Efficiently computing a linear extension of the sub-hierarchy of a concept lattice

Anne Berry[1], Marianne Huchard[2], Ross M. McConnell[3], Alain Sigayret[1], and Jeremy P. Spinrad[4]

[1] LIMOS (Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes), CNRS UMR 6158, Université Clermont-Ferrand II, Ensemble scientifique des Cézeaux, 63177 Aubière Cedex, France. berry@isima.fr, sigayret@isima.fr

[2] LIRMM (Laboratoire d'Informatique de Robotique et de Micro-électronnique de Montpellier), CNRS UMR 5506, Université Montpellier II, 161 rue Ada, 34392 Montpellier cedex 5, France.
huchard@lirmm.fr

[3] EECS Department, Vanderbilt University, Nashville, TN 37235 USA.
spin@vuse.vanderbilt.edu

[4] Computer Science Department, Colorado State University, Fort Collins, CO 80523-1873 USA. rmm@cs.colostate.edu

Abstract. Galois sub-hierarchies have been introduced as an interesting polynomial-size sub-order of a concept lattice, with useful applications. We present an algorithm which, given a context, efficiently computes an ordered partition which corresponds to a linear extension of this sub-hierarchy.

1 Introduction

Formal Concept Analysis (FCA) aims at mining concepts in a set of entities described by properties, with many applications in a broad spectrum of research fields including knowledge representation, data mining, machine learning, software engineering or databases. Concepts are organized in concept (Galois) lattices where the partial order emphasizes the degree of generalization of concepts and helps to visually apprehend sets of shared properties as well as groups of objects which have similarities.

The main drawback of concept lattices is that the number of concepts may be very large, or even exponential in the size of the relation.

One of the options for dealing with this problem is to use a polynomial-size representation of the lattice which preserves the most pertinent information. A way of doing this is to restrict the lattice to the concepts which introduce a new object or property, leading to two similar structures called the 'Galois sub-hierarchy' (GSH) and the 'Attribute Object Concept poset' (AOC-poset). GSH has been introduced in the software engineering field by Godin and Mili in 1993 ([11]) for class hierarchy reconstruction and successfully applied in later research works ([12, 20, 14, 6]).

Recent work has shown interest of GSH in an extension of FCA to Relational Concept Analysis (RCA); RCA has been tested to identify abstractions in UML (Unified Modeling Language, see [19]) diagrams allowing to improve such diagrams in a way that had not been explored before ([7]). AOC-poset has been used in applications of FCA to non-monotonic reasoning and domain theory ([13]) and to produce classifications from linguistic data ([17, 16]). Considering AOC-poset or GSH is interesting from two points of view, namely the algorithmic and the conceptual (human perception), because the structure which is used has only a restricted number of elements.

Several algorithms have been proposed to construct the Galois sub-hierarchy, either incrementally or globally. Incremental algorithm Ares [8] and IsGood [10] add a new object given with its property set in an already constructed GSH. The best worst-case complexity is in $O(k^3\, n^2)$ for Ares (in $O(k^4\, n^2)$ for IsGood) where k is the maximal size of a property set and n is the number of elements in the initial GSH. Note that best pratical results are nevertheless obtained by IsGood. The global algorithm Ceres ([15]) computes the elements of the GSH as well as the order and has worst case complexity in $O(|O|\,(|O| + |P|)^2)$.

In this paper, we present an algorithm which outputs the elements of the GSH in a special order, compatible with a linear extension of the GSH: roughly speaking, we decompose each element of the GSH into an extent (which is the set of objects of this element) and an intent (which is the set of properties of this element), and we output a list of subsets of objects and properties, such that if E_1 is a predecessor of E_2 in the GSH, then both the intent and the extent of E_1 are listed before the intent and extent of E_2 in our output ordering.

To do this, we use a partition refinement technique, inspired by work done in Graph Theory, which can easily be implemented to run in linear time. We used this in previous works to improve Bordat's concept generation algorithm (see [4]), in order to rapidly group together the objects (or, dually, the properties) which are similar. Partition refinement has also been used to reorder a matrix ([18]).

One of the interesting new points of the algorithm presented here is that it uses the objects and properties at the same time, instead of just the objects or just the properties.

The other interesting development is that the orderings on the properties and objects created by our algorithm define a new representation of the input relation, essentially by re-ordering its rows and columns, creating a zone of zeroes in its lower right-hand corner.

The paper is organized as follows: in Section 2, we give a few necessary notations, and present a running example which we will use throughout the paper to illustrate our work. Section 3 presents some results from previous papers, and explains the general algorithmic process which is used. Section 4 gives the algorithm, as well as some interesting properties of the output. The algorithm is proved in Section 5.

2 Notations and Example

It is assumed that the reader is familiar with classical notions of partial orderings and lattices, and is referred to [5], [1] and [9]. We will need a few preliminary notations and definitions.

Given a context $(\mathcal{O}, \mathcal{P}, R)$, $\mathcal{O}$ is a set of objects and $\mathcal{P}$ a set of properties, for $X \subseteq \mathcal{O}$, $Y \subseteq \mathcal{P}$, we will denote by $R(X, Y)$ the subrelation $R \cap (X \times Y)$. In the algorithms we present in this paper, objects and properties can be interchanged, so we will need to unify notations x' and x'' from [9] into $R[x]$ as follows: we will denote by $R[x]$ the set $\{y \in \mathcal{P},\ (x, y) \in R\}$ if $x \in \mathcal{O}$ and $\{y \in \mathcal{O},\ (y, x) \in R\}$ if $x \in \mathcal{P}$. $\bar{R}$ will denote the complement of relation R: $(x, y) \in \bar{R}$ iff $(x, y) \notin R$.

We will say that a concept $A' \times B'$ is a **successor** of concept $A \times B$ if $A \subset A'$ and there is no intermediate concept $A'' \times B''$ such that $A \subset A'' \subset A'$. A concept $A' \times B'$ is a **descendant** of concept $A \times B$ if $A \subset A'$. The notions of **predecessor** and **ancestor** are defined dually.

In the rest of this paper, we will use the same running example to illustrate our definitions and algorithms.

Example 1. Binary relation R:

Set of objects:
$\mathcal{O} = \{1, 2, 3, 4, 5, 6\}$,

Set of properties:
$\mathcal{P} = \{a, b, c, d, e, f, g, h\}$.

	a	b	c	d	e	f	g	h
1		×	×	×	×			
2	×	×	×				×	×
3	×	×				×	×	×
4				×	×			
5			×	×				
6	×							×

The associated concept lattice $\mathcal{L}(R)$ is shown in Figure 1.

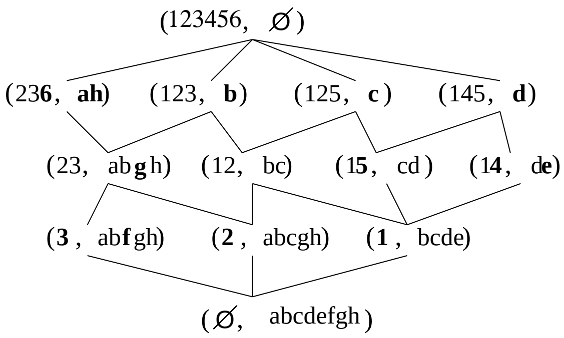

Fig. 1. Concept lattice $\mathcal{L}(R)$ of Example 1. For example, $(12, bc)$ denotes concept $\{1, 2\} \times \{b, c\}$.

Definition 1. *An object (resp. a property) x is said to be* introduced *by a concept if x is in the extent (resp. intent) of this concept and no ancestor (resp. descendant) of this concept contains x in its extent (resp. intent). An element of the lattice is said to be an* **introducer** *if it introduces a property or an object.*

In [9], an introducer of an object is called an 'Object Concept' and an introducer of a property is called a 'Property Concept'.

Definition 2. *([11]) The* **Galois sub-hierarchy** *(GSH) of a concept lattice is the partially ordered set of elements $X \times Y$, $X \cup Y \neq \emptyset$, such that there exists a concept where X is the set of objects introduced by this concept and Y is the set of properties introduced by this concept. The ordering of the elements in the GSH is the same as in the lattice.*

Let E_1 and E_2 be two elements of the GSH. We will denote $E_1 < E_2$ if E_1 is an ancestor (i.e. represented below in the GSH) of E_2 and $E_1 \leq E_2$ if $E_1 = E_2$ or $E_1 < E_2$.

The **Attribute Object Concept Poset** (AOC-Poset) is a GSH with addition of the top and bottom elements of the lattice, if they are not present in the GSH ([16]).

Example 2.
For example, $23 \times abgh$ is the introducer of g, as the descendants of this concept: $236 \times ah$, $123 \times b$ and $123456 \times \emptyset$ do not have g in their intent.

Simplifying the labeling of Figure 1 to the introduced objects and properties leads to the lattice represented in Figure 2. Then, removing trivial nodes and nodes labeled with empty sets leads to the Galois sub-hierarchy represented in Figure 3.

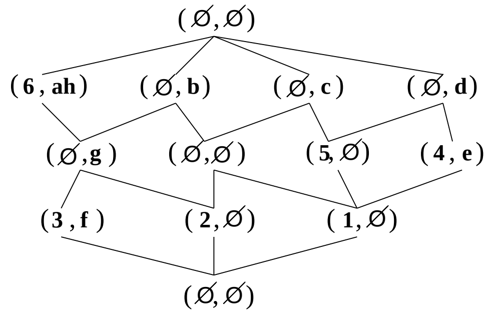

Fig. 2. Simplified labelling of Lattice of Figure 1, first step to generate Galois sub-hierarchy of Figure 3.

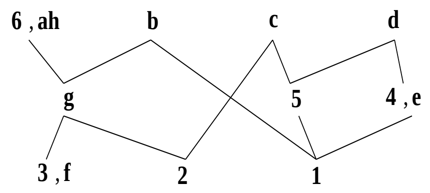

Fig. 3. Galois sub-hierarchy of Example 1, with the simplified labeling: b, for example, corresponds to $(\emptyset, \{b\})$. This hierarchy is obtained by removing trivial nodes from simplified lattice of Figure 2.

3 Previous results and algorithmic process

In order to explain our algorithm, we will need some extra notions which we introduced in previous papers to improve concept-related algorithms.

First, we will need the notion of *maxmod*, which is a set of objects or properties which share the same row or column. These are useful, because when generating concepts using Bordat's algorithm for example, the elements of a maxmod of the relation or subrelation which is used are always grouped together in the concepts. Note that if the relation is reduced ([1]), all maxmods are of cardinality one; however, reducing a relations costs $O(|\mathcal{O}+\mathcal{P}|^3)$ time, whereas our algorithm is in $O(|\mathcal{O}+\mathcal{P}|^2)$, so we cannot assume that the relation is reduced.

Definition 3. *([2]) Let $(\mathcal{O}, \mathcal{P}, R)$ be a context. For $x, y \in \mathcal{O}$ or $x, y \in \mathcal{P}$, x and y belong to the same* **maxmod** *if $R[x] = R[y]$. We refer to maxmods as* object maxmods *or* property maxmods, *according to the nature of their elements.*

The maxmods define a partition of $\mathcal{O}+\mathcal{P}$. The GSH, with its simplified labeling, contains exactly the maxmods of $\mathcal{O}+\mathcal{P}$. For any maxmod M, we will denote $E(M)$ the element of the GSH which contains M in its label.

Example 3. For example, $\{a, h\}$ defines a maxmod, as a and h have identical columns in the matrix of R. a and h always appear together, both in the lattice and in the GSH. The set of maxmods is:
$\{\{a, h\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}\}$.
These are exactly the labels of the GSH of Figure 1.

Another important notion, which stems from Graph Theory, is that of *domination*:

Definition 4. *([2]) Let $(\mathcal{O}, \mathcal{P}, R)$ be a context. For $x, y \in \mathcal{O}$ or $x, y \in \mathcal{P}$, x is said to* **dominate** *y if $R[x] \subseteq R[y]$. A maxmod X is said to dominate another maxmod Y if for $x \in X$, for $y \in Y$, x dominates y.*

In the rest of this paper, we will often use maxmods, and denote them informally without columns, for example we will use ah instead of $\{a, h\}$; matrices will be represented with maxmods instead of lone objects or properties, which is equivalent to keeping in the matrix only one representative for every set of lines which are identical.

The notion of domination leads us to a decomposition of the GSH into an object-GSH and a property-GSH, as introduced in [3]. We will use the following simple terms in referring to this:

Definition 5. *([3]) Let $(\mathcal{O}, \mathcal{P}, R)$ be a context. The partial order on the property maxmods is called the* property domination order*, and likewise the partial order on the object maxmods is called the* object domination order*.*

Note that both partial orders on objects and properties are compatible with the GSH:

Theorem 1. *([3]) Let P_1 and P_2 be two property maxmods ; then P_1 dominates P_2 iff $E(P_1) < E(P_2)$. Let O_1 and O_2 be two object maxmods, then O_1 dominates O_2 iff $E(O_2) < E(O_1)$.*

Example 4. Figure 4 shows the domination orders on our example.

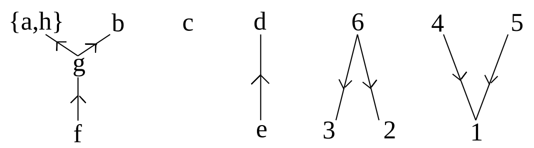

Fig. 4. Domination order of Example 1. For example, we see that e dominates d and f dominates b, whereas 6 dominates 2.

We showed in [4] that a linear extension of the object or property ordering can be very efficiently computed in linear time, using another Graph Theory tool: partition refinement. The partition algorithm is given in Section 4.

Our aim here is to compute a linear extension of the GSH. Since the object and property orderings are preserved in the GSH, both the corresponding linear extensions are compatible with the GSH. Our idea here is to use the object linear extension and the property linear extension and merge them into a linear extension of the GSH.

4 The algorithm

We present an algorithm which takes a context as input, and outputs a linear extension of the GSH.

Our algorithm uses as a primitive a partition refinement technique (based on a process presented in [4]), called Algorithm MAXMOD-PARTITION, which computes a linear extension of the domination order. We use this primitive twice in our main algorithm (Algorithm Tom Thumb): the first time, we use Algorithm MAXMOD-PARTITION with an arbitrary ordering L on $\mathcal{P}$; the second time, we use in the input the ordering on $\mathcal{O}$ output by the first pass of Algorithm MAXMOD-PARTITION.

Algorithm MAXMOD-PARTITION
Input: A context $(\mathcal{O}, \mathcal{P}, R)$, a set S which is either $\mathcal{O}$ or $\mathcal{P}$, and an ordered partition L of $(\mathcal{O} + \mathcal{P}) - S$.
Output: An ordered partition of the maxmods of S.
PART is a queue, initialized with S;
for each class of partition L taken in the input ordering **do**
  choose a representative x of the class;
  for each class K of PART such that $|K| > 1$ **do**
    $K' \leftarrow K \cap R[x]$;
    $K'' \leftarrow K - R[x]$;
    if $K' \neq \emptyset$ and $K'' \neq \emptyset$ **then**
      In PART, replace K by K' followed by K'';
return PART.

Note that the execution can be stopped if all classes trivially contain a single element.
This process has the remarkable property that a given maxmod taken in the output partition can dominate only maxmods which lie to its left:

Theorem 2. *([4]) Algorithm* MAXMOD-PARTITION *outputs an ordered list of maxmods such that if maxmod A dominates maxmod B, then B is before A in this list.*

Example 5. Let us apply Algorithm MAXMOD-PARTITION to our running example, using $S = \mathcal{O}$, L is the total ordering (a, b, c, d, e, f, g, h) of $\mathcal{P}$ (each partition is trivially formed of a single element).

({1,2,3,4,5,6})
  ↓ R[a]={2,3,6}
({2,3,6}; {1,4,5})
  ↓ R[b]={1,2,3}
({2,3}; {6}; {1}; {4,5})
  ↓ R[c]={1,2,5}
({2}; {3}; {6}; {1}; {5}; {4}).

Our algorithm for computing a linear extension of the GSH first uses Algorithm MAXMOD-PARTITION to compute a linear extension of the object maxmods, using any ordering of the property set, then uses the output ordering on object maxmods to find a special linear extension of the property maxmods. In a third step, the algorithm will merge the two extensions. The result is a list LINEXT of maxmods, which represent a linear extension of the GSH, in which for any element of the GSH formed by both an object maxmod and a property maxmod, these appear consecutively.

Algorithm TOM THUMB
Input: A context $(\mathcal{O}, \mathcal{P}, R)$.
Output: A linear extension of the GSH, where object maxmods and property maxmods are separated (if an element of the GSH contains an object maxmod O and a property maxmod P then P and O appear consecutively in the linear extension).

1. Apply MAXMOD-PARTITION to $(\mathcal{O}, \mathcal{P}, R)$, with $S = \mathcal{O}$ and L an arbitrary ordering of $\mathcal{P}$, resulting in an ordered partition $(Y_1, ..., Y_q)$ of **object** maxmods;
2. Apply MAXMOD-PARTITION, with $S = \mathcal{P}$, and using for L partition $(Y_q, ...,$ $Y_1)$, i.e. the partition obtained at step one in reverse order, resulting in an ordered partition $(X_1, ..., X_r)$ of **property** maxmods;
3. LIST is an empty queue.
   j $\leftarrow$ q; i $\leftarrow$ 1;
   while j>0 and i$\leq$r **do**
      choose representatives: $y \in Y_j$ and $x \in X_i$;
      if $(y, x) \in R$ **then** add X_i to queue LIST; i $\leftarrow$ i+1;
      else add Y_j to queue LIST; j $\leftarrow$ j $-$1;
   // At this point, there may remain either objects or properties, which are added to LIST
   while j>0 **do** add Y_j to queue LIST; j $\leftarrow$ j $-$1;
   while i$\leq$r **do** add X_i to queue LIST; i $\leftarrow$ i+1;
   LINEXT $\leftarrow$ LIST in reverse; **Return** LINEXT.

Note that LIST computes a linear extension which is in reverse order with respect to the orientation of the GSH we have chosen for this paper, as illustrated in our running example. LIST thus has to be output in reverse as LINEXT. In the final LINEXT output, the list of properties output by MAXMOD-PARTITION is reversed, while the list of objects is preserved.

Example 6. An execution of Algorithm TOM THUMB on Relation R from Example 1.

Step 1: Partition the object set, using any ordering on the property set. With ordering (a, b, c, d, e, f, g, h) used in Example 5, (2,3,6,1,5,4) is obtained.

Step 2: Partition property set using objects ordering (4,5,1,6,3,2).
({a,b,c,d,e,f,g,h})
$\downarrow$ R[4]={d,e}
({d,e}; {a,b,c,f,g,h})
$\downarrow$ R[5]={c,d}

({d}; {e}; {c,}; {a,b,f,g,h})
↓ R[1]={b,c,d,e}
({d}; {e}; {c,}; {b}; {a,f,g,h})
↓ R[6]={a,h}
({d}; {e}; {c,}; {b}; {a,h}; {f,g})
↓ R[3]={a,b,f,g,h} let the partition unchanged
↓ R[2]={a,b,c,g,h}
({d}; {e}; {c,}; {b}; {a,h}; {g}; {f})

Step 3: Merge ordered partitions (4,5,1,6,3,2) and (d,e,c,b,ah,g,f).
LIST=();
current object: 4, current property: d, $(4, d) \in R$, d is chosen next,
LIST=(d);
current object: 4, current property: e, $(4, e) \in R$, e is chosen next,
LIST=(d; e);
current object: 4, current property: c, $(4, c) \notin R$, 4 is chosen next,
LIST=(d; e; 4);
current object: 5, current property: c, $(5, c) \in R$, c is chosen next,
LIST=(d; e; 4; c);
current object: 5, current property: b, $(5, b) \notin R$, 5 is chosen next,
LIST=(d; e; 4; c; 5);
current object: 1, current property: b, $(1, b) \in R$, b is chosen next,
LIST=(d; e; 4; c; 5; b);
current object: 1, current properties: ah, $(1, a) \notin R$, 1 is chosen next,
LIST=(d; e; 4; c; 5; b; 1);
current object: 6, current properties: ah, $(6, a) \in R$, ah is chosen next,
LIST=(d; e; 4; c; 5; b; 1; ah);
current object: 6, current property: g, $(6, g) \notin R$, 6 is chosen next,
LIST=(d; e; 4; c; 5; b; 1; ah; 6);
current object: 3 current property: g, $(3, g) \in R$, g is chosen next,
LIST=(d; e; 4; c; 5; b; 1; ah ;6 ;g);
current object: 3 current property: f, $(3, f) \in R$, f is chosen next,
LIST=(d; e; 4; c; 5; b; 1; ah; 6; g; f);
no property left, add objects 3 then 2,
LIST=(d; e; 4; c; 5; b; 1; ah; 6; g; f; 3; 2).
The resulting list of maxmods is $(d; e; 4; c; 5; b; 1; ah; 6; g; f; 3; 2)$.
In the GSH of Example 2, ah and 6 are in the same element, and so are e and 4, as well as f and 3.
The output LINEXT represents linear extension
$(\{2\}, \{3, f\}, \{g\}, \{6, ah\}, \{1\}, \{b\}, \{5\}, \{c\}, \{4, e\}, \{d\})$
of the GSH.

Note that, as usual, $\mathcal{O}$ and $\mathcal{P}$ could be interchanged in the algorithms above.

Time Complexity:
Algorithm MAXMOD-PARTITION can be implemented to run in $O(|R|)$ time,

thus Step 1 and Step 2 cost $O(|R|)$. In Step 3, each time we enter the while-loop $i + (q - j)$ is incremented; as $i + (q - j) \leq r + q \leq |\mathcal{P}| + |\mathcal{O}|$, Step 3 costs $O(|\mathcal{P}| + |\mathcal{O}|)$. Algorithm Tom Thumb has then a complexity in $O(|R|)$.

Algorithm Tom Thumb runs in linear time, but does not explicitely compute the elements of the GSH. Computing these elements, using a brute-force approach, costs $O((|\mathcal{P}| + |\mathcal{O}|)^3)$ time, and no better process is known for this. However, in order to compute these elements, we could use the output of Algorithm Tom Thumb in the following fashion: if an element of the GSH has both a non-empty extent and a non-empty intent, call them O and P, then in list LinExt output by the algorithm, P comes first and O is just after P; thus we need to test all pairs of the list where an object maxmod immediately follows a property maxmod, to find out whether they together form an element of the GSH. This test can be performed by taking an element o_1 in O and an element p_1 in P; (O, P) is an element of the GSH iff $R[o_1] \times R[p_1]$ is a rectangle, i.e. the corresponding cartesian product is a subset of R. The test costs $O((|\mathcal{P}| + |\mathcal{O}|)^2)$ time for each pair which is tested. The overall cost of computing all the elements of the GSH may become lower than $O((|\mathcal{P}| + |\mathcal{O}|)^3)$ in some cases.

Algorithm Tom Thumb turns out to have a variety of interesting properties, due to the fact that it computes a very special linear extension of the GSH, as Theorem 3 for which will will define the notion of 'staircase':

Definition 6. *Let $(\mathcal{O}, \mathcal{P}, R)$ be a context. Let $\alpha = (o_1, ..., o_q)$ be a total ordering of $\mathcal{O}$ and $\beta = (p_1, ..., p_r)$ be a total ordering of $\mathcal{P}$. Let M be the matrix of R, with the rows ordered by α and the columns ordered by β.*

We will say that M has a lower-right-hand staircase of zeroes *if there exist an total function φ from an interval $[o_h, o_q]$ of α to $\mathcal{P}$ such that:*

- *for o_i and o_j objects of $[o_h, o_q]$, if $o - i$* before *o_j in α then $\varphi(o_i)$ is* after *$\varphi(o_j)$ in β, and*
- *for each i in $[h, q]$, the rectangle $S_i = \{o_i\} \times [\varphi(o_i), p_r]$ is a rectangle of zeroes (i.e. $\forall y \in [\varphi(o_i), p_r]$, $M[o_i, y] = 0$).*

We will say that the union Z_2 of all these rectangles S_i of zeroes is a lower-right-hand staircase of zeroes *of M. We will denote by Z_1 the other part $M - Z_2$ of M.*

Theorem 3. *Using an arbitrary ordering α of $\mathcal{O}$ to compute with Algorithm* Maxmod-Partition *an ordered partition α of the maxmods of $\mathcal{P}$ (as in Step 2 of the Tom Thumb Algorithm), and reordering the rows and columns of the matrix with α in reverse and β results in a matrix which has a lower-right-hand staircase of zeroes.*

The proof follows the definition of the partition process: at each step, 'ones' are put to the left, and 'zeroes' are left at the right; this is repeated one step higher, to partition the zone above the previous zeroes zone; this results in a staircase of zeroes, in any matrix defined in this fashion.

Proof. Let $(\mathcal{O}, \mathcal{P}, R)$ be a context. Let $\alpha = (o_1, ..., o_q)$ be a total ordering of $\mathcal{O}$ and $\beta = (p_1, ..., p_r)$ be a total ordering of $\mathcal{P}$. Let M be the matrix of R with the rows ordered by α in reverse: $(o_q, ..., o_1)$ and the columns ordered by $\alpha = (p_1, ..., p_r)$.

At the first step of the algorithm, if $\overline{R}[o_1] \neq \emptyset$, set $\mathcal{P}$ will be split into $K' = R[o_1]$ and $K'' = \overline{R}[o_1]$; thus $\forall y \in K''$, $M[o_1, y] = 0$, and K'' is the righmost class of Part at the end of step 1. If $\overline{R}[o_1] = \emptyset$, let $k = 0$; note that this only occurs if $R[o_1] = \mathcal{P}$. Suppose that at the end of step i of the algorihm, the rightmost class of Part is $S_i = \bigcap_{j=1}^{i} \overline{R}[o_i]$. If, at step $i+1$, $\overline{R}[o_{i+1}] \neq \emptyset$, $S_{i+1} = \bigcap_{j=1}^{i+1} \overline{R}[o_k]$ will be the new rightmost class in Part. If $\overline{R}[o_{i+1}] = \emptyset$, let $k = i$.

Thus we recursively construct a list $\{o_1\} \times S_1$, ..., $\{o_k\} \times S_k$ of rectangles of zeroes, with inclusions $S_1 \supseteq ... \supseteq S_k$. This will define the successive elements of the ordered partition on $\mathcal{P}$, which w.l.o.g. are: $R[o_1]$, $R[o_2] \cap S_1$, ... , $R[o_k] \cap S_{k-1}$, S_k.

We can thus define function φ from $[o_1, o_k]$ to $\mathcal{P}$ by $\varphi(o_i) = S_i$ with the different rectangles of zeroes S_i of Definition 6. Thus M has a lower-right-hand staircase of zeroes $Z_2 = \cup_{i=1}^{k} (\{o_i\} \times S_i)$. Note that the use of a reverse ordering on $\mathcal{O}$ makes the object indices different from those of Definition 6.

Example 7. Let us use (2,3,6,1,5,4) and (d,e,c,b,ah,g,f) as output by Steps 1 and 2 of the execution of Algorithm Tom Thumb of Example 6. The resulting matrix is:

R	d	e	c	b	ah	g	f
2			×	×	×	×	
3				×	×	×	×
6					×		
1	×	×	×	×			
5	×		×				
4	×	×					

The lower-right-hand part of the matrix contains only zeroes. The limit of this zone is defined by the succession of queries on R given in Step 3 of the Tom Thumb Algorithm: $(x, y) \notin R$, $(4, c) \notin R$, $(5, b) \notin R$, $(1, a) \notin R$, $(6, g) \notin R$. Thus $Z_2 = \{4\} \times \{c, b, ah, g, f\} \cup \{5\} \times \{b, ah, g, f\} \cup \{1\} \times \{ah, g, f\} \cup \{6\} \times \{g, f\}$.

5 Proof of the algorithm

The Tom Thumb Algorithm does a traversal of the GSH in a such fashion that an element of the GSH is reached – and then put in the list – only after all its descendants in the GSH are reached:

Theorem 4. *Algorithm* Tom Thumb *gives a linear extension of the GSH, where object maxmods and property maxmods are separated: if a property maxmod P and an object maxmod O are in the same element of the GSH then O appears just before P in the linear extension.*

In order to prove this, we will use Theorem 3, as well as the following lemmas.

Lemma 1. *In the course of the execution of Algorithm Tom Thumb,*

1. *when a* property *maxmod P is added to* List*, then for any object maxmod O added after P, (O, P) is in Z_1.*
2. *when an* object *maxmod O is added to* List*, then for any property P added after O, (O, P) is in Z_2.*

Proof. Let $(\mathcal{O}, \mathcal{P}, R)$ be a context. Let $\alpha = (O_1, ..., O_q)$ be the partition in object maxmods obtained at the first step of Algorithm Tom Tumb, let $\beta = (P_1, ..., P_r)$ be the partition in property maxmods obtained at the second step of Algorithm Tom Tumb, using α in reverse: $(O_q, ..., O_1)$, as ordered partition L in the input. Let M be the matrix of R with rows in order α and columns in order β. By Theorem 3, M has a lower-right-hand staircase of zeroes.

1. Let O be an object maxmod and let P be a property maxmod such that O is after P in List. If P and O have been compared in the third step of Algorithm Tom Tumb, then $(O, P) \in R$ and (O, P) may not be in Z_2. On the other hand, if P has been inserted in List without being compared to O, this means there exists another object maxmod O' which is after O in α, which has been compared to P, and which has been put after P as $(O', P) \in R$. Thus $(O', P) \in Z_1$ and, by Definition 6 of M, (O, P) is in Z_1.
2. We will prove by induction that for each object maxmod O_i of α, all the properties P_j put after O_i in List verify $(O_i, P_j) \in Z_2$.
   The first step of Algorithm Tom Thumb begins by comparing O_q and P_1. All the property maxmods of $R[O_q]$ are put before O_q in List, as for $P_j \in R[O_q]$, $(O_q, P_j) \in R$. The property maxmods of $\overline{R}[O_q]$ will be put after O_q in List and will constitute the first 'step' S_1 of the lower-right-hand staircase in matrix M. Thus for all $P_j \in \overline{R}[O_q]$, $(O_1, P_j) \in Z_2$.
   Suppose that for object maxmod O_i, all the property maxmod P_j that are after O_i in List verify $(O_i, P_j) \in Z_2$.
   When O_{i-1} is processed, the set $\mathcal{B}$ of property maxmods which have yet to be processed can be split into $\mathcal{B} \cap R[O_{i-1}]$ and $\mathcal{B} \cap \overline{R}[O_{i-1}]$. Then, the property maxmods of $\mathcal{B} \cap R[O_{i-1}]$ will be put before O_{i-1} in List and these of $\mathcal{B} \cap \overline{R}[O_{i-1}]$ will be put after O_{i-1}. By Definition 6, $O_{i-1} \times (\mathcal{B} \cap \overline{R}[O_{i-1}])$ is an element of the staircase of zeroes of matrix M. Thus, all P_j in $\mathcal{B} \cap \overline{R}[O_{i-1}]$ put after O_{i-1} in List will verify $(O_{i-1}, P_j) \in Z_2$.

Lemma 2. *Let O be an object maxmod associated with element $E(O)$ of the GSH, let P be a property maxmod associated with $E(P)$; then $(O, P) \in R$ iff $E(O) \leq E(P)$.*

Proof.

- $\Rightarrow$ This follows directly from the definitions of concept lattice and GSH: the introducer of P is concept $R[P] \times R[R[P]]$. If $(O, P) \in R$, O is in this concept and in all its predecessors, one of which is the introducer of O. Then $E(O) \leq E(P)$ if $(O, P) \in R$.

$\Leftarrow$ If $E(O) \leq E(P)$, the introducer of P is a descendant of the introducer of O and thus will have O in its extent. Then $(O, P) \in R$.

Theorem 5. *For any pair (O, P) of maxmods such that $(O, P) \notin R$ and $(O, P) \in Z_1$, O and P belong to non-comparable elements $E(O)$ and $E(P)$ of the GSH.*

Proof. Let (O_1, P_1) be a zero in Z_1. Suppose by contradiction that P_1 and O_1 belong to comparable elements of the GSH.

Since $(O_1, P_1) \notin R$, by Lemma 2, we must have $E(P_1) < E(O_1)$. Let us consider the moment when P_1 is added to LIST: by Lemma 1, since $(O_1, P_1) \in Z_1$, O_1 has not yet been added. Let O_2 be the object which is queried by Step 3 of Algorithm Tom Thumb, and which results in adding P_1 to LIST, let $E(O_2)$ be the element of the GSH which contains O_2: we have $(O_2, P_1) \in R$. Clearly, the algorithm will insert O_2 after P_1 in LIST; by Lemma 2, $E(O_2) \leq E(P_1)$.

Combining the above remarks together, we obtain $E(O_2) \leq E(P_1) < E(O_1)$, so by Theorem 1, O_1 dominates O_2. But this is impossible, by Theorem 2, as the ordering used ensures that O_1 can dominate only objects which are output before it by Algorithm MAXMOD-PARTITION; since this ordering is used in reverse by Step 3 of Algorithm Tom Thumb, O_2, which is used first, cannot be dominated by O_1 - a contradiction.

We are now ready to prove Algorithm TOM THUMB:

Proof. (*of Theorem 4*) Let A and B be two different maxmods, belonging to elements $E(A)$ and $E(B)$ of the GSH respectively. There will be two cases:

- Suppose that $E(B) < E(A)$. We will show that B is placed before A in LINEXT, which is equivalent to saying that A is placed before B in LIST.
  1. If A and B are both property maxmods: by Theorem 1, B dominates A. Clearly, the ordering output by Step 2 of Algorithm Tom Thumb is preserved in LIST. By Theorem 2, A is before B in LIST.
  2. If A and B are both object maxmods: by Theorem 1, A dominates B. The ordering output by Step 1 of Algorithm Tom Thumb is reversed in LIST. By Theorem 2, A is before B in LIST.
  3. If A is a property maxmod and B is an object maxmod: as $E(B) < E(A)$, by Lemma 2, $(B, A) \in R$, which implies $(B, A) \notin Z_2$. By contraposition of the second item of Lemma 1, A must be before B in LIST.
  4. If A is an object maxmod and B is a property maxmod: as $E(B) < E(A)$, by contraposition of Lemma 2, $(A, B) \notin R$. By Theorem 5, $(A, B) \notin Z_1$, as $E(A)$ and $E(B)$ are comparable. Finally, by contraposition of the first item of Lemma 1, object maxmod A is before property maxmod B in LIST.
- Suppose that $E(B) = E(A)$. This corresponds to the case where object maxmod A and property maxmod B appear in the same element of the GSH. We will show that B is placed after A in LINEXT, which is equivalent to saying that A is placed after B in LIST.

If $E(A) = E(B)$ then, by Lemma 2, $(A, B) \in R$ and thus $(A, B) \notin Z_2$. By contraposition of the second item of Lemma 1, object maxmod A is after property maxmod B in LIST. Finally, as a direct consequence of the precedent case, no other maxmod will appear between A and B in LIST.

6 Conclusion

In this paper, we present a new algorithm to efficiently compute an ordering on the object and property maxmods which is compatible with a linear extension of the Galois sub-hierarchy.

It turns out that the linear extension we compute has very special properties, which will require further investigation, both as useful for dealing with Galois sub-hierarchies, and as interesting in the more general context of handling a binary relation. For example, the way the algorithm traverses the sub-hierarchy is interesting, as well as the definition of some zones of the matrix with non-comparable elements.

Moreover, the family of reorderings computed by our Tom Thumb algorithm turns out to often lead to cases where concept generation can be accomplished faster than in the general case. Experimentation on this is being pursued.

References

1. Barbut M., Monjardet B.: *Ordre et classification.* Classiques Hachette, (1970).
2. Berry A., Sigayret A.: Representing a concept lattice by a graph. *Proc. Discrete Maths and Data Mining Workshop, 2nd SIAM Conference on Data Mining (SDM'02), Arlington (VA, USA)*, (April 2002). *Discrete Applied Mathematics, special issue on Discrete Maths and Data Mining*, **144(1-2)** (2004) 27–42.
3. Berry A., Sigayret A.: Maintaining class membership information. *Advances in Object-Oriented Information Systems - OOIS 2002 Workshops*, LNCS, **2426** (Sept. 2002) 13–23.
4. Berry A., Bordat J-P., Sigayret A.: Concepts can't afford to stammer. *INRIA Proc. International Conference "Journées de l'Informatique Messine" (JIM'03), Metz (France)*, (Sept. 2003). Submitted as 'A local approach to concept generation.'
5. Birkhoff G.: *Lattice Theory.* American Mathematical Society, 3rd Edition, (1967).
6. Dao M., Huchard M., Leblanc H., Libourel T., Roume C.: A New Approach of Factorization : Introducing Metrics. *Proc. 8th IEEE international Software Metrics Symposium (METRICS'02), Ottawa (Canada)*, (June 2002) 227–236.
7. Dao M., Huchard M. Rouane Hacene M., Roume C., Valtchev P.: Improving Generalization Level in UML. *LNCS Proc. 12th International Conference on Conceptual Structures (ICCS'04)*, (2004) 346–360.
8. Dicky H., Dony C., Huchard M., Libourel T.: ARES, Adding a class and RESstructuring Inheritance Hierarchies. Proc. *11$^{\text{me}}$ Journées Bases de Données Avancées*, Nancy (France), (1995) 25–42.
9. Ganter B., Wille R.: *Formal Concept Analysis.* Springer, (1999).
10. Godin R., Chau T. T.: Comparaison d'algorithmes de construction de hiérarchies de classes. *L'Objet*, **5(3)** (2000) 321–338.

11. Godin R., Mili H.: Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. *Proc. OOPSLA'93, Washington (DC, USA)*, Special issue of Sigplan Notice, **28(10)** (1993) 394–410.
12. Godin R., Mili H, Mineau G., Missaoui R., Arfi A., Chau T. T.: Design of Class Hierarchies Based on Concept (Galois) Lattices. *Theory and Practice of Object Systems.* **2(4)** (1998) 117–134.
13. Hitzler P.: Default Reasoning over Domains and Concept Hierarchies. *LNAI Proc. 27th German Conference on Artificial Intelligence, (KI'04), Ulm (Germany).* (Sept. 2004).
14. Huchard M., Dicky H., Leblanc H.. Galois lattice as a framework to specify building class hierarchies algorithms. *Theoretical Informatics and Applications*, EDP Science ed., **34** (Jan. 2000) 521–548.
15. Huchard M., Leblanc H.: From Java classes to Java interfaces through Galois lattices. *Proc. 3rd International Conference on Orders, Algorithms and Applications (Ordal'99)*, Montpellier (France), (1999) 211–216.
16. Osswald R., Pedersen W.: Induction of Classifications from Linguistic Data. *Proc. ECAI-Workshop on Advances in Formal Concept Analysis for Knowledge Discovery in Databases (FCAKDD'02), Lyon (France)*, (July 2002).
17. Pedersen W.: A Set-Theoretical Approach for the Induction of Inheritance Hierarchies. *Electronic Notes in Theoretical Computer Science*, **51** (July 2001).
18. Spinrad J. P.: Doubly Lexical Orderings of Dense 0-1 Matrices. *Information Processing Letters*, **45** (1993) 229-235.
19. UML 2.0 superstructure Specification. *OMG Final Adopted Specification ptc/03-08-02.* URL http://www.omg.org/
20. Yahia A., Lakhal L., Cicchetti R., Bordat J-P.: iO2, An Algorithmic Method for Building Inheritance Graphs in Object Database Design *Proc. 15th Conf. on Conceptual Modeling (ER'96), Cottbus (Germany)*, LNCS **1157** (1996) 422–437.