Establishing Logical Rules from Empirical Data

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

E-mail: `jlp@virginia.edu`

Abstract

We review a method of generating logical rules, or axioms, from empirical data. This method, using closed set properties of formal concept analysis, has been previously described and tested on rather large sets of deterministic data.

The contribution of this paper is a completely new extension of this method to create implications involving numeric inequalities.

1 Rule Based Systems

Russell and Norvig assert that "the representation of knowledge, and reasoning processes that bring knowledge to life, are central to the entire field of artificial intelligence" (p. 194 [22]). Reasoning in turn is based on logical implication which enables both forward and backward chaining techniques. In this paper we introduce a somewhat unusual representation of both existential knowledge and reasoning rules, that is logical implications, about a closed world of which we have gained some knowledge.

We assume a classical, deterministic world in which inference steps, such as $modus\ ponens$, are valid.[1] Our system does not actually make inferences, but rather simply creates a set of consistent axioms or rules or implications that can be subsequently used by logic programs [10] employing Prolog [6, 7]. Nor do we support multiple models, as can be the case with stable set logic [17] or answer set programs [3] using, for example A-Prolog [12]. There is only one model — that of which we currently have partial knowledge. Our understanding of this world consists of only that which we have empirically observed, without any $apriori$ axioms. Of course, we may have to change our operative rules as our knowledge of the world expands — that is the nature of learning. We demonstrate how this occurs in Section 3.

By a rule R_k, we mean a logical implication which we denote by $P \rightarrow C$. That is, if the precedent P is true, then the consequent C must be true. (Some logicians prefer the symbolism $P \supset C$, suggesting that the truth of C is contained within the truth of P.) In this paper we will be using a simple first-order predicate logic and we will develop implication in disjunctive normal form, that is, P can have the form $P \equiv (p_1 \wedge \ldots \wedge p_m) \vee (p_{m+1} \wedge \ldots \wedge p_n)$, while C will always be a conjunction of predicates. We will normally denote conjunction (and) by juxtaposition, as in $p_1 p_2 p_3 p_4$, unless we want to deliberately emphasize the conjunctive aspect.

Given this preamble, our goal is to create a rule-based world view from the kinds of existential data found in a typical relational database such as Figure 1. Here we have 8 objects, or observations,

		P								
		a	b	c	d	e	f	g	h	i
O	1	×	×					×		
	2	×	×					×	×	
	3	×	×	×				×	×	
	4	×		×				×	×	×
	5	×	×		×		×			
	6	×	×	×	×		×			
	7	×		×	×	×				
	8	×		×	×		×			

Figure 1. A relation R_1 composed of 8 objects and 9 predicates.

[1] It is evident that in many real life situations knowledge may be probabilistic. In such a world, inference by Bayesian-like rules is more appropriate. But, we are not prepared to engage in such a world and deliberately ignore it.

each of which may, or may not, exhibit 9 different properties, or predicates, $a, \ldots, i$. Each row is an existential assertion, and we can interpret the first row as $a(o_1) \wedge b(o_1) \wedge g(o_1)$ or just $abg(o_1)$. By the rule-based world view we mean *all* the possible logical implications, or rules R_k, for which the existential world described by the database can be a model. Thus the knowledge set $\mathcal{K} = \{R_1 \ldots R_k\}$ must be consistent.

2 Logical Implication from Empirical Observations

The notion of "closure" plays a major role in our representation of the real world. In particular we will be concerned with closed sets of objects, closed sets of predicates and closed sets of numbers.

2.1 Closure Concepts

By a "closure system" over a "universe" $\mathbf{U}$, we mean a collection $\mathcal{C}$ of sets $X, Y, \ldots Z \subseteq \mathbf{U}$ satisfying the property that if $X, Y \in \mathcal{C}$ then $X \cap Y \in \mathcal{C}$. The sets of $\mathcal{C}$ are said to be the closed sets of $\mathbf{U}$. Alternatively, one can define a closure operator φ on $\mathbf{U}$ satisfying the following 3 axioms for all X, Y, Z:

$X \subseteq X.\varphi$,

$X \subseteq Y$ implies $X.\varphi \subseteq Y.\varphi$

$X.\varphi.\varphi = X.\varphi$.

(For technical reasons we prefer to use suffix operator notation, so read $X.\varphi$ as X closure.) Readily, a set X is closed, $i.e.$ in $\mathcal{C}$, if $X.\varphi = X$. The equivalence of these two alternative definitions is well known [18], and we will use both in the following sections.

Closure systems can satisfy many other axioms, and those that do give rise to different varieties of mathematical systems. If $(X \cup Y).\varphi = X.\varphi \cup Y.\varphi$ we say φ is a topological closure. If the system satisfies the "exchange axiom", that is if $p, q \notin X.\varphi$ but $q \in (X \cup \{p\}).\varphi$ then $p \in (X \cup \{q\}).\varphi$, then the system can be viewed as a kind of linear algebra, or more generally a "matroid". The Galois closure we will be using in this section satisfies neither of these additional axioms. But later in Section 5, we will be using "antimatroid" closure operators, that is those which satisfiy the "anti-exchange axiom" if $p, q \notin X.\varphi$ and $q \in (X \cup \{p\}).\varphi$ then $p \notin (X \cup \{q\}).\varphi$.

2.2 Galois Closure and Concept Lattices

The approach we will follow is similar to Formal Concept Analysis (or FCA) that was first developed by Rudolf Wille and is best presented in [11]. FCA begins with a relation R between two sets, say a set $\mathcal{O}$ of objects and a set $\mathcal{P}$ of object predicates, or attributes. Using standard relational terminology, each object $o_j \in \mathcal{O}$ can be regarded as a row in R and each predicate $p_k \in \mathcal{P}$ is a column. Each attribute p_k is a binary, logical property, $i.e.$ true or false. A concept C_n is a pair of subsets $C_n = (O_n, P_n)$ where $O_n \subseteq \mathcal{O}$, $P_n \subseteq \mathcal{P}$ with the property T that for every $o_i \in O_n$, every $p_k \in P_n$ is true. Each concept is assumed to be maximal, that is for the set O_n there is no larger subset $P'_n \supset P_n$ satisfying property T, and for P_n there is no larger subset $O'_n \supset O_n$ satisfying T.

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 by containment with respect to the predicate sets P_n.[2] If we start with the relation R_1 of Section 1 we obtain the concept lattice $\mathcal{L}$ shown as Figure 2. Each node is labelled with a closed set

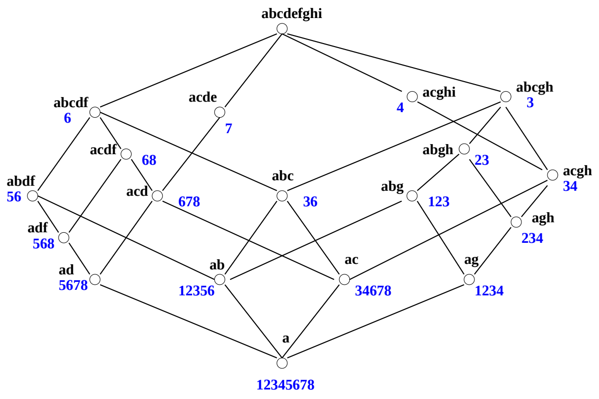


Figure 2. Concept lattice $\mathcal{L}$ corresponding to Relation R_1.

of predicates and a closed set of objects satisfying those predicates, or properties. These two closed sets constitute the concept pair. For example, the combination of properties adf is found in rows, or objects, 5, 6 and 8.

[2]Ganter and Wille [11] prefer to order with respect to object set containment yielding the dual lattice.

In the case of FCA, the closure operator is called the Galois closure between $\mathcal{O}$ and $\mathcal{P}$, and has been well studied [5, 8] In this paper we emphasize the closure aspect as denoted by φ, rather than the concept aspect developed in [11].

2.3 Closed Sets, Generators and Logical Implication

Let C denote a closed set. Then there is some set $A \subseteq C$ such that $A.\varphi = C$. If A is a minimal such set, w.r.t. set inclusion, we call it a generator of C denoted by $C.\gamma$, or by $A \rightarrow C$ [18]. 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 2(b), how does one derive the generating sets?" For example, what is the generator of the closed set $acde$ in Figure 2(a)? In [13] it is shown that "if C covers C_i in a closure lattice[3] then $C.\gamma \cap (C - C_i) \neq \emptyset$." 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 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 $acde.\gamma = \{e\}$, or $e \rightarrow acde$. That this is true is evident from Figure 2(a). Property 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 [21]. The essence of this process is to find at least one element of C which "cannot" be an element/predicate of the covered set C_i. We use this intuition to find numeric generators in Section 5.

3 "Learning" New Rules

We hesitate to use the term "learning" because of its many overtones. But, as we make new observations of our world they may either add to, or contradict, existing rules — our knowledge base must be changed. Here we should note that we assume a static world, but one in which we are constantly making new discoveries. Therefore, it is not equivalent to what has been called "dynamic logic programming" [2] in which the properties associated with a single object can change as a result of agent manipulation.

[3] C covers C_i if we do not have $C_i \subset C' \subset C$, C' closed.

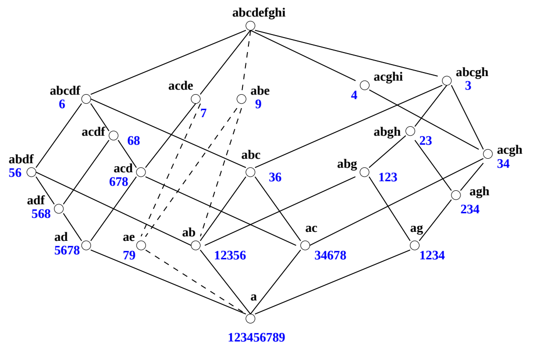
Given an entire relation R, as in Figure 1, there exist algorithms to construct its closed set lattice $\mathcal{L}$; however we find it preferable to construct $\mathcal{L}$ incrementally, one row or observation, at a time. This is certainly similar to real data acquisition.

By the nature of Galois closure each new row of observed 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_k in $\mathcal{L}$. Because of the intersection property of closed set lattices described in Section 2.1, we must now calculate $C \cap C_k$ for each C_k covered by C'. If $C \cap C_k$ already exists in $\mathcal{L}$, nothing more needs be done; otherwise $C \cap C_k$ must itself be recursively entered into $\mathcal{L}$ as well. This can create a recursive cascade of insertions, but as noted in [19] this is fortunately rather rare.

In Figure 3 a new observation, labeled 9, has been entered with the properties a, b and e. The

		P								
		a	b	c	d	e	f	g	h	i
	1	×	×					×		
	2	×	×					×	×	
	3	×	×	×				×	×	
	4	×		×				×	×	×
O	5	×	×		×		×			
	6	×	×	×	×		×			
	7	×		×	×	×				
	8	×		×	×		×			
	9	×	×			×				

(a)



(b)

Figure 3. Modified relation R (a) and Resulting lattice $\mathcal{L}$ (b).

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 3(b) by the dashed lines.

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. This is not unusual in knowledge acquisition. New examples may generate exceptions to a general rule which are often conjunctively explained. Other rules, however, may be simplified as inappropriate predicates are pruned from the premise/generator.

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

4 Two Real Life Examples

Does this approach actually work? This is a reasonable question and the answer is "yes". It does; and we offer the two following working examples to demonstrate proof of concept given binary true/false predicates.

4.1 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" [14]. Figure 4 illustrates representative values of the first 6 (of 22) attributes used to characterize mushrooms in [14]. Each attribute 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 4 yield 42 effective predicates.[4] The lattice generated by the $8,124 \times 42$ relation R consists of 2,641 closed concepts[5]; and, because some concepts have multiple generators, 3,773 distinct implications, or rules.

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

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

```
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 4. The first 6 attributes of the MUSHROOM data set, with nominal values.

To provide some sense of this data set we list in Figure 5 a few of those rules $P \rightarrow C$ in which a singleton predicate P implies `p0`, or poisonous. We 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 been observed, or its support, to the right.

```
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 5. All implications in MUSHROOM with $|P| = 1$ and $\texttt{p0} \in C$.

Are there simple combinations of attributes that also denote poisonous? Figure 6 illustrates those non-trivial conjunctive implications $P \rightarrow C$ for which $|P| = 2$ and $\texttt{p0} \in C$.

A more detailed description of this application can be found in [21].

4.2 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 19th century science. "Causality" implies necessity, or

```
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 6. Rules with two predicate precedents that denote poisonous mushrooms.

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 [4, 15]. They used closed concepts to reveal significant clusters of code modules. Our interest instead has been to uncover "likely" causal dependencies between such modules.[6] To do this we examined 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 7.) The first step is to find which events logically imply other events, that is, which events e_i, if they occur in a trace 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 4.1.

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 each $e_k \in C$ has been preceded in the trace by every $e_i \in P$.[7] Each trace can be treated as a temporally ordered set. We are then looking for events that temporally dominate other events. The logic is simplified if we create a single $n \times n$ boolean array `cant_dominate`, where $(e_k, e_i) \in$ `cant_dominate` if in any trace the first occurrence of e_i precedes any e_k. This `cant_dominate` relation provides a filter that we use to reduce the set of logical implications to a set of likely causal dependencies.

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 [25]. 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 5. These we ran against the `cant_dominate` filter yielding 43 likely causal dependencies. An even smaller subset of 17 of these is shown in Figure 7. (From what we know of the

```
        support
concept 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 7. Some likely causal dependencies.

$JBoss$ system, without having the actual source code, these all seem to be true dependencies.) More details can be found in [20].

Although the creation of closed set lattices, as described in section 2.3 and 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. In both cases there has been a clear "value added" by providing logi-

[6]Michael Ernst coined the term "likely" in his search for code invariants [9]. It seems extremely appropriate here as well.

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

cal rules that can provide the input for a subsequent logical inference engine. $c.f.$ [16].

5 Finding Implications with Numeric Predicates

Formal Concept Analysis, as well as our development to date, has been focused on binary predicates. Either a predicate p is true for a given object/observation, or it is not. However, much of our understanding of the natural world is numeric. We count and we measure.

Both Alberti $et\ al.$ [1] and Simons $et\ al.$ [23], find it necessary to employ mathematical inequalities to implement effective constraint logic programming. The approach that we describe in this section is based on inequalities, as well. Q^2 learning is more ambitious. It seeks to discover quantitative functional relationships between its variables; but the qualitative aspect (its second "Q") is also couched in rules of inequalities [24].

It is perhaps worth noting that mathematical functions constitute a compressed and highly effective notation for expressing a family of implications of the form "if x equals ..., then y equals ...". But to discover such mathematical functions, Q^2 and most numerical data modelling procedures presume knowledge of the independent (generating) and dependent variables, together with an assumed dependence structure, such as linear. Our approach does not.

Every boolean algebra, or lattice $\mathcal{L}$, is a closure system because $x, y \in \mathcal{L}$ implies $x \wedge y \in \mathcal{L}$. In particular, any predicate $p = 0$ or 1 is a trivial boolean algebra, or closure system $\mathcal{C}_p$. Each element, 0 or 1, is its own generator. We achieve far greater expressive power if we let each predicate p_k of an observational tuple $(p_1, \ldots, p_n)$ be the generator of a closed set in a closure system $\mathcal{C}_k$, where $\mathcal{C}_k$ can be more complex than just $\{0, 1\}$. If each closure system $\mathcal{C}_k$ is antimatroid, as defined in Section 2.1, then every closed set has a unique generator [18]. Consequently, if the tuple of predicates $(p_1 \ldots, p_n)$ is a tuple of unique generators, then $(p_1 \ldots, p_n)$ denotes a unique closed set in the n-fold direct product $\mathcal{C}_1 \ldots, \mathcal{C}_n$.

What are some examples of these generating predicates? Consider the $\leq$ operator on an ordered, or partially ordered, set. In [18], it is shown that a collection of principal ideals, that is sets of the form $\{y : y \leq z\}$, is a prototypical antimatroid closure system with z the unique generator of the set. We use this "down set" closure in the following development.

It will be easier to understand the theory we are about to develop if we first consider a concrete example. Figure 8 has been shamelessly copied from ([11], p.44). It summarizes the ratings of 14 monuments on the $Forum\ Romanum$ by different travel guides. Here, B = Baedecker, G = Les Guides Bleus, M = Michelin and P = Polyglott.

		B	G	M	P
a	Arch of Septimus Severus	1	1	2	1
b	Arch of Titus	1	2	2	
c	Basilica Julia			1	
d	Basilica of Maxentius	1			
e	Phocas column		1	2	
f	Curia				1
g	House of the Vestals			1	
h	Portico of Twelve Gods		1	1	1
i	Tempel of Antonius and Fausta	1	1	3	1
j	Temple of Castor and Pollux	1	2	3	1
k	Temple of Romulus		1		
l	Temple of Saturn			2	1
m	Temple of Vespasian			2	
n	Temple of Vesta		2	2	1

Figure 8. Ratings of Roman monuments by 4 guide books.

The 14 numeric 4-tuples can be partially ordered in the usual way, that is $(x_1, x_2, x_3, x_4) \leq (y_1, y_2, y_3, y_4)$ if and only if $x_k \leq y_k, k = 1, \ldots 4$. Figure 9, in which individual tuples have been prefixed with a letter denoting the monument giving rise to that tuple, illustrates this ordering. In order to make it a closure lattice, whenever two tuples are covered by a common tuple, their "intersection tuple" has been entered into the order, as mandated by the incremental growth algorithm of Section 3. These intersection tuples have been underlined for emphasis.

Consider the tuple $n : (0, 2, 2, 1)$ which covers the tuples $(0, 1, 2, 1)$ and $(0, 2, 2, 0)$ in $\mathcal{L}$. Recall from Section 2.3 that the generators of a set are determined by the sets it covers. We claim that the generators of $n : (0, 2, 2, 1)$ are $G > 1 \wedge P > 0$. Moreover, we claim that the implication embodied by this closed set is $(G > 1 \wedge P > 0) \rightarrow (M \geq 2 \wedge B \geq 0)$. First, verify that in Figure 8 this implication is in fact true, with j and n being confirming instances.

The principle behind this derivation, which was sketched in Section 2.3 and rigorously proven for sets in [13], is to show "what constitutes the difference between this closed set and the closed sets it

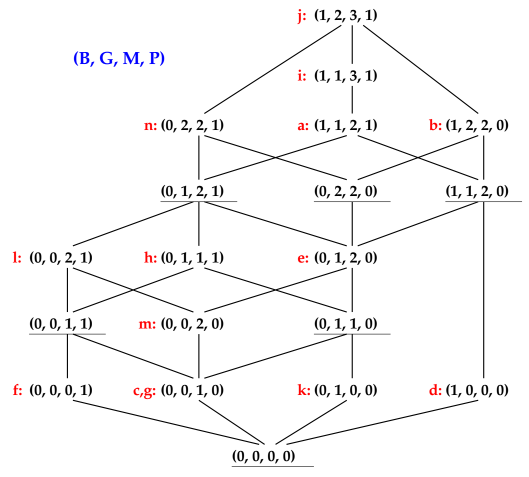


Figure 9. Lattice of closed sets implicit in Figure 8.

covers”. In our case, the only difference between $(0, 2, 2, 1)$ and $(0, 1, 2, 1)$ is that “G is not $\leq$ 1”, or equivalently “G is $>$ 1”. Similarly, $(0, 2, 2, 1)$ is not in the closed set $(0, 2, 2, 0$ because P is not ≤ 0, that is $P > 0$. The conclusion, $i.e.$ those predicates with no differences, could be $B = 0$ and $M = 2$. But, what we really know is that $\neg(B < 0)$ and $\neg(M < 2)$; so $B \geq 0$ and $M \geq 2$ represent the best inferences, or $(G > 1 \wedge P > 0) \rightarrow (B \geq 0 \wedge M \geq 3)$. The adjacent tuple in Figure 9 is $a : (1, 1, 2, 1)$ which covers the tuples $(0, 1, 2, 1)$ and $(1, 1, 2, 0)$. Thus the same reasoning yields $(\neg(B \leq 0) \wedge \neg(P \leq 0) \rightarrow \neg(G < 1) \wedge \neg(M < 2)$ or equivalently $(B > 0 \wedge P > 0) \rightarrow (G \geq 1 \wedge M \geq 2)$ which is supported by observations a, i and j.

Figure 10 provides a rather typical relationship between numeric data that is a bit more complex than Figure 8. We will again use downset closure, $\leq$, to order both the x and y values.

The resultant closure lattice is shown in Figure 11. Using the same logic that we have described above, we can assert that $y > 3 \rightarrow x \geq 5$, based on node c and that $x > 10 \rightarrow y \geq 5$, based on node i, among many others. These can be verified in Figure 10. We have begun to reason about the relationship between x and y.

6 Conclusion

Using the properties of closed sets to establish logical axioms for inference systems has great promise. The two examples of Section 4 demonstrate that we can automatically extract valid rules

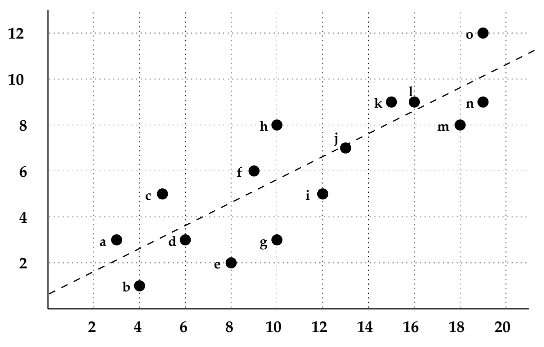


Figure 10. A semi-random distribution of points.

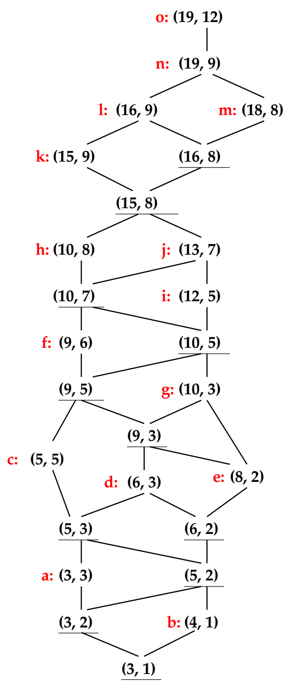


Figure 11. The closure lattice corresponding to Figure 10.

from rather large sets of data. The extraction of logical implications where some of the predicates are mathematical inequalities is, we believe, completely new. Unfortunately, we have not yet automated it in a software system because there are subtiles we do not fully understand. Instead, we are just beginning to explore its potential, which we believe is considerable.

Our goal is to begin generating implications of the form “$if\ [p_1 \wedge (x > 7)] \vee [p_2 \wedge p_3 \wedge (y < 9)]\ then\ \ldots$”, We know, for example, that gene expression is dependent not only on the boolean presence of specific genes, but also on the stage (time) of the replication and degree of separation of gene sequences. These latter factors are best expressed as numeric inequalities. Similarly, evidence based diagnostic systems typically consider both the boolean presence of symptoms and the bounded behaviour of specific variables, $.e.g.$ cholesterol > 180.

The approach of this paper can facilitate the creation of, or refinement of, a variety of intelligent computer systems.

References

[1] Marco Alberti, Marco Gavanelli, Evelina Lamma, Federico Chesani, Paola Mello, , and Paolo Torroni. Compliance Verification of Agent Interaction: A Logic-based Software Tool. *Applied Artif. Intel.*, 20:133–157, 2006.

[2] José Júlio Alferes, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. LUPS — A language for updating logic programs. *Artificial Intel.*, 138:87–116, 2002.

[3] Marcello Balduccini, Michael Gelfond, and Monica Nogueira. Answer set based design of knowledge systems. *Ann. Math. Artif. Intel.*, 47:183–219, Sept. 2006.

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

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

[6] W.F. Clocksin and C.S. Mellish. *Programming in Prolog* . Springer Verlag, New York, NY, 1981.

[7] Alain Colmerauer. An Introduction to Prolog III . *Comm. ACM*, 33(7):69–90, July 1990.

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

[9] 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.

[10] Alan Fern and Robert Givan. Sequential infererence with reliable observations; Learning to construct force-dynamic models. *Artifical Intelligence*, 170:1081–1100, Sept. 2006.

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

[12] Michael Gelfond and Nicola Leone. Logic programming and knowledge representation — The A-Prolog perspective. *Artif. Intell.*, 138:3–38, 2002.

[13] 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.

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

[15] 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.

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

[17] Ilkka Niemelä. Logic program with stable model semantics as a constraint programming paradigm. *Annals of Math. Artif. Intell.*, 25:241–273, 1999.

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

[19] John L. Pfaltz. Transformations of Concept Graphs: An Approach to Empirical Induction . In *2nd International Workshop on Graph Transformation and Visual Modeling Techniques*, pages 320–326, Crete, Greece, July 2001. Satellite Workshop of ICALP 2001.

[20] 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.

[21] 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.

[22] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[23] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138:181–234, 2002.

[24] Daniel Vladušič, D. Šuc, I. Bratko, and W. Rulka. Q^2 Learning and its Application to Car Modeling. *Applied Artifical Intelligence*, 20:675–701, 2006.

[25] 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.