What Constitutes a Scientific Database?

John L. Pfaltz
Dept. of Computer Science, Univ. of Virginia
Charlottesville, VA 22904-4740

E-mail: jlp@virginia.edu

Abstract

We propose that a scientific database should be inherently different from, say a business database. The difference is based on the nature of science itself, in which hypotheses, or logical implications, form an essential part of the discipline. Empirical observations give rise to tentative hypotheses. Individual hypotheses are then tested, refuted or refined, by further empirical observation.

In the paper, we propose representing the observational data of science in a lattice format that also conveys all the logical implications that can be supported by those observations. We claim that such a structure can be incrementally created and that the hypotheses formed will adapt to new data. We demonstrate its practicality by presenting two real situations in which it has been used.

Finally, we look at the rather considerable storage costs associated with this approach and discuss other limitations that are still unresolved in this new approach to the representation of scientific data.

1 Introduction

What constitutes a "scientific" database? For example, it is in any way different from a "business" database? To a large extent it depends on what one regards as "science". Certainly, a key aspect of science is the observation of properties of real phenomena and the subsequent recording of these observations. For this, relational databases might seem quite adequate because rows can denote phenomena, or objects, and columns can denote their attributes.

Nevertheless, the relational model as it has evolved through SQL [8, 10] seem to be far better suited for accounting and certain kinds of statistical databases, than for the more general notion of scientific enquiry. Consequently there have been many extensions of SQL to better represent the needs of the scientific community. Briefly, these may be categorized as spatial, temporal and evolutionary needs.

1.1 Spatial Scientific Databases

Corporate accounting and administrative phenomena occur in at most a one dimensional (temporal) world. A flat tabular structure with a single join operator to relate items in different tables is completely adequate. But scientific phenomena occurs in a 3-dimensional universe, and mechanisms to describe and manipulate spatial relationships have been recognized as essential. Since the late 80's there have been many extensions of the relational model to accommodate spatial data, c.f. [12, 15, 41] Some have been more successful than others. If the objects (tuples) themselves are spatially distributed then it is easy to denote this with 3 coordinate attributes. But, neither SQL nor the relational model provides spatial operators and many, such as "nearest neighbors", are difficult to implement.

1.2 Temporal Scientific Databases

Time is easy to represent in any database through the use of time stamps. The problem here seems only to find operators that facilitate search and/or use of this information. Linear temporal logic has centered on four standard operators:

$X\alpha$	denoting	"next after α"
$F\alpha$	$''$	"eventually α"
$G\alpha$	$''$	"generally (or always) α"
$\alpha U\beta$	$''$	"α until β"

as described in [13]. Some of these have been added to SQL to enhance its search capabilities. [4, 45].

1.3 Evolving Scientific Databases

The need to represent and manipulate spatial and temporal data has been a well-known difference between "scientific" and "business" data systems. Less frequently mentioned is the need for scientific system to be evolutionary. The required attributes, or columns, of a relational accounting database have been known for years and have been standardized with respect to GAAP accounting principles. Many other business database applications, such as inventory control, are similarly well developed. One changes the data, but not the schema.

But in science, one often does not know in advance just what properties, or attributes, will be important. Frequently, in the course of scientific enquiry one realizes that other aspects of the phenomena must be observed and recorded. This requires not only changing the schemas of the database itself, but also dealing with the attendant "missing values", c.f. [39, 40, 42].

1.4 Object-Oriented Databases

Criticizing relational database systems for inadequately supporting scientific data is a bit like "flogging a dead horse". These problems have been known for years and have constituted one major impetus for creating the object oriented model [2]. We have only repeated the many arguments that have already been presented in order to illustrate the common belief that it is the inclusion of temporal and spatial data, together with schema evolution that is critical to the concept of a scientific database.

Object-oriented databases can provide mechanisms to resolve many of these deficiencies. We can represent spatial objects, temporal objects, set objects and use inheritance to specialize individual varieties of these scientific concepts. The author had created one such object-oriented system called ADAMS [36], which easily supported schema evolution and unbounded scientific subscripting [31, 35]. There have been many others, such as: EXODUS [6], Gemstone [5], O_2 [24], Cactis [20], ROSE [19], ODE [1] and ObjectStore [23].

An object-oriented approach provides the facility to customize tailored scientific database systems. Yet, pure O-O systems seem to be relatively rare in practice. The author would offer two reasons for this. First, they tend to be slow. Yet, the early relational systems were themselves abysmally slow relative to the then dominant hierarchical and network models. Second, and most important we believe, O-O databases encourage complexity. As anyone who had slugged through typical C++ code can attest, inheritance over multiple object templates can quickly become bewildering. The great strength of the relational model has been its utter simplicity.

Our approach begins with just such a simple relation. It then constructs another layer to model an aspect of science that has not been considered in any of the preceding discussion. Yet it is still a "simple" structure.

2 Assertions in a Scientific Database

Gathering data from observations and storing it is not, by itself, "science" — even though it is an essential part of science, Most believe that the nature of "science" requires the formulation of hypotheses, or logical assertions, based on empirical observation which can be then tested against more experimental evidence. This is not an original idea; others have explored the formulation of implications from scientific data [25, 28]. The contribution of this paper is a "scientific database" structure that can record both the data and the hypotheses in a surprisingly uniform manner.

There can be many kinds of scientific hypotheses. But, broadly speaking they will fall into two general categories: deterministic and probabilistic. In this paper, we shall be concerned solely with deterministic hypotheses. We are not saying that statistical hypotheses should not be included with probabilistic data; rather we are only saying that we don't know how to do it!

Customarily, a scientific hypothesis is a universally quantified implication. Existential assertions can be important in science, but they are not the kind of hypothesis that requires repeated experimental evidence to confirm or refute. We will assume that a scientific hypothesis is an implication of the form $(\forall o)[P(o) \rightarrow C(o)]$, that is, if for any object o, if the premise $P(o)$ is true then the consequent, or conclusion, $C(o)$ must be true as well. For example, we might assert that "if o is a freely falling object then o accelerates at a rate of 980 centimeters/second/second".

It is not hard to present experimental evidence refuting this assertion as stated. For example, any object with substantial air resistance falls more slowly. In science, either the hypothesis may be dis-

carded altogether or, more commonly, the premise may be refined. In this case we might assert that "every object freely falling in a vacuum accelerates at a rate of 980.665 centimeters/second/second". (Subsequent experiment can also change the consequent.) Letting f denote the property of freely falling, v denote presence of a vacuum, sl denote sea level, and g denote the "gravitational constant", we finally rewrite the hypothesis as $(\forall o)[f(o) \wedge v(o) \wedge sl(o) \rightarrow g(o)]$. Conjunctive premises seem to be a hallmark of scientific assertions.

We want such deterministic assertions to be effectively represented in the same database that records the experimental observations themselves. It will require a rather different database structure.

2.1 Formal Concepts and Galois Closure

The approach we have followed is similar to Formal Concept Analysis (or FCA) that was first developed by Rudolf Wille and is best presented in [17]. FCA begins with a relation R between two sets, say a set $\mathcal{O}$ of objects and a set $\mathcal{A}$ of object attributes, or predicates. Using standard relational terminology, each object $o_j \in \mathcal{O}$ can be regarded as a row in R and each attribute $a_k \in \mathcal{A}$ is a column. Each attribute a_k is a binary, logical property, $i.e.$ true or false. A concept C_n is a pair of subsets $C_n = (O_n, A_n)$ where $O_n \subseteq \mathcal{O}$, $A_n \subseteq \mathcal{A}$ with the property T that for every $o_i \in O_n$, every $a_k \in A_n$ is true. Each concept is assumed to be maximal, that is for the set O_n there is no larger subset $A'_n \supset A_n$ satisfying property T, and for A_n there is no larger subset $O'_n \supset O_n$ satisfying P.

The collection $\mathcal{C}$ of all concepts C_n, so defined, forms a closure system; that is, the intersection of any two concepts in $\mathcal{C}$ is a concept. Consequently, the collection $\mathcal{C}$ of concepts forms a lattice when partially ordered with respect to the attribute sets A_n by containment.[1] An example of a relation R and corresponding lattice $\mathcal{L}$ is shown in Figure 1. Each node is labelled with a closed set of attributes and a closed set of objects — the concept pair. For example, the combination of attributes adf is found in rows, or objects, 5, 6 and 8.

For any closure system $\mathcal{C}$ defined in terms of being closed under intersection as we have done, there will be a closure operator φ satisfying the three closure axioms:
C1: $\forall X \in \mathcal{C}$, $X \subseteq \varphi(X)$,

[1] Ganter and Wille [17] prefer to order with respect to object set containment yielding the dual lattice.

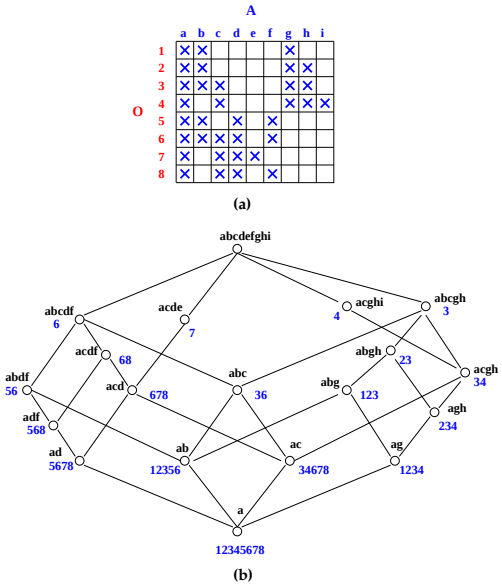

Figure 1. Relation R (a) and concept lattice $\mathcal{L}$ (b).

C2: $\forall X, Y, \in \mathcal{C}$, $X \subseteq Y$ implies $\varphi(X) \subseteq \varphi(Y)$,
C3: $\forall X$, $\varphi(\varphi(X)) = \varphi(X)$
that yields the same system. In the case of FCA, the closure operator is called the Galois closure between $\mathcal{O}$ and $\mathcal{A}$, and has been well studied [7, 9, 30] In this paper we emphasize the closure aspect as denoted by φ, rather than the concept aspect developed in [17].

2.2 Closed Sets, Generators and Logical Implication

Let C denote a closed set. Then there is some set $A \subseteq C$ such that $\varphi(A) = C$. If A is a minimal such set, w.r.t. set inclusion, we call it a generator of C denoted by $\gamma(C)$, or by $A \rightarrow C$ [32]. The latter symbolism is not accidental. If φ is a Galois closure, then closed set generation and logical implication are identical.

The issue now becomes: "given a lattice of closed concepts, such as Figure 1(b), how does one derive the generating sets?" For example, what is the generator of the closed set $acde$ in Figure 1(a)? In [21] it is shown that "if C_1 covers C_2 in a closure lattice $\mathcal{L}$[2] then $\gamma(C_1) \cap (C_1 - C_2) \neq \emptyset$." With this theorem, one can construct the generator of any closed set C as a combination of elements $e_i \in (C - C_i)$ where C covers C_i in $\mathcal{L}$. The

[2] C_1 covers C_2 if we do not have $C_2 \subset C \subset C_1$, C closed.

node $acde$ covers only one node acd in the lattice $\mathcal{L}$, so $C - C_1 = \{e\}$. Thus, using the result above we can show that the generator $\gamma(acde) = \{e\}$, or $e \rightarrow acde$. That this is true is evident from Figure 1(a). Attribute e is only found in observation 7, where a, c, and d were also seen. So the property e, in this case, trivially implies properties a, c and d. With the result above, the derivation of generating sets is an inexpensive, local construction whose details can be found in [37]. One may choose to explicitly represent the generating sets of each node in $\mathcal{L}$, or simply reconstruct them "on the fly".

2.3 Updating the Data Lattice

The nature of any working data base is that new data will be constantly added; this is certainly true of on-going scientific enquiry. Given an entire relation R, as in Figure 1(a), there exist algorithms to construct its closed set lattice $\mathcal{L}$, [16, 17]. However we, and others [18, 46], find it preferable to construct $\mathcal{L}$ incrementally, one row or observation, at a time. This is certainly similar to real data acquisition.

By the nature of Galois closure each new row, or observation properties, is closed. So it either already exists as a node in $\mathcal{L}$, or will constitute a new node. First, one must find its location in $\mathcal{L}$ using set inclusion in a search down from the top (or up from the bottom) of $\mathcal{L}$. If the observation has no new properties and so already exists we simply increment the set of occurrences. Otherwise, the new set of properties is inserted as a closed set C, covered by some existing set C', which is the smallest set containing C. Usually C' will be covering other closed sets C_i in $\mathcal{L}$. Because of the intersection property of closed set lattices described in Section 2.1, we must now calculate $C \cap C_i$ for each C_i covered by C'. If $C \cap C_i$ already exists in $\mathcal{L}$, nothing more needs be done; otherwise $C \cap C_i$ must itself be recursively entered into $\mathcal{L}$ as well. This can create a recursive cascade of insertions, but as noted later in Section 5.2. this is fortunately rather rare.

In Figure 2 a new observation, labeled 9, has been entered with the properties a, b and e. The node abe has been entered into $\mathcal{L}$ where it is covered by the top node consisting of all attributes. Now we must check the intersection of this closed set abe with all others covered by the top of $\mathcal{L}$. We have $abe \cap abcdf = ab$, $abe \cap acde = ae$, $abe \cap acghi = a$ and $abe \cap abcgh = ab$. Of these, only ae is new. It must be recursively entered as indicated in Figure 2(b) by the dashed lines.

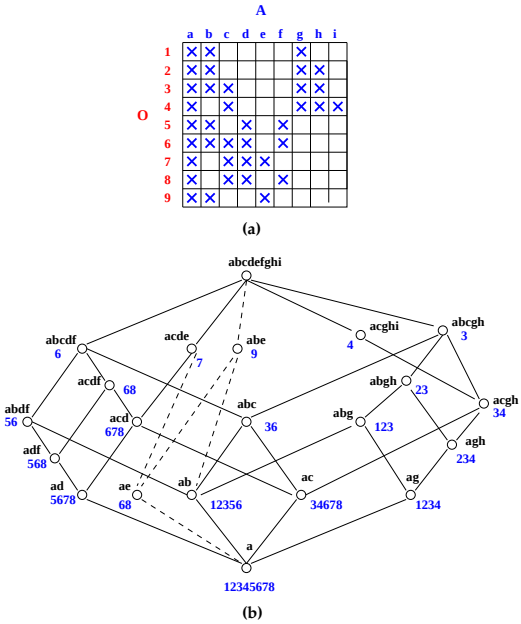

Figure 2. Modified relation R (a) and Resulting lattice $\mathcal{L}$ (b).

Because abe covers both ae and ab in $\mathcal{L}$ and $abe - ae = \{b\}$, $abe - ab = \{e\}$, its generator must be be. Of more interest is how the generator of $acde$ has changed. Initially we had $e \rightarrow acde$ because $acde$ only covered acd. Now we have $ce \vee de \rightarrow acde$ because $acde - acd = \{e\}$ and $acde - ae = \{cd\}$. This can be more formally expressed as $(\forall o \in O)[(c(o) \wedge e(o)) \vee (d(o) \wedge e(o) \rightarrow a(o) \wedge c(o) \wedge d(o) \wedge e(o)]$. As more information is entered the generators, or premises, of many closed sets, or conclusions, are expanded in this way while others may be simplified.

Observation and new data can change our understanding of the world.

3 Two Implemented Examples

A fundamental question is "can this lattice structured database be practical?" Does it really work? It does; and we offer the two following working examples to simply demonstrate proof of concept.

3.1 The MUSHROOM Database

Using precisely the mechanisms described in the preceding section, we took as input the physical properties of 8,124 different mushrooms as given in the "The Audubon Society Field Guide to North

American Mushrooms" [26]. Figure 3 illustrates representative values of the first 6 (of 22) attributes used to characterize mushrooms in [26] Each at-

```
Attr-0  edibility:
  e=edible, p=poisonous
attr-1  cap shape:
  b=bell, c=conical, f=flat, k=knobed, s=sunken,
   x=convex
attr-2  cap surface:
  f=fibrous, g=grooved, s=smooth, y=scaly
attr-3  cap color:
  b=buff, c=cinnamon, e=red, g=gray, n=brown,
  p=pink, r=green, u=purple, w=white, y=yellow
attr-4  bruises?:
  t=bruises, f=doesn't bruise
attr-5  odor:
  a=almond, c=creosote, f=foul, l=anise, m=musty,
  n=none, p=pungent, s=spicy, y=fishy
attr-6  gill attachment:
  a=attached, d=descending, f=free, n=notched
```

Figure 3. The first 6 attributes of the MUSHROOM **data set, with nominal values.**

tribute was encoded as one, or more, separate predicates. For example, we have the predicates `e0` for "edible", `p0` for "poisonous", `a6` for "attached gill" and `n6` for "notched attachment". Encoded this way the attributes of Figure 3 yield 42 effective predicates.[3] The lattice generated by the $8,124 \times 42$ relation R consists of 2,641 closed concepts[4]; and, because some concepts have multiple generators, 3,773 distinct implications, or rules.

To provide some sense of this data set we list in Figure 4 all those rules $P \rightarrow C$ in which a singleton predicate P implies `p0`, or poisonous. We

CONCEPT	IMPLICATION	SUPPORT
668	c5 -> p0, x1, f4, f6	192
924	f5 -> p0, f6	2160
1401	g2 -> p0, w3, t4, n5	4
1597	s5 -> p0, f4, f6	576
1687	y5 -> p0, f4, f6	576
2022	m5 -> p0, y2, f4	36
2562	c1 -> p0, n5, f6	4

Figure 4. All implications in MUSHROOM **with $|P| = 1$ and `p0` $\in C$.**

have added the concept number to the left to indicate where this rule was uncovered in the observations and the number of times the implications has be observed, or its support, to the right.

Are there simple combinations of attributes that also denote poisonous? Figure 5 illustrates those non-trivial conjunctive implications $P \rightarrow C$ for which $|P| = 2$ and `p0` $\in C$.

CONCEPT	IMPLICATION	SUPPORT
667	p3, f4 -> p0, x1, c5, f6	64
696	f2, p3 -> p0, x1, f4, c5	32
1495	b1, b3 -> p0, t4, n5, f6	12
1567	b1, p3 -> p0, t4, n5, f6	12
2081	y3, n5 -> p0, f4, f6	24
2177	e3, f4 -> p0	876
2181	y2, a6 -> p0, f4, m5	18
2372	c3, a6 -> p0, y2, f4, m5	6
2470	e3, a6 -> p0, y2, f4, m5	6
2561	c1, y3 -> p0, y2, f4, n5	2
2561	c1, f4 -> p0, y2, y3, n5	2
2563	c1, y2 -> p0, n5, f6	3

Figure 5. Rules with two predicate precedents that denote poisonous mushrooms.

A more detailed description of this application can be found in [37].

3.2 Analysis of Software Trace Data

An important concept in science is that of deterministic causality in which the occurrence of some event, or conjunction of events, must necessarily "cause" a consequent event. Indeed, this was the holy grail of Newtonian physics and much of 19^{th} century science. "Causality" implies necessity, or logical implication. But, it also assumes a temporal aspect. The consequent event must temporally follow all assumed antecedent events. One arena where we expect deterministic causality is software execution. It is a reasonable place to test our ideas.

The application of FCA to re-engineering of legacy software has been explored by others [3, 27]. They used closed concepts to reveal significant clusters of code modules. Our interest instead has been to uncover "likely" causal dependencies between such modules.[5] To do this we examine trace sequences of procedure invocations, which we regard as events. (Each procedure invocation, or event, can be assigned an integer identifier for easier display, as in Figure 6.) The first step is to find which events logically imply other events, that is, which events e_i, if they occur in a sequence, must imply the occurrence of events e_k within the same sequence. To do this we treat the occurrence of an event as if it were an attribute of the sequence and create the closure lattice as in Section 3.1.

[3] Encoding all 22 physical attributes yields 85 distinct predicates.

[4] The entire $8,124 \times 85$ relation R generates a lattice of 104,104 closed concepts.

[5] Michael Ernst coined the term "likely" in his search for code invariants [14]. It seems extremely appropriate here as well.

As we noted above, logical implication denoted by $\rightarrow$ need not mean deterministic causality, which we will denote by $\Rightarrow$. If $P \rightarrow C$ then we can assert $P \Rightarrow C$ only if for each $e_k \in C$, it has been preceded in the trace by every $e_i \in P$.[6] Each trace can be treated as a temporally ordered set. We are then looking for events that temporally dominate other events.

To test this approach on real trace data, the author examined an open source, professional statistical package available from $JBoss$ at `www.jboss.com`. All of the method entrance events of the transaction management module in $JBoss$ 1.4.2 were instrumented by my colleagues, Jinlin Yang and David Evans [47]. They then ran the entire $JBoss$ regression test suite to collect 1,227 trace sequences consisting of 498,489 events of which 144 were distinct. By an "event" in these traces we mean the invocation of a method. The shortest trace consisted of only 6 events; the longest involved 1,405 events.

The representation of these 1,272 trace sequences as a closed set lattice consisted of 1,804 nodes. For simplicity we extracted the 79 logical implications that had only singleton antecedents, similar to Figure 4. These we ran against a temporal filter yielding 43 likely causal dependencies. An even smaller subset of 17 of these is shown in Figure 6. From what we know of the $JBoss$ system, without having the actual source code, these all seem to be true dependencies. For example, the underlying stack architecture is apparent from sequences such as `35=>36=>37=>38`. More details can be found in [34].

concept	support size	causal dependencies (likely)
1733	1,099	{12}=>{13...24}
445	1,100	{17}=>{22,23}
		{20}=>{21}=>{22,23}
251	966	{25}=>{26}=>{27}=>{28}=>{29}
		{28}=>{30}=>{31}=>{32}
391	962	{35}=>{36}=>{37}=>{38}
443	977	{41}=>{42}=>{43}=>{44}
945	852	{46}=>{47,48,49,60,62}
458	1,077	{47}=>{48}=>{49}=>{51}=>{60}=>{62}
448	1,098	{50}=>{53}=>{54}=>{55}=>{58}=>{61}
53	1,091	{56}=>{57}
375	28	{66}=>{67,69,70}
1754	28	{68}=>{69,70,71,72}
1745	65	{73}=>{74}=>{75}
725	3	{84}=>{117}=>{118}
575	62	{86}=>{87}=>{88}=>{25...44,45,63}
272	1	{89}=>{33,34,90...100)

Figure 6. Some likely causal dependencies.

Although the creation of closed set lattices, as described in section 2.2 and 2.3, together with the real life examples of this section may seem impossibly complex, this lattice structured representation of data based on closed sets is really rather simple. Precisely the same code (which is available from the author) was used to represent both mushroom attributes and software events. And in both cases there has been a clear "value added" by providing logical rules that can be exploited by working scientists, $c.f.$ [29, 44].

4 Numeric Data

The vast majority of scientific data is numeric. Unfortunately, FCA works well with boolean predicates, but is problematic when the predicates are numeric. Our more recent research has shown that by treating each observation as a tuple of antimatroid closure spaces, rather than just a set of true predicates (as in Sections 3.1 and 3.2), we can represent a wide range of numeric assertions as well.

Now, each observed predicate, or entry in a row, of the relation R will be regarded as representing a single closed set Z_i in a closure space, $\mathcal{C}_i$. The entire row then denotes a single closed set, $Z_1 \times \ldots \times Z_n$ in the direct product space $\mathcal{C}_1 \times \ldots \times \mathcal{C}_n$. A formal justification of this approach can be found in [17, 38].

A closure space is said to be antimatroid if every closed set has a unique generating set. Every boolean space, comprised of just two closed sets $\{false\}, \{true, false\}$, is antimatroid. The Galois closure used to define the lattice of closed sets is not antimatroid; but we can require that each component space $\mathcal{C}_k$ of $\mathcal{C}_1 \times \ldots \times \mathcal{C}_n$ must be. The reason for this is that then any closed set Z_i can be represented by just its generating set, $\gamma(Z_i)$, rather than the entire set. For example the geometric convex hull operator is antimatroid [11]. Thus if spatial objects are encapsulated by their convex hulls, these objects can be represented in $\mathcal{L}$ by their generating verticies. Representation of spatial data by generators looks interesting but has not yet been explored.

It is with numeric data that this representation of closed sets by their generators can really payoff. Let $\mathcal{S}$ denote any partially ordered set, such as time, operator precedence, or the real numbers. A useful closure operator on $\mathcal{S}$ is downset, or ideal, closure $\varphi_{\downarrow}$, where $\varphi_{\downarrow}(Y) = \{x | x \leq y \in Y\}$. Readily,

[6] Since events may occur multiple times in a trace, we actually ensure that the $first$ occurrence of e_k is preceded by all $e_i \in P$.

the maximal elements of Y generate $\varphi_{\downarrow}(Y)$. When $\mathcal{S}$ is numeric, the generator $\gamma(Z)$ of any closed set $Z = \varphi_{\downarrow}(Y)$ is precisely its single maximum value.

In Figure 7, we have arbitrarily distributed el-

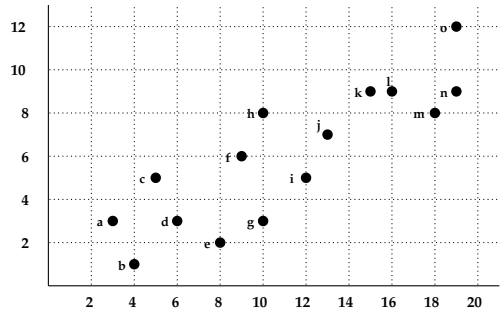

Figure 7. A scattering of points.

ements (x_i, y_i) in a 2-dimensional space. Using downset closure, $\varphi_{\downarrow}$, we get the lattice shown as Figure 8. Each element (x_i, y_i) constitutes a closed set in $\mathcal{L}$. Extra nodes that are forced by intersection have been underlined. Notice its rather tall, narrow structure in comparison to that of Figures 1 and 2.

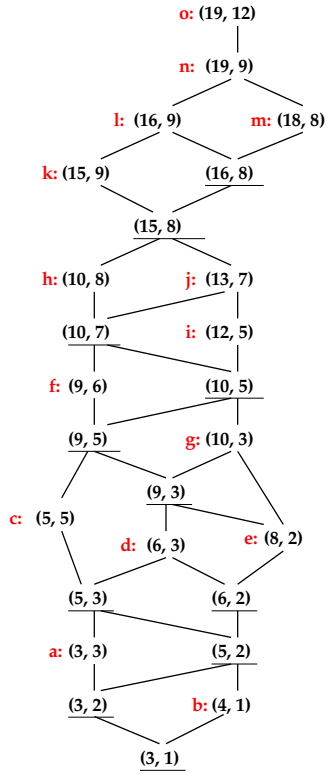

Figure 8. The lattice $\mathcal{L}$ formed of points from Figure 7

In order to derive logical implications for the numeric elements of a lattice such as $\mathcal{L}$, we turn our thinking around somewhat. In Section 2.2 we used the lattice covering relation to make assertions of the form $(\forall o)[p(o) \rightarrow c(o)]$, or "if p is true for all $o \in O$, then the consequent c must be true as well". Now, we will interpret this as "if p is $not\ false$ for all $o \in O$ then c $cannot\ be\ false$ either". This will make more sense with an example.

Consider the node c:(5,5) which covers $(5, 3)$ in $\mathcal{L}$. The generator of c is a value y that is not less than or equal 3, that is $y > 3$. In this case x cannot be less that 5, so we have $(y > 3) \rightarrow (x \geq 5)$. Comparison with Figure 7 shows this to be a valid implication. Node i:(12,5) covers node $(10, 5)$, so using this same covering principle we have $(x > 10) \rightarrow (y \geq 5)$ which again is seen to be valid in Figure 7.

Discovering just which kinds of numerical inequalities can be involved in scientific hypotheses is a rather exciting component of our current research.

5 Discussion

In the preceding sections we have demonstrated that by recording empirical data consisting of observed properties and observation identifiers in a lattice rather than a flat relational file, one also implicitly records all the logical implications that are derivable from that data. This offers impressive potential benefits; but there are also significant unresolved problems as well. In this section we will discuss both.

5.1 Storage Overhead

In our implementations, all sets of attributes, of predicates, of events, of observations, of lattice node links are represented by bit strings with one bit per element.[7] Besides compressing the representation, this technique permits the encoding of set operators as fast logical bit string operations. Nevertheless the representation of sets and links in the lattice structure adds to storage overhead.

However, more important is the creation of additional nodes in the lattice. Each distinct observation creates a node in the lattice. But, because the lattice models a closure system, non-empty intersections of nodes must be added to the lattice — as if they had been observed. For example, although there are only 8,124 items in the complete

[7] No upper limit on set size is assumed.

relational MUSHROOM data set, the lattice representation expands to 104,104 nodes. The 95,980 additional nodes represent a huge increase in storage in this case. Other applications may require more, or less, storage overhead depending on the degree of redundancy in the input data.[8]

Thus a lattice representation extracts a hefty price in storage overhead. But, storage has become increasingly cheap. Moreover, when we consider the additional overhead involved in storing $wysiwyg$ documents, such as WORD, compared to flat ASCII text files, or formatable text files, such as LaTex, this price does not seem so extraordinary.

5.2 Computational Cost

Because, in the worst case, one may have to sweep through the entire lattice to find the location of, and to insert, each new input node which can then force many new intersection nodes, the construction algorithm can be shown to be $O(n^3)$ over the entire data set. However, for two reasons this is not the case in practice. The effective construction cost seems to be closer to $O(n^{1.5})$, because one normally finds the insertion site in linear time, where actual insertion is typically local with few additional intersection nodes [33]. Moreover, scientific data tends to trickle in rather gradually. Because this representation supports incremental creation, whatever construction costs exist can be amortized over days, or even months.

5.3 Limited Data Representation

This is by far the most serious unresolved limitation with the representation as it stands today. In Section 1, we criticized the relational model because it failed to adequately represent temporal and spatial data, both of which can be fundamental to scientific enquiry. Yet, our examples of Section 3 only represent nominal data!

The entire approach employing concept lattices is based on closed sets and closure operators. It is essential that each observation value be a closed set. Ordered sets, such as numeric data, support the familiar $less\ than$, $greater\ than$, and $interval$ closures [32]. Indeed, numeric assertions in careful scientific discourse are typically characterized by numeric intervals, not equality. Section 4 has begun exploration of these opportunities.

[8] In the reduced case shown in Section 3.1 for illustrative purposes, only 176 of the 8,124 input rows are distinct. They generated the 2,641 nodes cited in the text.

Of the operators of linear temporal logic, X, U, F, G, only the first X (or $next$) is not a closure operator. Given the ease with which we could integrate temporal events in the analysis of software trace data, in Section 3.2, we believe that with some modification we may be able to make temporal assertions as well.

Spatial data is probably the hardest to represent in any system [43]. The generators of antimatroid closure can be valuable in the representation of spatial objects [22]. But, implementation of "covering", and "intersection" concepts are at this point still difficult.

5.4 Logical Feedback

The formulation of scientific hypotheses and empirical observation do not occur as distinct, separable events. Instead, they represent an on-going feedback process. Empirical observation stimulates scientific hypotheses, which are then used to direct more empirical research which may subsequently modify, or refute, them. By the same token, reasonably well established hypotheses, or assertions, may be used to question empirical observation. We have described in Section 2.3 how new data can be used to completely alter the generators, or premises, of an existing node or consequent. But, in our software, when that occurs to a well-established rule the observed data is temporarily rejected and flagged for special attention. The observation may be wrong. But, if the existential data is correct, as for example "an egg laying mammal", then it is clearly important and should still be singled out for special treatment.

These kinds of rare and unusual occurrences can be important. We return to Figure 4 which documents 7 different ways of identifying poisonous mushrooms on the basis of a single attribute. The five implications with support of at least 30 instances are invariably found by frequent set mining. But, we see that `c1` and `g2` also indicate poisonous; and with a support of only 4 instances, they are unlikely to be found. But, eating any of the 4 kinds of mushroom with a conical cap or grooved cap cover could have serious consequences.

6 Conclusion

The title of this paper is "what is constitutes a scientific database?" It's a good question. Provision for storing and retrieving spatial and temporal data, we would contend, are hallmarks of a good

"technical" database. Our city planning department operates a first rate spatial database, as do many engineering and architectural firms. Few are engaged in "science". The actual hallmark of a scientific database is the ease with which the underlying hypotheses, scientific assertions, rules or implications (call them what you may) can be obtained from the observed data. We have shown that representation of this data as a lattice of Galois closed sets facilitates precisely this capability.

But, as we have carefully pointed out, the representation of scientific data as a lattice of closed sets is not without its problems. It is storage intensive and still incapable of representing temporal and spatial data. But, if its problems seem large, so does its potential. It offers a unified way of interactively involving scientific observations with scientific hypotheses as evidenced by our two real life applications.

References

[1] R. Agrawal and N.H. Gehani. ODE (object database and environment): The language and the data model. In *Proc. 1989 ACM SIGMOD Conf.*, volume 18, pages 36–45, Portland, OR, June 1989.

[2] Malcolm Atkinson, Francois Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik. The Object-Oriented Database System Manifesto . In *First International Conf. on Deductive and Object-Oriented Databases*, pages 223–240, Kyoto, Japan, Dec. 1989.

[3] Thomas Ball. The Concept of Dynamic Analysis. *Proc. Seventh European Software Engineering Conf.*, pages 216–234, Sept. 1999.

[4] Michael H. Bohlen, Christian S. Jensen, and Richard T. Snodgrass. Temporal Statement Modifiers. *ACM Trans. on Database Systems*, 25(4):407–456, Dec. 2000.

[5] Paul Butterworth, Allen Otis, and Jacob Stein. The Gemstone Object Database Management System . *Comm. of the ACM*, 34(10):64–77, Oct. 1991.

[6] Michael J. Carey, David J. DeWitt, Joel E. Richardson, and E. J. Shekita. Object and File Management in the EXODUS Extensible Database System . In *Proc. 12th VLDB Conf.*, pages 91–100, Kyoto, Japan, Aug. 1986.

[7] Gabriele Castellini. *Categorical Closure Operators*. Birkhauser, Boston, 2003.

[8] Donald D. Chamberlin, Arthur M. Gilbert, and Robert A. Yost. A History of System R and SQL/Data System. In *Proc. 7th VLDB Conf.*, pages 456–464, Cannes, France, Sept. 1981.

[9] Klaus Denecke, Marcel Erné, and Shelly L. Wismath. *Galois Connections and Applications*. Kluwer Academic, Dordrecht, 2004.

[10] Paul DuBois. *MySQL*. New Riders Publ., 1999.

[11] Paul H. Edelman and Robert E. Jamison. The Theory of Convex Geometries. *Geometriae Dedicata*, 19(3):247–270, Dec. 1985.

[12] Max J. Egenhofer. Spatial SQL: A Query and Presentation Language . *IEEE Trans. Knowledge & Data Eng.*, 6(1):86–95, Feb. 1994.

[13] E. Allen Emerson. Temporal and Modal Logic . In *Handbood of Theoretical Computer Science*, pages 997–1071. Elsevier Science, 1990.

[14] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically Discovering Likely Program Invariants to Support Program Evolution. *IEEE Trans. Software Eng.*, 27(2):1–25, Feb. 2001.

[15] Shashi K. Gadia and Vimal Chopra. A Relational Model and SQL-Like Query Language for Spatial Databases. In Nabil R. Adam and Bharat K. Bhargava, editors, *Advanced Database Systems*. Springer-Verlag. Berlin, 1993.

[16] Bernhard Ganter and Klaus Reuter. Finding All Closed Sets: A General Approach. *Order*, 8(3):283–290, 1991.

[17] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis - Mathematical Foundations*. Springer Verlag, Heidelberg, 1999.

[18] Robert Godin and Rokia Missaoui. An Incremental Concept Formation Approach for Learning from Databases. In *Theoretical Comp. Sci.*, volume 133, pages 387–419, 1994.

[19] M. Hardwick and D. Spooner. ROSE Data Manager: Using Object Technology to Support Interactive Engineering Applications . *IEEE Trans. Knowledge & Data Eng.*, 1(2), June 1989.

[20] Scott E. Hudson and Roger King. Cactis: A Self-Adaptaive, Concurrent Implementation of an Object-Oriented Database Management System . *ACM Trans. on Database Sys.*, 14(3):291–321, Sept. 1989.

[21] Robert E. Jamison and John L. Pfaltz. Closure Spaces that are not Uniquely Generated. *Discrete Appl Math.*, 147:69–79, Feb. 2005. also in Ordinal and Symbolic Data Analysis, OSDA 2000, Brussels, Belgium July 2000.

[22] Ralph Kopperman and John Pfaltz. Jordan Surfaces in Discrete Topologies . In Reinhard Klette and Jovisa Zunic, editors, *IWCIA, 10th Intern'l Workshop on Combinatorial Image Analysis*, volume Springer Verlag LNCS # 3322, pages 334–350, Aukland, NZ, Dec. 2004.

[23] Charles Lamb, Gordon Landis, Jack Orenstein, and Dan Weinreb. The Objectstore Database System . *Comm. ACM*, 34(10):50–63, Oct. 1991.

[24] Christophe Lecluse, Phillippe Richard, and Fernando Velez. O_2, an Object-Oriented Data Model . In *SIGMOD Conf.*, pages 424–433, Chicago, IL, June 1988.

[25] Douglas B. Lenat and G. Harris. Designing a Rule System That Searches for Scientific Discoveries . In D. A. Waterman and Frederick Hayes-Roth, editors, *Pattern-Directed Inference Systems*. Academic, 1978.

[26] G. H. Lincoff. *The Audubon Society Field Guide to North American Mushrooms*. Alfred A. Knopf, New York, 1981.

[27] Christian Lindig and Gregor Snelting. Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis. In *Proc. of 1997 International Conf. on Software Engineering*, pages 349–359, Boston, MA, May 1997.

[28] J. Liu and M. C. Desmarais. A Method of Learning Implication Networks from Empirical Data: Algorithm and Monte-Carlo Simulation-Based Validation . *IEEE Trans. on Knowledge and Data Eng.*, 9(6):990–1004, Nov. 1997.

[29] Jack Minker. *Logic-Based Artifical Intelligence*. Kluwer Academic, 2001.

[30] Oystein Ore. Galois Connexions. *Trans. of AMS*, 55:493–513, 1944.

[31] John L. Pfaltz. A Functional Approach to Scientific Database Implementation. In *6th Intern'l Working Conf. on Scientific and Statistical Database Management*, pages 132–143, Ascona, Switzerland, Jun. 1992.

[32] John L. Pfaltz. Closure Lattices. *Discrete Mathematics*, 154:217–236, 1996.

[33] John L. Pfaltz. Incremental Transformations of Lattices: A key to Effective Knowledge Discovery. In A. Corradini, H. Ehrig, H-J Kreowsik, and G. Rozenberg, editors, *First Intern. Conf. on Graph Transformation*, volume Springer Verlag LNCS # 2505, pages 351–362, Barcelona, Spain, Oct. 2002.

[34] John L. Pfaltz. Using Concept Lattices to Uncover Causal Dependencies in Software. In B. Ganter and L. Kwuida, editors, *Proc. Int. Conf. on Formal Concept Analysis, Springer LNAI #3874*, pages 233–247, Dresden, Feb. 2006.

[35] John L. Pfaltz and James C. French. Implementing Subscripted Identifiers in Scientific Databases. In Z. Michalewicz, editor, *Statistical and Scientific Database Management*, Lecture Notes in Computer Science, #420, pages 80–91. Springer-Verlag, Berlin-Heidelberg-New York, Apr. 1990.

[36] John L. Pfaltz and James C. French. Scientific Database Management with ADAMS. *Data Engineering*, 16(1):14–18, Mar. 1993.

[37] John L. Pfaltz and Christopher M. Taylor. Closed Set Mining of Biological Data. In *BIOKDD 2002, 2nd Workshop on Data Mining in Bioinformatics*, pages 43–48, Edmonton, Alberta, July 2002.

[38] John L. Pfaltz and Josef Šlapal. Transformations of Discrete Closure Systems. *Theoretical Comp. Sci.*, 2007. (in review).

[39] Young-Gook Ra and Elke A. Rundensteiner. A Transparent Schema-Evolution System Based on Object-Oriented View Technology . *IEEE Trans. Knowledge & Data Eng.*, 9(4):600–624, July 1997.

[40] John F. Roddick. SQL/SE - A Query Language Extension for Databases Supporting Schema Evolution. *SIGMOD Record*, 21(3):10–16, Sept. 1992.

[41] Betty Salzberg and David B. Lomet. Spatial Database Access Methods. *SIGMOD Record*, 20(3), Sept. 1991.

[42] Betty Salzberg and Vassilis J. Tsotras. Comparison of Access Methods for Time-Evolving Data. *ACM Computer Surveys*, 31(2):158–221, June 1999.

[43] Markus Schneider and Thomas Behr. Topological Relationships Between Complex Spatial Objects. *ACM Trans. on Database Sys.*, 31(1):39–81, Mar. 2006.

[44] Padhraic Smyth and Rodney M. Goodman. An Information Theoretic Approach to Rule Induction from Databases. *IEEE Trans on Knowledge and Data Eng.*, 4(4):301–316, Aug. 1992.

[45] Richard T. Snodgrass and et al. TSQL2 Language Specification. *SIGMOD Record*, 23(1):65–86, Mar. 1994.

[46] Petko Valtchev, Rokia Missaoui, and Robert Godin. A Framework for Incremental Generation of Frequent Closed Itemsets. In Peter Hammer, editor, *Workshop on Discrete Mathematics & Data Mining, 2nd SIAM Conf. on Data Mining*, pages 75–86, Arlington, VA, April 2002.

[47] Jinlin Yang, David Evans, Deepali Bhardwaj, Thirumalesh Bhat, and Manuvir Das. Terracotta: Mining Temporal API Rules from Imperfect Traces. In *28th Internl. Conf. on Software Engineering (ICSE 2006)*, Shanghai, China, May 2006.