity of the algorithm for determining the results of inductive queries.

Further work needs to be done to investigate the use of the lattices of intervals and attribute sets discussed in Section 2 of this paper, and how best to represent and reason with inconsistent information in databases. It is also important to test the behaviour of the hierarchical algorithms on realistic examples, and improve them where necessary.

References

1. P. Besnard and T.H. Schaub. Circumscribing inconsistency. In *Fifteenth International Joint Conference on Artificial Intelligence*, 1997.
2. D. Corbett and R. Woodbury. Unification over constraints in conceptual graphs. In W. Tepfenhat and W. Cyre, editors, *Seventh International Conference on Conceptual Structures*. Springer, 1999.
3. H. Davulcu, J. Freire, M. Kifer, and I.V. Ramakrishnan. A layered architecture for querying dynamic web content. In *ACM SIGMOD*, pages 491–502, Philadelphia, USA, 1999. ACM.
4. Z. Kedad and E. Metais. Dealing with semantic heterogeneity during data integration. In Jacky Akoka, Mokrane Bouzeghoub, Isabelle Comyn-Wattiau, and Elisabeth Metais, editors, *18th International Conference on Conceptual Modeling*, volume 1, pages 325–339, Paris, 1999. Springer.
5. H. Mannila. Inductive databases and condensed representations for data mining. In Jan Maluszynski, editor, *International Logic Programming Symposium*. MIT Press, 1997.
6. S. Parsons. Current approaches to handling imperfect information in data and knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):353–372, 1996.
7. D. Perlis. Sources of, and exploiting, inconsistency: preliminary report. *Applied Non-Classical Logics*, 7(1):13–24, 1997.
8. S.P. Rice and J.F. Roddick. Lattice-structured domains to represent imperfect data in databases. Technical Report ACR-00-006, University of South Australia, June 2000.
9. J.F. Roddick. *A Model for Temporal Inductive Inference and Schema Evolution in Relational Database Systems*. Doctor of philosophy, La Trobe University, 1994.
10. J.F. Roddick, N.G. Craske, and T.J. Richards. Handling discovered structure in database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):227–240, 1996.
11. J.F. Roddick and S.P. Rice. Towards inductive queries. In *Ninth Australasian Conference on Information Systems*, volume 2, pages 534–542, Sydney, Australia, 1998. University of New South Wales.
12. P. Smets. Probability, possibility and belief: which and where? In Dov M. Gabbay and Philippe Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems: Quantified Representation of Uncertainty and Imprecision*, volume 1, pages 1–24. Kluwer Academic Publishers, 1998.