Theoretical Informatics and Applications Will be set by the publisher
Informatique Théorique et Applications

GALOIS LATTICE AS A FRAMEWORK TO SPECIFY BUILDING CLASS HIERARCHIES ALGORITHMS *

M. Huchard[1], H. Dicky[1] and H. Leblanc[1]

Abstract. In the context of object-oriented systems, algorithms for building class hierarchies are currently receiving much attention. We present here a characterization of several *global* algorithms. A global algorithm is one which starts with only the set of classes (provided with all their properties) and directly builds the hierarchy.

The algorithms scrutinized were developped each in a different framework. In this survey, they are explained in a single framework, which takes advantage of a substructure of the Galois lattice associated with the binary relation mapping the classes to their properties. Their characterization allow to figure the results of the algorithms without running them in simple cases. This study once again highlights the Galois lattice as a main and intuitive model for class hierarchies.

ACM classification. D.1.5, I.2.2

AMS Subject Classification. 06A15.

1. Introduction

Building a hierarchy of classes is an important step in object-oriented development. These hierarchies must be easily maintained and adapted to new requirements, while their components must be widely reusable. The level of reuse is closely related to a clear organization of components as well as to the emergence of components at different abstraction levels. These qualities should be obtained through systematic construction and organization of the classes. The emergence of abstractions (superclasses) is the result of an intrisically complex computation.

Keywords and phrases: Object-oriented systems, class hierarchy, Galois lattice, class hierarchy construction algorithms

* *This work is supported by France Télécom R&D, contract 971B602.*

[1] LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France, email: {name}@lirmm.fr

Because it is based on the factorization of common features, this task requires an examination of common features for all possible class subsets, which are 2^n, where n is the number of classes. This complexity is fortunately reduced in numerous cases where classes are partitionned into obviously unrelated sets.

These considerations have given rise to the development of methods to assist designers, and to provide them with heuristics [12] or algorithms for constructing and modifying class hierarchies. Concerning algorithms, some are *incremental*, and add classes one after another to an already existing hierarchy. By contrast, *global* algorithms start with a binary relation *own-property* mapping the classes to their properties, and produce a class hierarchy, that is, a set of classes possibly larger than the input class set, and a partial order on this set. Due to the similarity of some of the proposed algorithms, a precise characterization in a unifying framework is necessary to specify their respective results.

Galois lattice theory will be used as the unifying framework allowing to understand and compare the results of the algorithms. As we will show, the studied global algorithms build a particular sub-order of the Galois lattice associated with the *own-property* relation, or a structure close to that sub-order. To specify these algorithms, we will compare them with a simple Galois sub-hierarchy construction scheme. The importance for hierarchy construction of the Galois lattice and its special sub-order was firstly highlighted by [7] and was also used in [6, 9] as the underlying structure for an incremental algorithm (which adds one class to a class hierarchy). Galois lattices (or concept lattices [16]) have many applications in domains such as knowledge representation, machine learning (conceptual clustering), classification, software engineering, and more recently data mining. In [8], the main applications are presented.

The paper is organized as follows. Section 2 shows how different are the results of the algorithms upon an actual example. Section 3 describes the theoretical framework. The Galois lattice associated with a binary relation is defined along with some simplifications, as proposed in [7,9]. These simplifications lead to the definition of the Galois sub-hierarchy, which is a specified suborder of the Galois lattice. In Section 4, a Galois sub-hierarchy construction scheme is given. It does not constitute an efficient algorithm, but it is very useful as a tool to understand the algorithms we study. Then four different algorithms are characterized: the algorithm of Chen et al. [2] in Section 5, the algorithm of Moore et al. [14] in Section 6, the algorithm of Cook [4] in Section 7 and finally, an algorithm of Mineau et al. [13] (proposed in the framework of conceptual graphs) in Section 8. In Section 9, we summarize and compare the different results, then we conclude with the advantages and drawbacks of using the Galois lattice and sub-hierarchy as models of class hierarchies.

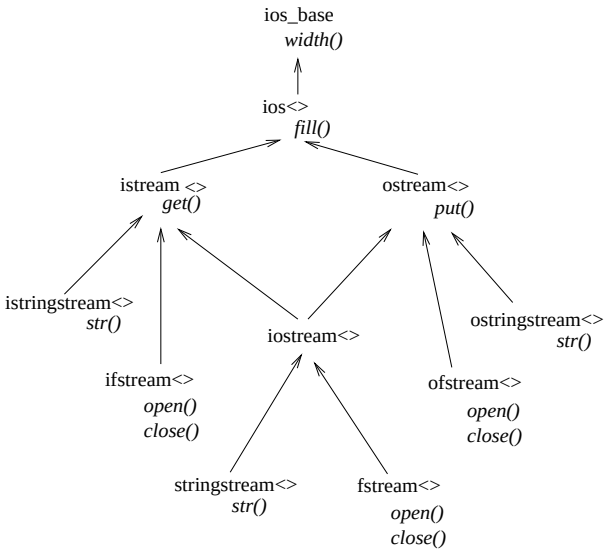

FIGURE 1. The main elements of the C++ stream hierarchy

2. ILLUSTRATING THE DIFFERENCES BETWEEN CONSTRUCTED HIERARCHIES

We motivate our study with an example borrowed from C++ [15]. It shows both the need of algorithms to build or improve class or type hierarchies, and the differences between hierarchies produced by the main construction algorithms proposed in the literature.

The hierarchy we will examine is a simplified[1] version of the stream hierarchy (see Figure 1). Class names have been shorten: prefixes `basic_` have been removed, and template parameters are omitted. Besides, we focus on the available operations, and not on their implementation. The resulting structure may be considered as the hierarchy of the C++ stream types.

At first sight, this stream hierarchy has a good shape, as it presents a central diamond representing the basic kinds of streams regarding input/output features, and further specializations associated to different data structures (file and string). If we consider this hierarchy more closely, however, it is not so well organized. First, several abstractions, like "file" or "string" are missing. They are actually spread over several types, as `ofstream`, `ifstream`, `fstream`, according the input/output operations. As a consequence, operations like "open", "close", or "str" are declared in more than one type. In some C++ versions, they are even implemented almost the same way. Second, several specialization links are missing: "input/output file" (`fstream`) might appear as a specialization of both "input file" (`ifstream`) and "output file" (`ofstream`) but the hierarchy does not contain this information.

[1] Note however that the simplification does not change the main semantics of the hierarchy.

own-property	width()	fill()	get()	put()	str()	open()	close()
ios_base	x						
ios<>	x	x					
istream<>	x	x	x				
istringstream<>	x	x	x		x		
ifstream<>	x	x	x			x	x
ostream<>	x	x		x			
ostringstream<>	x	x		x	x		
ofstream<>	x	x		x		x	x
iostream<>	x	x	x	x			
stringstream<>	x	x	x	x	x		
fstream<>	x	x	x	x		x	x

FIGURE 2. The C++ stream hierarchy mapping

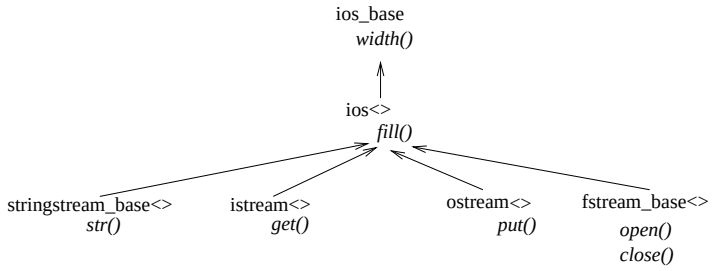

FIGURE 3. The stream hierarchy re-built with the algorithm of Mineau et al.

All the algorithms we study rearrange such hierarchies, at least to propose designers other possible solutions. We will consider five solutions to our example, four of them are built by known algorithms, the fifth is the reference used to compare them and has, from our point of view, the most logical organization. The input of these algorithms is a binary relation mapping the classes to their properties (Figure 2).

The algorithm of [13] essentially factorizes features. We believe it is meant for objects which are not comparable. This is why, applied to our problem where types are comparable, some types are missing (see Figure 3). This algorithm is nevertheless a very good basis because if we apply both the algorithm and the "dual" (type-oriented instead of property-oriented) the whole reference structure is obtained, as we will see below.

The algorithms [2] and [4] intend adding the missing types, but they do not compare these new types together (Figure 4). As a result, the factorization of properties is correct, but specialization links are missing, even more than in the original C++ hierarchy, for instance between `fstream` and `iostream`, and the whole scheme becomes complicated.

Another algorithm, proposed by [14] in a prototype-based language framework, furthermore assumes that input classes have to appear as leaves in the output

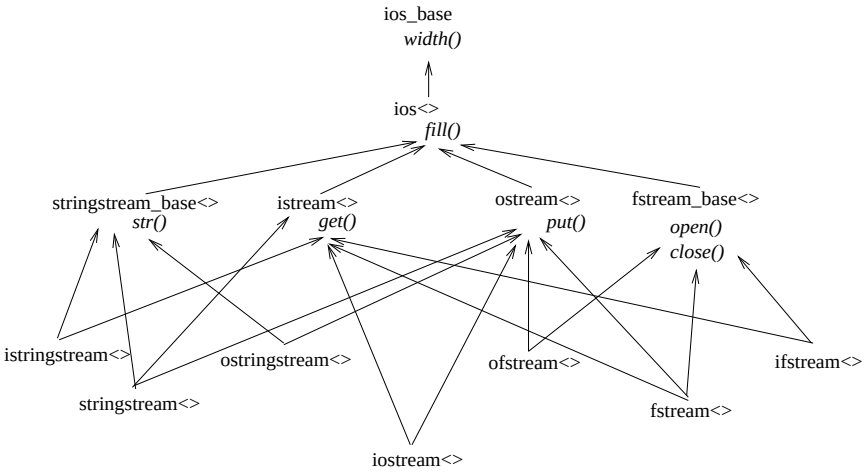

FIGURE 4. The stream hierarchy re-built with the algorithms of Chen et al. and Cook

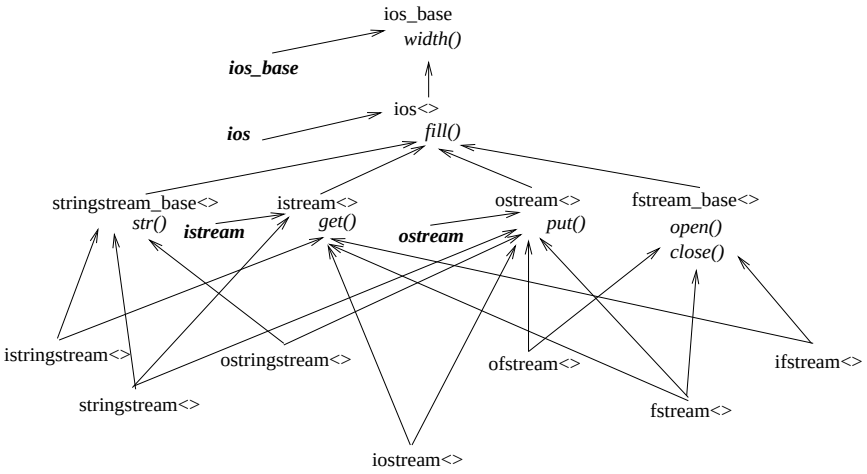

FIGURE 5. The stream hierarchy re-built with the algorithm of Moore et al.

hierarchy. Apart of that, its result is the same as in [2] and in [4] (in Figure 5, which shows the output hierarchy, the classes added because of this additional assumption are in bold face).

The structure we will take as a reference is shown Figure 6. Compared with others, it contains all the specialization links one can expect; it has a very clear construction principle: a copy of the central diamond is made following the two kinds of data structure (file and string).

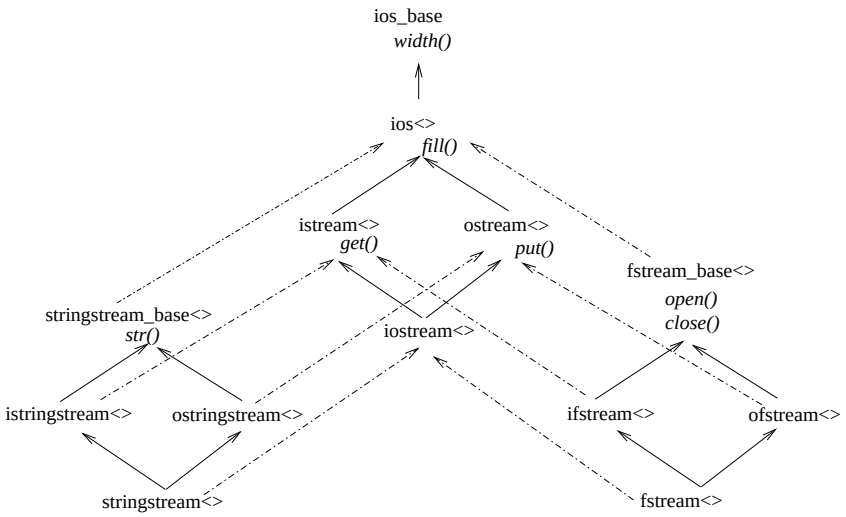

FIGURE 6. The stream hierarchy built from the Galois sub-hierarchy

The respective advantages of the different output hierarchies may be discussed, but at least, this introductory example demonstrates that a characterization of these algorithms is essential. Next section gives the theoretical framework we think suitable for that.

3. GALOIS LATTICES AND SUB-HIERARCHIES

This section introduces some useful definitions about Galois lattices and sub-hierarchies. For more details, the reader can refer to [1, 5, 8]. Proofs are given for properties which are not well known in the field.

A Galois lattice essentially presents and organizes all non-empty intersections between property sets of given objects (here classes, in the sense of object programming). In that structure, the properties shared are clustered into "concepts" which describe objects. Properties are maximally factorized, with a minimal number of concepts introduced by the factorization.

In the next two subsections we first define the Galois lattice structure associated to a binary relation, and then the simplifications that lead to the Galois sub-hierarchy structure.

3.1. GALOIS LATTICES

Let E and F be two finite sets and let R be a binary relation between E and F. In our context, E is the set of classes, and F the set of properties. $(x, y) \in \mathcal{R}$ for $x \in E$ and $y \in F$ means x owns y. Figure 7 shows the binary relation used to illustrate the following definitions.

Properties

Classes	a	b	c	d	e	f
1	x	x				
2			x			
3	x		x	x		x
4	x		x		x	x
5	x					

FIGURE 7. Relation R1 between E=1..5 and F=a..f

Definition 3.1. The construction is based on two fundamental mappings f and g defined below.
$f : \mathcal{P}(E) \to \mathcal{P}(F)$
$X \longmapsto \{y \in F \mid \forall x \in X, (x, y) \in \mathcal{R}\}$
$g : \mathcal{P}(F) \to \mathcal{P}(E)$
$Y \longmapsto \{x \in E \mid \forall y \in Y, (x, y) \in \mathcal{R}\}$

For $X \subseteq E$, $f(X)$ is the set of properties owned by all classes of X. For instance, with $X = \{2, 3, 4\}$, $f(X)$ is the set of properties shared by 2, 3 and 4, and more exactly $\{c\}$. Symmetrically, for $Y \subseteq F$, $g(Y)$ is the set of classes owning all the properties of Y. For instance, if $Y = \{a, c\}$, $g(Y)$ is the set of classes that own both a et c, that is $g(Y) = \{3, 4\}$. These mappings play a very symmetrical role, as shown below.

Definition 3.2. Let $(A, \leq_A)$ and $(B, \leq_B)$ be ordered sets, and let $f_1 : A \to B$ and $f_2 : B \to A$ be mappings. The pair (f_1, f_2) is a Galois connection when for all $a \in A$ and $b \in B$, $a \leq_A f_2(b)$ if and only if $b \leq_B f_1(a)$.

Proposition 3.3. *Let f and g as in Definition 3.1. (f, g) is a Galois connection between $(\mathcal{P}(E), \subseteq)$ and $(\mathcal{P}(F), \subseteq)$, that is, $\forall X \in \mathcal{P}(E)$, $Y \in \mathcal{P}(F)$, $X \subseteq g(Y)$ if and only if $Y \subseteq f(X)$.*

Furthermore, f and g have the obvious following property.

Proposition 3.4. *f and g are decreasing mappings.*

The composition of f and g defines two operators, $h_E = gof$ and $h_F = fog$. $h_E(X)$ is the set of classes that own all the properties that are owned by all the classes of X. Symmetrically $h_F(Y)$ is the set of properties that are owned by all the classes that own all the properties of Y.

Proposition 3.5. *$h_E = gof$ et $h_F = fog$ are closure operators, that is (for instance for h_E):*

1. *(increasing) $X_1, X_2 \subseteq E$, $(X_1 \subseteq X_2) \Rightarrow (h_E(X_1) \subseteq h_E(X_2))$*
2. *(extensive) $X \subseteq E \Rightarrow X \subseteq h_E(X)$*
3. *(idempotent) $X \subseteq E \Rightarrow h_E(h_E(X)) = h_E(X)$*

Definition 3.6. The sets $X \subseteq E$ (resp. $Y \subseteq F$) such that $h_E(X) = X$ (resp. $h_F(Y) = Y$) are said to be closed with respect to h_E (resp. to h_F).

For instance, $\{a, c, f\}$ is a closed set, while $\{a, c\}$ is not. We have indeed $g(\{a, c\}) = \{3, 4\}$, and $f(g(\{a, c\})) = f(\{3, 4\}) = \{a, c, f\}$.

These closed sets may be organized into a lattice. We recall first the definitions of infimum, supremum, and lattice [5].

Definition 3.7. Let P an ordered set, and let $S \subset P$.

When the set $\{x \in P \mid \forall s \in S,\ s \geq x\}$ has a largest element, this element is called the infimum of S.

When the set $\{x \in P \mid \forall s \in S,\ s \leq x\}$ has a least element, this element is called the supremum of S.

If for all $x, y \in P$, $\{x, y\}$ admits an infimum and a supremum, denoted respectively by $x \wedge y$ and $y \vee x$, then P is a lattice.

Definition 3.8. The set of sets closed with respect to h_E, denoted by $\mathcal{L}_E$ (resp. $\mathcal{L}_F$ for h_F), ordered by the set inclusion $\subset$ (resp. by $\supset$) is a lattice. The infimum and supremum operators are as follows, where X_1, X_2 are closed sets of h_E (resp. Y_1, Y_2 closed sets of h_F) :

$$X_1 \wedge X_2 = (X_1 \cap X_2) \text{ (resp. } Y_1 \wedge Y_2 = h_F(Y_1 \cup Y_2))$$
$$X_1 \vee X_2 = h_E(X_1 \cup X_2) \text{ (resp. } Y_1 \vee Y_2 = Y_1 \cap Y_2)$$

Figure 8 shows $\mathcal{L}_E$ on the left, and $\mathcal{L}_F$ in the center. Let us look at some examples of infimum and supremum computations.

For $X_1 = \{1\}$ and $X_2 = \{3, 4\}$, we have $X_1 \vee X_2 = h_E(X_1 \cup X_2) = \{1, 3, 4, 5\}$ which shows that $\{1, 3, 4\}$ is not a closed set. With $X_1 = \{3\}$ and $X_2 = \{4\}$, we have $X_1 \vee X_2 = h_E(X_1 \cup X_2) = \{3, 4\}$ which shows that $\{3, 4\}$ is a closed set. With $X_1 = \{1, 3, 4, 5\}$ and $X_2 = \{2, 3, 4\}$, we have $X_1 \wedge X_2 = X_1 \cap X_2 = \{3, 4\}$.

$\mathcal{L}_E$ and $\mathcal{L}_F$ are isomorphic (there is a bijective map between them which preserve the infimum and supremum), and this property leads to the Galois lattice definition.

Definition 3.9. A concept is a pair (X, Y), with $X \in \mathcal{L}_E$, and $Y \in \mathcal{L}_F$, and $Y = f(X)$ (or equivalently $X = g(Y)$).

Definition 3.10. The Galois lattice is the set $\mathcal{L}$ of concepts, provided with the order $<_{\mathcal{L}}$ defined by $(X_1, Y_1) <_{\mathcal{L}} (X_2, Y_2)$ iff $X_1 \subset X_2$ (which is equivalent to $Y_2 \subset Y_1$). Infimum and supremum operators consist in applying to left and right members of the pairs the infimum and supremum operators defined for $\mathcal{L}_E$ (resp. $\mathcal{L}_F$).

Figure 8 (right) shows the Galois lattice for the example described in Figure 7.

3.2. Galois sub-hierarchy

In [8], several lattice simplifications based upon the closed sets (Def. 3.6) are proposed. The size of the whole lattice can, in the worst case, be exponential with respect to the number of classes and the number of properties, so it is judicious to reduce the set of concepts.

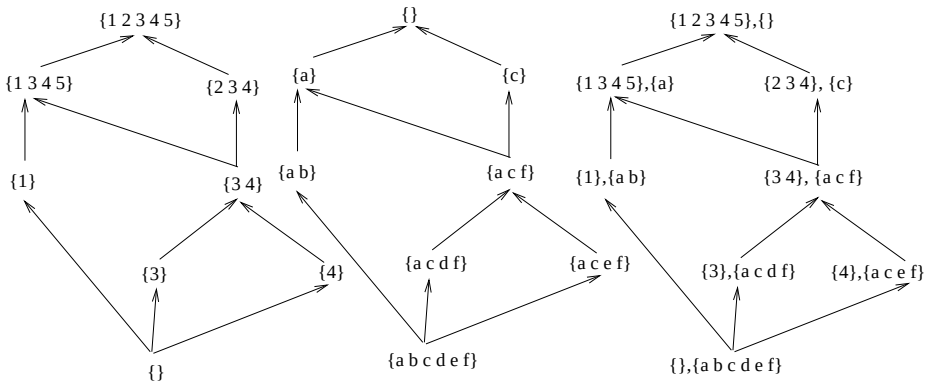

FIGURE 8. Lattices $\mathcal{L}_E$, $\mathcal{L}_F$ and $\mathcal{L}$

Moreover, redundancy in the structure itself can be observed between the lattice node labels (the pairs of closed sets) and the order [7]. A property (resp. a class) belonging to a node belongs to all the nodes below (resp. above). The simplication we will use considers that properties and classes are unnecessarily repeated in each concept along lattice paths, and need be quoted only in their introducing node.

Due to this redundancy, two symmetrical simplifications are made on pairs of closed sets in the lattice. The simplification concerning properties is presented in more detail, being the more intuitive from an object-oriented point of view.

Definition 3.11. Let (X, Y) be a vertex of $\mathcal{L}$, the vertex simplified according to properties is denoted by (X, Y') where $Y' = Y \setminus \text{inherited}(Y)$ with $\text{inherited}(Y) = \{y \in Y \text{ s.t. } \exists Y_2 \in \mathcal{L}_F,\ y \in Y_2 \text{ and } Y_2 \subset Y\}$

Properties removed this way from Y are inherited from vertices higher than (X, Y) in the lattice (according to the order $<_{\mathcal{L}}$).

This simplification of a concept also applies to its right part. For a property closed set $Y \in \mathcal{L}_F$, the simplification leads to $Y' = Y \setminus \text{inherited}(Y)$.

Before giving some properties of this simplification, let us give a property useful in later proofs.

Proposition 3.12. *For any $X \subseteq E$, $f(X)$ is a closed set w.r.t h_F.*

Proof. From Proposition 3.5.2. (h_F is extensive), it follows that $f(X) \subseteq h_F(f(X))$. Symmetrically, Proposition 3.5.2. (h_E is extensive) implies $X \subseteq h_E(X)$, and the fact that f is decreasing yields $f(X) \supseteq f(h_E(X))$. Remarking that $f(h_E(X)) = f(g(f(X))) = h_F(f(X))$ concludes the proof. □

Proposition 3.13. *If (X, Y') results from a simplification according to properties applied to (X, Y), then for all y, $y \in Y'$ if and only if $g(\{y\}) = X$.*

Proof. Firstly, it is shown that $y \in Y'$ implies $g(\{y\}) = X$.

$\forall y \in Y$, $g(\{y\}) \supseteq g(Y)$ since g is decreasing. Now $Y = f(X)$ according to the closed set definition, $g(Y) = g(f(X)) = X$ since X is a closed set of h_E. As a consequence, $g(\{y\}) \supseteq X$, for all $y \in Y$.

Suppose now that $g(\{y\}) \neq X$ with $y \in Y'$. $g(\{y\})$ is a closed set (Proposition 3.12). There is thus a vertex (X'', Y'') in $\mathcal{L}$ such that $X'' = g(\{y\})$. By definition, $(X'', Y'') >_{\mathcal{L}} (X, Y)$ since $X'' \supset X$. As $y \in Y''$, y is inherited and does not belong to Y' according to the simplification definition. Contradiction.

Conversely, if $g(\{y\}) = X$, y appears in all classes of X and only in these classes, then $y \in f(X) = Y$ according to the definition of the Galois connection; moreover, as for every vertex (X'', Y'') higher in the lattice, we have $X'' \supset X$, there is in X'' at least one class that does not own y, so y can not belong to $f(X'')$: y can not be inherited. □

Proposition 3.14. *If (X, Y') is the result of the simplification according to properties applied to (X, Y), then for any $y \in Y'$, (X, Y) is the highest vertex in the lattice where y appears. y is said "declared" by the vertex (X, Y).*

Proof. According to the definition of the simplification, there is no vertex higher in the lattice that owns y in its simplified right part. By Galois lattice construction, a closed set corresponds to only one vertex, so (using Proposition 3.13) there can not be two vertices that declare y. □

A simplification according to classes can be defined symmetrically.

Definition 3.15. Let (X, Y) be a pair of closed sets, the simplification according to classes leads to (X', Y) where $X' = X \setminus \text{subclass}(X)$, such that $\text{subclass}(X) = \{x \in X \text{ s.t. } \exists X_2 \in \mathcal{L}_E, x \in X_2 \text{ and } X_2 \subset X\}$.

A class removed from X this way, appears in the left part of one vertex lower than (X, Y) (considering $<_{\mathcal{L}}$).

A simplified class closed set X' for a class closed set $X \in \mathcal{L}_E$ is $X' = X \setminus \text{subclass}(X)$.

Proposition 3.16. *For any $Y \subseteq F$, $g(Y)$ is a closed set.*

Proposition 3.17. *If (X', Y) is the pair of closed sets (X, Y) simplified according to classes, for any x, $x \in X'$ if and only if $f(\{x\}) = Y$.*

Proposition 3.18. *If (X', Y) is the pair of closed sets (X, Y) simplified according to classes, $\forall x \in X'$, (X, Y) is the lower vertex where x appears. x is said "defined by" (X, Y).*

Definition 3.19. Vertices whose simplified form, according to class and property simplification, is $(\emptyset, \emptyset)$ are called "empty simplified elements".

Proposition 3.20. *If (X, Y) is an empty simplified element, then:*

- *$\forall x \in X$, there is a pair $(X_1, Y_1) \in \mathcal{L}$ s.t. $(X_1, Y_1) <_{\mathcal{L}} (X, Y)$ and $x \in X_1$,*
- *$\forall y \in Y$, there is a pair $(X_1, Y_1) \in \mathcal{L}$ s.t. $(X, Y) <_{\mathcal{L}} (X_1, Y_1)$ and $y \in Y_1$.*

Definition 3.21. The Galois sub-hierarchy $\mathcal{S}$ is the suborder of the Galois lattice induced by its non-empty simplified elements.

	a	b	c	d	e	f
1	x	x	x			
2			x			
3	x		x	x		x
4	x		x		x	x
5	x	x				

A binary relation R2 on E = {1, 2 ,3, 4, 5} and F = {a, b, c, d, e, f}

FIGURE 9. Relation R2

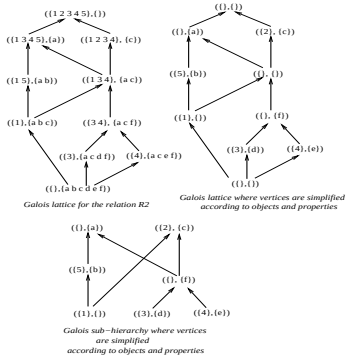

FIGURE 10. Simplification according classes (left), properties (center), and Galois sub-hierarchy (right)

Figures 9 and 10 show a binary relation, which is a variant of the previous example, and the following associated structures: the Galois lattice, the simplified Galois lattice, and the Galois sub-hierarchy. The Galois lattice, as well as the Galois sub-hierarchy may be interpreted as standard inheritance hierarchies that each organizes the input classes. The left part of a simplified pair contains the names of the input classes defined by this vertex. When this left part is empty, a new factorization superclass has been discovered. The right part of a simplified pair contains the properties declared by the vertex. When this right part is empty, all properties of the vertex are declared by superclasses. The Galois sub-hierarchy preserves the most informative part of the lattice: all the vertices that either define at least one class, or declare at least one property.

4. A SIMPLE CONSTRUCTION SCHEME

As a background of the algorithms we want to characterize, we propose a simple construction scheme for the Galois sub-hierarchy. Its role is to help to understand the behaviour of the algorithms to be characterized, thus it has to be considered

without efficiency concern, only as a starting point to develop efficient (polynomial) algorithms.

This construction is based on producing two kinds of set pairs :

- pairs $(g(f(\{e\})), f(\{e\}))$, for every $e \in E$;
  each pair is obtained from a single class e, by the construction of the pair of closed sets that defines this class;
  the set of these pairs will be denoted by O_C for "Obtained from a Class";

- pairs $(g(\{a\}), f(g(\{a\})))$, for every $a \in F$;
  these pairs are obtained from a single property a, when building the pair of closed sets that declares this property;
  we will denote by O_P the set of such pairs for "Obtained from a Property".

Let us remark that, firstly O_C and O_P may intersect, secondly several properties (resp. classes) can lead to the same pair of closed sets, when $g(\{a\}) = g(\{a'\})$ for some $a, a' \in F$ (resp. $f(\{e\}) = f(\{e'\})$, for some $e, e' \in E$), and then an efficient algorithm based on this principle, at least, has to avoid redundant computation of pairs. The algorithms studied later properly produce some of these pairs, without duplicates. The pairs are ordered according to the inclusion between left (or right) members.

Let $\mathcal{R}$ be a binary relation between E and F, and (f, g) defined as in 3.1. Let $\mathcal{L}$ be the Galois lattice (resp. $\mathcal{S}$ the Galois sub-hierarchy) associated with $\mathcal{R}$, and denote by $V(\mathcal{L})$ (resp. $V(\mathcal{S})$) its set of vertices. Let $O_C = \{(g(f(\{e\})), f(\{e\})),$ for $e \in E\}$ and $O_P = \{(g(\{a\}), f(g(\{a\}))),$ for $a \in F\}$. Finally let $\mathcal{V}$ be the set of empty simplified pairs. By definition of $\mathcal{S}$, $V(\mathcal{S}) = V(\mathcal{L}) \setminus \mathcal{V}$.

We prove below (Theorem 4.4) the correctness of such a construction, that is $V(\mathcal{S}) = O_C \cup O_P$.

Lemma 4.1. *$O_C \cup O_P \subseteq V(\mathcal{L}))$, that is the produced pairs are vertices of $\mathcal{L}$.*

Proof. By Proposition 3.12, $f(\{e\})$ for $e \in E$ is a closed set with respect to h_E. $g(f(\{e\}))$ is the associated closed set with respect to h_F in the isomorphism between $\mathcal{L}_\mathcal{E}$ and $\mathcal{L}_\mathcal{F}$, thus $(g(f(\{e\})), f(\{e\}))$ is an element of $\mathcal{L}$.

We have thus $O_C \subseteq V(\mathcal{L}))$. Symmetrically, using Proposition 3.16, $O_P \subseteq V(\mathcal{L})$. □

Lemma 4.2. *$(O_C \cup O_P) \cap \mathcal{V} = \emptyset$, that is no empty simplified element is produced.*

Proof. For $y \in F$, the pair $(g(\{y\}), f(g(\{y\})))$ can not be an empty simplified element, since in particular $y \in f(g(\{y\}))$ and by properties 3.13 et 3.14, y can not appear in higher elements. Then $O_P \cap \mathcal{V} = \emptyset$. Symmetrically, by properties 3.17 and 3.18, $O_C \cap \mathcal{V} = \emptyset$.

□

Lemma 4.3. *Every pair of closed sets which is not an empty simplified element is produced: $V(\mathcal{S}) \subseteq (O_C \cup O_P)$.*

Proof. Let (X, Y) be a pair of closed sets of $V(\mathcal{S})$. As $(X, Y) \notin \mathcal{V}$, either X contains a class which does not belong to lower elements of $\mathcal{L}$, or Y contains a

property which does not appear in higher elements. In the second case, let y such a property. If (X', Y') is the simplified form of (X, Y) according to classes and properties, $y \in Y'$ and by Property 3.13, $g(\{y\}) = X$. As a result, $(X, Y) \in O_P$. The first case is symmetrical, and gives $(X, Y) \in O_C$. □

Theorem 4.4. *Let $\mathcal{R}$ be a binary relation and (f, g) defined as in 3.1. Let $\mathcal{S}$ be the Galois sub-hierarchy associated with $\mathcal{R}$, and $V(\mathcal{S})$ its set of vertices. If $O_C = \{(g(f(\{e\})), f(\{e\})), \text{for } e \in E\}$ and $O_P = \{(g(\{a\}), f(g(\{a\}))), \text{for } a \in F\}$, then $V(\mathcal{S}) = O_C \cup O_P$.*

Proof. From previous lemmas, we have $V(\mathcal{S}) \subseteq (O_C \cup O_P) \subseteq V(\mathcal{L})$. This inequality may be rewritten removing the set $\mathcal{V}$, giving $(V(\mathcal{S}) \setminus \mathcal{V}) \subseteq ((O_C \cup O_P) \setminus \mathcal{V}) \subseteq (V(\mathcal{L}) \setminus \mathcal{V})$, or, since $(O_C \cup O_P) \cap \mathcal{V} = \emptyset$, $V(\mathcal{S}) \subseteq (O_C \cup O_P) \subseteq V(\mathcal{S})$. □

Next sections propose a characterization of several global algorithms, using the Galois lattice framework. For each algorithm, three subsections describe respectively:

- the notations and aims of the authors,
- a presentation of the algorithm,
- a characterization of what is computed by the algorithm.

For each algorithm, we match the notations of the papers with our own (E, F, f, g, ...).

5. Chen and Lee algorithm [2]

5.1. Notations and aims

Objects are given with their set of properties. Figure 11 is an example found in [2] where it is used to describe the algorithm. The object set is $E = T = \{t_1, ..t_n\}$, the property set is $F = A = \{a_1, ..a_n\}$. A "class" is defined as a subset of A, a "set of classes" is defined as a partition of A. A mapping IC (for "Inverse Containement") is defined by $IC(a) = g(\{a\})$, for $a \in A$ (and $IC(C) = g(C)$ for $C \in \mathcal{P}(A)$).

The algorithm is intended to produce a subtype hierarchy, which is defined as types (classes in our vocabulary) ordered by the inclusion relation between property sets (Figure 12). In this figure and in some places in our description of the algorithm, the symbol which is used in the Chen and Lee paper to denote the right part (simplified) of a pair (the class, for the authors), will also be used to denote the pair itself.

It is important to mention that the paper also contains a more general discussion about properties, which is not detailed here since it is out the scope of this paper.

5.2. Algorithm description

We will follow the steps as they are presented in Chen and Lee paper.

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17
t1	x	x	x	x		x	x								x		
t2	x	x	x		x	x		x							x		
t3	x	x	x		x	x			x						x		
t4	x	x	x		x					x	x	x	x	x			
t5	x	x	x		x					x	x			x			
t6	x	x	x		x									x			
t7	x	x	x													x	
t8	x	x	x													x	x
t9	x	x	x														x

FIGURE 11. Relation R3

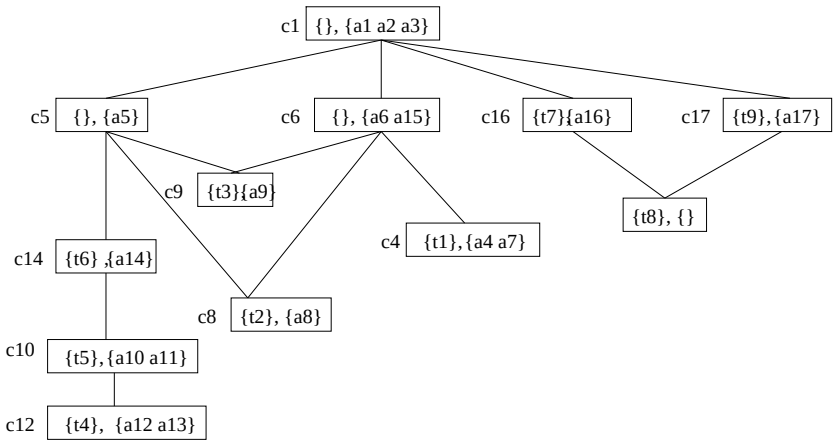

FIGURE 12. Subtype hierarchy for Relation R3

Step 0. ▷ *"Find IC(a) for each attribute a" [2]*
As a result, $g(\{a\})$ is computed for every $a \in F$.

```
g({a1})=g({a2})=g({a3})
       ={t1, t2, t3, t4, t5, t6, t7, t8, t9}
g({a4})=g({a7})={t1}
g({a5})={t2, t3, t4, t5, t6}
g({a6})=g({a15})={t1, t2, t3}
g({a8})={t2}
g({a9})={t3}
g({a10})=g({a11})={t4, t5}
g({a12})=g({a13})={t4}
g({a14})={t4, t5, t6}
g({a16})={t7, t8}
g({a17})={t8, t9}
```

Step 1. ▷ *"Find a partition of attribute set A as a set of classes C' <...> based on the equivalence relations among the containment sets of attributes" [2]*
This allows to obtain the pairs (X, Y') where $X = g(\{a\})$ and Y' is the set $f(g(\{a\}))$ simplified according to properties.

In the following, such a set will be denoted by $f'(g(\{a\}))$ and not $(f(g(\{a\})))'$ as an extension of the « prime » notation, and for sake of readibility. Each set Y' built in such a way is a class.

```
X1={t1, t2, t3, t4, t5, t6, t7, t8, t9}
    Y'1=C1={a1, a2, a3}
X2={t1}
    Y'2=C4={a4, a7}
X3={t2, t3, t4, t5, t6}
    Y'3=C5={a5}
X4={t1, t2, t3}
    Y'4=C6={a6, a15}
X5={t2}
    Y'5=C8={a8}
X6={t3}
    Y'6=C9={a9}
X7={t4, t5}
    Y'7=C10={a10, a11}
X8={t4}
    Y'8=C12={a12, a13}
X9={t4, t5, t6}
    Y'9=C14={a14}
X10={t7, t8}
     Y'10=C16={a16}
X11={t8, t9}
     Y'11=C17={a17}
```

In the following, $C' = \{C1..C17\}$.

Steps 2.1 and 2.2. ▷ *"Create the directed acyclic graph $G = (C', Ed)$, where $Ed = \{c_i \rightarrow c_j / IC(c_j) \subset IC(c_i)\}$ <...> Remove all the transitive subclass relations" [2]*

The following order $<$ is built: $(X_1, Y_1') < (X_2, Y_2')$ iff $X_1 \subset X_2$.

This order is the lattice order.

Step 2.3. ▷ *"Associate objects T to classes <...> For each object t_i, consider the subgraph G_i induced by C_j's such that t_i is in $IC(C_j)$. If G_i contains only one leaf[2] then associate t_i to that leaf; otherwise, create a class c with no attributes and add edges from the leaves to c and associate t_i to c." [2]*

This step may be characterized in the following way:

- for an object t such that there is a property a with $f(g(\{a\})) = f(\{t\})$, a pair $C = (X, Y)$, $Y = f(\{t\})$ was built at step 1, and by Proposition 3.17, $t \in X'$. An example of this case is given with $t = t_6$ and $a = a_{14}$.
- for an object t such that there is no property a with $f(g(\{a\})) = f(\{t\})$, a pair $C = (\{t\}, \emptyset)$ is created and linked to the lower classes C_j such that $t \in g(C_j)$. Let C^{ext} be the pair extended considering that the link corresponds to an inclusion relation: if C_Y^{ext} is the right part of the pair, C_Y^{ext} contains the right parts of higher pairs, and as C^{ext} has no lower pairs, $C_X^{ext} = \{t\}$.

  For each property a_i of t, a pair $(g(\{a_i\}), f(g(\{a_i\})))$ was built, with $t \in g(\{a_i\})$. This pair is higher than C^{ext} after the linkage, thus (considering all a_i) $f(\{t\}) \subseteq C_Y^{ext}$. Furthermore, the elements higher than C^{et} are pairs $C^j = (C_X^j, C_Y^j)$ such that $t \in C_X^j$. For each of them, $f(\{t\}) \supseteq C_Y^j$, thus $f(\{t\})$ contains their union C_Y^{ext} and finally $C_Y^{ext} = f(\{t\})$.

[2]In this paper "leaf" is used in the common object-oriented meaning, for classes that have no sub-classes. In graph theory, they are leaves while considering the inheritance link oriented from classes to sub-classes.

	a	b	c	d	e
t1	X				
t2		X			
t3	X	X			
t4			X		
t5				X	
t6			X	X	
t7	X	X	X	X	
t8	X	X	X	X	
t9	X	X	X	X	X

FIGURE 13. Relation R4

The result for the example appears in Figure 12, where object set and property sets are simplified.

5.3. CHARACTERIZATION

Theorem 5.1. *The algorithm of Chen and Lee builds the following graph:*

- *Vertices are:*
  - *(first form) all pairs of O_P, and*
  - *(second form) all pairs $(\{t\}, f(\{t\}))$, $t \in E$, such that there is no pair $(-, f(\{t\})$ in O_P ;*
- *links are:*
  - *vertices of the form 1 are linked according to the transitive reduction of $\subseteq$ between extended left parts, thus like they would be linked in the Galois lattice;*
  - *vertices of the form 2 are not linked together*
  - *vertices of the form 2 are linked to lower vertices of form 1, according to $\subseteq$ between extended left parts.*

The proof was given in the previous section.

Let us look now at this result. For a vertex of the second form, $\{t\}$ may be a closed set (such that $g(f(\{t\})) = \{t\}$) but not necessarily. When it is not the case, $(\{t\}, f(\{t\})) \notin \mathcal{S}$, and some vertices of the Galois sub-hierarchy $\mathcal{S}$ are not represented in the graph. They are indeed split into several elements of the form 2. Some links are missing to represent the entire $\subseteq$ relation between right parts, more precisely, possible links between vertices of form 2, and links from a vertex of form 1 to a vertex of form 2.

We give an example where the subtype hierarchy is not exactly the Galois sub-hierarchy. The relation of Figure 13 is ordered by the algorithm into the subtype hierarchy of Figure 14.

For the relation of Figure 13, step 1 produces the pairs below:

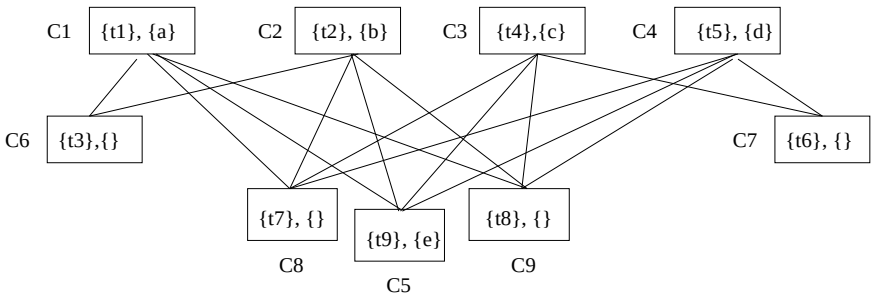

FIGURE 14. Subtype hierarchy for Relation R4

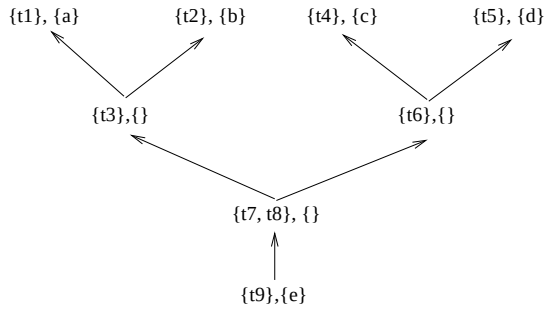

FIGURE 15. Galois sub-hierarchy for Relation R4

```
X1={t1, t3, t7, t8, t9}   Y'1=C1={a}
X2={t2, t3, t7, t8, t9}   Y'2=C2={b}
X3={t4, t6, t7, t8, t9}   Y'3=C3={c}
X4={t5, t6, t7, t8, t9}   Y'4=C4={d}
X5={t9}                   Y'5=C5={e}
```

These pairs define four incomparable classes. During step 2.3, four classes are created, respectively for t_3, t_6, t_7 and t_8, since these objects belong to the (extended) left part of several incomparable classes. For example, t_3 is in the (extended) left part of C_1 and C_2. As a second example t_7 is in the (extended) left part of C_1, C_2, C_3 and C_4. Comparing the subtype hierarchy to the Galois sub-hierarchy (Figure 15), we see that the pair of closed sets of the Galois sub-hierarchy $(\{t7, t8\}, \{a, b, c, d\})$ is split into two vertices (C_8 et C_9). Some inclusion relations between property sets are not represented by links in the subtype hierarchy. For example, between the vertices of form 2 C_8 and C_6, or from the vertex C_9 of form 1 to C_8 of form 2.

	a	b	c	d	e
o1	X	X	X		X
o2	X	X			X
o3		X			
o4	X		X	X	X

FIGURE 16. Relation R5

6. MOORE ET AL. ALGORITHM [14]

6.1. NOTATIONS AND AIMS

This algorithm takes as input an object set E, a property set F, and a binary relation $\mathcal{R}$ between them. The result is a directed acyclic graph whose vertices are labeled by objects and properties. In this graph, a vertex v can be associated with each object, such that properties of the label of v joined with the properties of higher vertices ('superclasses') labels is exactly the property set of the object. This graph satisfies the following properties:

- *maximal factorization:* a property appears on a unique vertex of the graph;
- *minimal number of internal node*;
- *inheritance links consistency:* all objects that inherit from a class C also inherit from a class D, then C must be a subclass of D;
- *the graph does not contain transitivity edges*;
- *objects must be represented by leaves.*

The authors consider the last point as a less important criterion that may be relaxed, by slight modifications of the algorithm. These criteria, as noticed by the authors, uniquely define the resulting hierarchy. Our characterization is an alternative proof of that point.

6.2. ALGORITHM DESCRIPTION

The algorithm is described, as in [14], using the relation presented in Figure 16.
Step 1. $\mathcal{R}$ is first represented as a directed bipartite graph, the "grouping graph" that we denote by G_g. The two set of vertices are E and F. For $e \in E$ and $f \in F$, fe is an edge of G_g iff $(e, f) \in \mathcal{R}$.

Obviously, in this graph, if we consider a vertex o of E, and its predecessors in F, we obtain a pair $(\{o\}, f(\{o\}))$; respectively, if we consider a vertex a of F and its successors in E, we obtain a pair $(g(\{a\}), \{a\})$.
Step 2. The set of all $g(\{a\}), \forall a \in F$, is built, and its elements are the labels of new nodes added in order to represent factorizations. An edge is introduced from the vertex labelled by a to the vertex labelled by $g(\{a\})$. For any $o \in g(\{a\})$, the edge ao is removed. Each edge represents now either a pair $(g(\{a\}), \{a\})$, or a

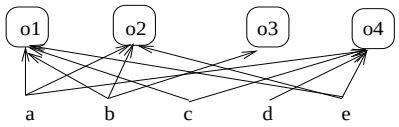

FIGURE 17. The grouping graph associated with Relation R5

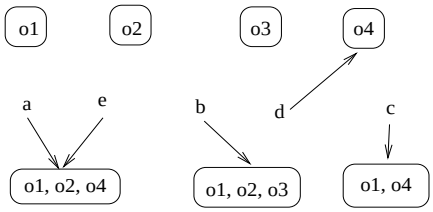

FIGURE 18. The mapping graph for Relation R5

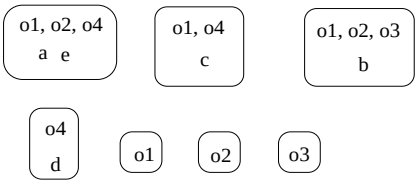

FIGURE 19. Classes for Relation R5

pair $(\{o\}, y)$, $y \in f'(\{o\}))$, where $f'(\{o\}) = f(\{o\}) \setminus \{a/\exists o' \neq o \text{ s.t. } a \in f(o')\}$. $f'(\{o\})$ represents $f(\{o\})$ restricted to properties that are owned only by o (see Figure 18). $f'(\{o\})$ is thus a closed set simplified according to properties.

Classes of the output hierarchy are obtained by completing the label of nodes which represent factorizations or input objects (such vertices are in boxes of Figure 18). Such a completion consists in adding to the node label the properties which are at the origin of the edges ending at the node. A node representing a factorization corresponds now to at least a pair $(g(\{a\}), f'(g(\{a\})))$ where $f'(g(\{a\})))$ is the result of the simplification of $f(g(\{a\})))$. A node representing an input object corresponds now to at least a pair $(\{o\}, f'(\{o\}))$. If an edge ends at the node, $f'(\{o\})$ is not empty. No duplicate pairs are produced during the construction. Figure 19 shows the result.

Step 3. Classes are ordered with respect to inclusion of the label's objects. The hierarchy contains only the transitive reduction of this order (Figure 20).

6.3. CHARACTERIZATION

We begin with a proposition, then prove two characterization theorems.

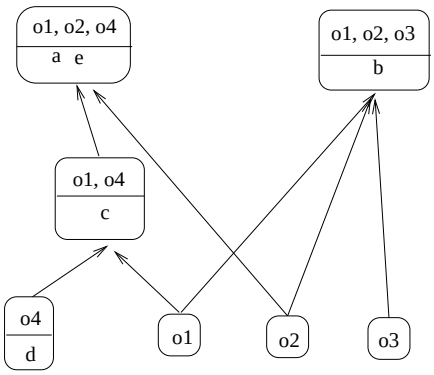

FIGURE 20. Class hierarchy for Relation R5

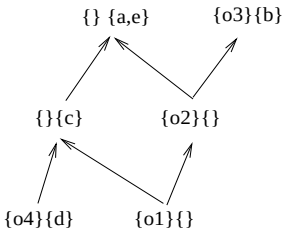

FIGURE 21. Galois sub-hierarchy of Relation R5

Proposition 6.1. *If, for all object pairs (e', e''), $f(e')$ and $f(e'')$ are incomparable, then for any $e \in E$ we have $h_E(\{e\} = \{e\}$.*

Proof. Let us assume the contrary. If there is $e \in E$ with $h_E(\{e\} \neq \{e\})$, then $\exists e' \neq e$ such that $e' \in h_E(\{e\})$, that is $f(e') \supset f(e)$.

□

Theorem 6.2. *The Moore and Clement algorithm builds the Galois sub-hierarchy associated with $\mathcal{R} \bigcup \{(o, a_o)/o \in E\}$, where a_o is a property added to each object and which is not owned by any other object (this property disappears after building).*

Proof. The conditions of Proposition 6.1 are satisfied, since, thanks to additonal properties, two objects can not have comparable property sets with respect to inclusion. It follows from the proposition and the algorithm description that all pairs $\{(h(\{e\}), f(\{e\}))/e \in E\}$ and all pairs $(g(\{a\}), f'(g(\{a\})))$ are processed. □

The difference with the Galois sub-hierarchy of $\mathcal{R}$ comes from the constraint requiring objects to be leaves of the output hierarchy.

Theorem 6.3. *The Moore and Clement algorithm builds all vertices of the Galois sub-hierarchy of $\mathcal{R}$ which have the forms 1 and 2 (see Chen and Lee characterization). Moreover, pairs $(\{o\}, f'(\{o\}))$, where there is one pair $(-, f(\{o\}))$ in O_P,*

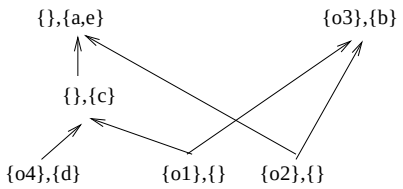

FIGURE 22. Chen and Lee Subtype hierarchy for Relation R5

are added. These pairs are ordered with respect to inclusion of their object sets (left part).

The proof is easily deduced from the algorithm description.

Figure 21 shows the Galois sub-hierarchy for the same Relation R5, while Figure 22 shows the subtype hierarchy that would be built using the Chen and Lee algorithm. Only the Galois sub-hierarchy orders correctly the object $o2$ with respect to the inclusion order between property sets : $o2$ is indeed higher than $o1$ (a superclass). Concerning $o3$, this object is represented by a node which is not a leaf in the Galois sub-hierarchy and the Chen and Lee subtype hierarchy, but is a leaf in the Moore and Clement hierarchy.

7. COOK ALGORITHM [4]

7.1. NOTATIONS AND AIMS

This algorithm is part of a method for the construction of an interface hierarchy reflecting a Smalltalk-80 class hierarchy and highlighting some controversial design features. Each class is associated with its "protocol", which contains the valid selectors (or method names) for the class. This protocol is a very simple interface for the class. The algorithm is assumed to build a protocol hierarchy, more precisely a hierarchy where protocols are organized with respect to inclusion.

We will follow the algorithm on the short example below, taken from [4]. Three classes A, B and C are given together with their protocols.

```
protocol(A)={isEmpty, at}
protocol(B)={isEmpty, at, add, remove}
protocol(C)={isEmpty, add, remove, first}
```

These protocols are organized so as to respect inclusion and to make selector sharing explicit. In our notation, E is the set of classes, F the set of selectors, and for a class C, $f(C)$ is the protocol associated with C.

7.2. ALGORITHM DESCRIPTION

Step 0. The relation "inverse" is first computed, giving the following result:

```
inverse(isEmpty)={A, B, C}
inverse(at)={A, B}
inverse(add)={B, C}
inverse(remove)={B, C}
```

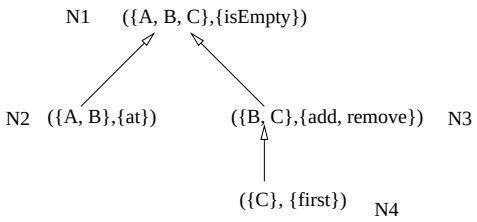

FIGURE 23. Pairs of closed sets ordered at step 2

```
inverse(first)={C}
```

In our notations, $\forall x \in F$, $inverse(x) = g(\{x\})$. The effect of this step is to produce $g(\{s\})$, for each $s \in F$.

Step 1. Pairs of closed sets $(inverse(s),\ hierarchy(inverse(s)))$ for each $s \in F$, (in our notation $(g(\{s\}), f'(g(\{s\})))$), are produced through the computation given below of the mapping *hierarchy* :

$hierarchy : \mathcal{P}(E) \rightarrow \mathcal{P}(F)$

For every $s \in F$, $hierarchy(inverse(s)) \leftarrow s$ (s is added to $hierarchy(inverse(s))$).

This algorithm, for each closed set X of E such that there is $a \in F$ with $X = g(\{a\})$, gives indeed the set $\{s\ s.t.\ g(\{s\}) = X\}$.

For the example, the produced pairs are the following:

```
({A, B, C},{isEmpty})
({A, B},{at})
({B, C},{add, remove})
({C}, {first})
```

No precise algorithm is given for the next steps 2 and 3.

Step 2. Pairs produced at step 1 are ordered with respect to inclusion between left members (class sets). Figure 23 shows the resulting hierarchy.

Step 3. The simplification according to objects applies, since a class name is associated with the «minimum of all nodes in which the class appears (a minimum node may need to be added for it)» [4]. For example, A and C are associated respectively with nodes $N2$ and $N4$. However, for classes such as B, which appear in several minimal nodes ($N2$ and $N3$), this description may have several interpretations. We will assume that a new node is created for each problematic class, and linked to the minimal nodes where the class appears. There is no argument specifying whether these new nodes are or not linked together.

7.3. CHARACTERIZATION

The characterization theorem and the corollary are given when the above assumption holds.

Theorem 7.1. *All pairs of O_P, simplified according to properties, that is all pairs $(g(\{a\}), f'(g(\{a\})))$ where $a \in F$, are produced at step 1 (form 1). Moreover, for each class C such that there is no $(-, f(\{C\}))$ in O_P, a pair $(\{C\}, \{\})$ is created (form 2). All these pairs are ordered by inclusion of their left part.*

Proof. The proof is deduced for pairs of form 1 from the algorithm description. Concerning pairs of form 2, the description of step 2.3 of the Chen and Lee algorithm applies. □

8. Mineau et al. algorithm [13]

We now present an algorithm published in the field of knowledge representation, where the same problems arise.

8.1. Notations and aims

The objects to be hierarchically organized are represented by conceptual graphs, which are graphs composed of two kinds of nodes, concept[3] nodes and relation nodes. In this particular case, relations are supposed to have an arity equal to two, thus relating two concepts. A graph of an object is decomposed into a set of triplets $< C_1, r, C_2 >$, where C_1 and C_2 are concepts and r a relation. A triplet $< C_1, r, C_2 >$ is generalized by replacing C_1, C_2 and/or r, by the character '?', giving seven new incomplete triplets. In our context, triplets will be the properties, and it is notable that they are related by some kind of partial generalization order. In this generalization order $<?, parts, ?>$ is above $< elephant, parts, ? >$. Each incomplete triplet IT, that is containing one or more '?', is associated with one or several "instanciation lists", one for each object O of whom at least one complete triplet may instantiate IT. Each instanciation list consists of one or several "element lists", one for each complete triplet of O which instantiates IT and these "element lists" describe the different ways the characters '?' may be replaced to obtain the complete triplets.

Let us take an example. We have two objects O_a and O_b, each composed of two complete triplets. $O_a = \{< a, r, 1 > < a, r, 2 >\}$, $O_b = \{< b, r, 1 > < b, r, 2 >\}$. The instantiation lists of the incomplete triplet $<?, r, ?>$ are $((a1)(a2))$ and $((b1)(b2))$, while the instantiation lists of $<?, ?, 1>$ are $((ar))$ and $((br))$.

To simplify further discussion, we introduce the "extended instantiation list" of an incomplete triplet, which is the set of all the complete triplets that instantiate the incomplete triplet (the extended list of instantiations of a complete triplet will be composed of the sole complete triplet). In our previous example, the extended instantiation list of $<?, r, ?>$ is $(< a, r, 1 > < a, r, 2 > < b, r, 1 > < b, r, 2 >)$ while the extended instantiation list of $<?, ?, 1>$ is $(< a, r, 1 > < b, r, 1 >)$ The generalisation order upon triplets is isomorphic to the order of inclusion upon their extended instantiation lists.

Some triplets are useful generalizations[4], while the others are superfluous. A triplet T is useless if and only if it has at least one occurrence of a '?' which is instantiated by the same symbol through all the complete triplets of the instantiation list of T. We will call such a '?' useless.

[3]Concepts in terms of conceptual graphs and unrelated to concepts of the Galois lattice.

[4]The authors call them "maximal" triplets.

In our previous example,$<?,?,1>$ is useless because its second '?' is always instantiated by 'r'.

Proposition 8.1. *A closed set of properties is determined by its useful triplets.*

Proof. First, there cannot be any useless triplet in a closed set of properties without its associated useful triplet, i.e. the useful triplet we get when we replace the useless '?' by their appropriate symbol(s). Both the useless triplet and its associated useful triplet have indeed the same extended instantiation list.

Conversely, whenever a useful triplet T belongs to a closed set of properties, all the ancestors of T in the property order also belong to the unsimplified closed set of properties, because of the way the generalization order on properties is built. □

In the previous example, $<?,?,1>$ will belong to the same closed set of properties as $<?,r,1>$.

This property will allow a new simplification of the closed sets of properties, keeping only useful properties.

One just adds all the ascendants (in the triplet order) of any useful triplet of a simplified closed set of properties to get the whole closed set.

We denote by f'' the composition of the two simplifications that get rid of both the inherited and the useless properties.

8.2. Algorithm description

We limit our description to the creation of concepts (pairs of closed sets). The paper, however, contains also a detailed algorithm building the transitive reduction of the inclusion order.

An intersection matrix IM represents the relation $\mathcal{R}$. E is the set of the objects represented by the conceptual graphs. F is the set of triplets. For a concept C, we shall note $C.L$ its left part, and $C.R$ its right part. For an incomplete triplet, p, we shall note $IL(p)$ its instantiation list.

For each property (triplet) $p \in F$ do {
  $N = g(\{p\})$
  if there is not yet a concept C such that $C.L = N$
    then create a concept C with $C.L = N$ and $C.R = \emptyset$
  let the concept C such that $C.L = N$
  if p is a complete triplet
    then $C.R \leftarrow C.R \cup \{p\}$
    else if p is incomplete and useful
      then $C.R \leftarrow C.R \cup \{p, IL(p)\}$
}

We shall take an example which is drawn from Relation R5, where a property a, is replaced by a complete triplet $< xa, ra, ya >$. We consider four objects with three complete triplets. $O1 = \{< xa, ra, ya >, < xb, rb, yb >, < xc, rc, yc >\}$, $O2$

	O1	O2	O3	O4
<xa, ra, ya>	x	x		x
<xb, rb, yb>	x	x	x	
<xc, rc,yc>	x			x
<?, ra, ya>	x	x		x
<xa, ?, ya>	x	x		x
<xa, ra, ?>	x	x		x
<?, rb, yb>	x	x	x	
<xb ?, yb>	x	x	x	
<xb, rb, ?>	x	x	x	
<?, rc, yc>	x			x
<xc ?, yc>	x			x
<xc, rc, ?>	x			x
<?, ?, ya>	x	x		x
<?, ra, ? >	x	x		x
<xa, ?, ?>	x	x		x
<?, ?, yb>	x	x	x	
<?, rb, ? >	x	x	x	
<xb, ?, ?>	x	x	x	
<?, ?, yc>	x			x
<?, rc, ? >	x			x
<xc, ?, ?>	x			x
<? ,?, ?>	x	x	x	x

FIGURE 24. Relation R6

$= \{< xa, ra, ya >, < xb, rb, yb >\}$, $O3 = \{< xb, rb, yb >\}$ $O4 = \{< xa, ra, ya >, < xc, rc, yc >\}$.

We can remark that the only incomplete useful triplet is $<?, ?, ? >$. Three concepts $C1$, $C2$ and $C3$ will be created, with[5] $C1.L = \{O1, O2, O4\}$, $C1.R = \{< xa, ra, ya >\}$, $C2.L = \{O1, O2, O3\}$,$C2.R = \{< xb, rb, yb >\}$, $C3.L = \{O1, O4\}$, $C3.R = \{< xc, rc, yc >\}$ $C4.L = \{O1, O2, O3, O4\}$, $C4.R = \{<?, ?, ? >\}$.

As expected, several closed sets of O_C are not built: the pair of closed sets $(\{O1\}, \{< xa, ra, ya >, < xb, rb, yb >, < xc, rc, yc >\})$ or the pair corresponding to $O2$.

8.3. CHARACTERIZATION

Theorem 8.2. *The algorithm produces all pairs of O_P, simplified by the removal of both useless generalizations and inherited properties, that is all the pairs of closed sets $(g(\{p\}), f''(g(\{p\})))$, with $p \in F$.*

[5] $IL(p)$ is not written out.

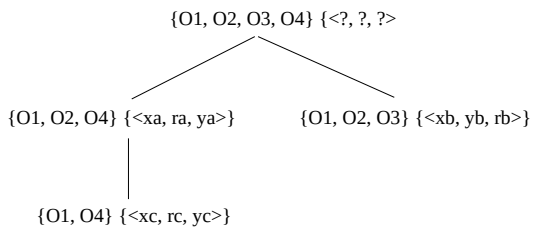

FIGURE 25. The graph of Relation R6

This comes easily from the algorithm description. The fact that the algorithm builds only these pairs is mainly due to the aimed application. The objects to be organized are not supposed to be comparable: neither a generalization nor a specialization of an elephant is not comparable to any other animal.

9. CONCLUSION

Firstly, we summarize the main elements of our study. As we have seen, none of the four algorithms studied here can be used to obtain the whole Galois sub-hierarchy. All of them produce the pairs of O_P, but diverge about the computation of pairs that "take the place of" more or less the pairs of $O_C \backslash O_P$. The next theorem recalls summarizes this.

Theorem 9.1.
- *The studied algorithms produce all the pairs of* O_P.
- *The Chen and Lee algorithm and the Cook algorithm are equivalent. They add to the initial pairs the pairs* $(\{t\}, f(\{t\})$ $t \in E$, *such that there is no pair* $(-, f(\{t\}))$ *in* O_P.
- *The Moore and Clement algorithm adds, to this common base, the pairs* $(\{o\}, f'(\{o\}))$, *such that* $(-, f(\{o\}) \in O_P$.
- *In all algorithms, pairs are ordered with respect to the inclusion of their left members (object sets).*

Such theorems allow to figure the results of the algorithms without running them (and that is indeed the way we draw the hierarchies of the stream example).

The similarities between these algorithms renew the interest of the Galois lattice, as a "natural" structure to find inheritance hierarchies.

It would be misleading not to mention that the use of the Galois lattice suffers from a few drawbacks. Firstly, it is a complex structure, in the worst case, exponential in the number of input objects and properties. This complexity is fortunately reduced in the Galois sub-hierarchy which has a number of vertices that can not exceed the sum of the number of properties and the number of input objects. Secondly, this structure implies multiple inheritance, requiring adjustments for languages that have only single inheritance. Finally, a maximal factorization, although globally interesting, is not always the better choice in design terms. Netherveless, we think that the advantages are sufficiently numerous to justify our

choice. The fact that the structure is formally defined allows a good and easy characterization of all operation results. The maximal factorization induces maximal reuse, and any change on a property is confined to the class that declares the property. The structure is the minimal structure that ensures maximal factorization. As a further criticism, this model may not seem accurate enough, as far as properties are concerned, since it does not take into account overloading and overriding features. For an extension to a more realistic model, the reader can refer to [6, 7]. Several high-scale experiences are also reported in [10, 11].

Note. We are involved, in the framework of an industrial project (supported by the France Télécom Research and Development center) in the construction of a class hierarchy development platform [3] for object languages such as C++, Java or Eiffel. This platform is dedicated to the Galois sub-hierarchy construction and evolution.

Authors would like to thank the anonymous referees for their meticulous review.

References

[1] Barbut M. and Monjardet B. (1970) *Ordre et classification : Algèbre et combinatoire*. Hachette.

[2] Chen J.- B. and Lee. S. C. (1996) Generation and reorganization of subtype hierarchies. *Journal of Object Oriented Programming*, 8(8).

[3] Chevalier N., Dao M., Dony C., Huchard M., Leblanc H., and Libourel T. (1999) An environment for building and maintaining class hierarchies. *ECOOP 99 Workshop - Object-Oriented Architectural Evolution - june 1999.*

[4] Cook W. R. (1992) Interfaces and Specifications for the Smalltalk-80 Collection Classes. In *Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'92*, pages 1–15.

[5] Davey B. A. and Priestley H. A. (1990) *Introduction to Lattices and Orders*. Cambridge University Press.

[6] Dicky H., Dony C., Huchard M., and Libourel T. (1996) On automatic class insertion with overloading. *Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'96.*, 31(10):251–267.

[7] Godin R. and Mili H. (1993) Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. *Special issue of Sigplan Notice - Proceedings of ACM OOPSLA'93*, 28(10):394–410.

[8] Godin R., Mili H., Mineau G., and Missaoui R. (1995) Conceptual Clustering methods based on Galois lattices and applications. *Revue d'intelligence artificielle*, 9(2).

[9] Godin R., Mineau G., and Missaoui R. (1995) Incremental structuring of knowledge bases. *Proceedings of International KRUSE symposium: Knowledge Retrieval, Use, and Storage for Efficiency Springer-Verlag's Lecture Notes in Artificial Intelligence*, 9(2):179–198.

[10] Godin R., Mili H., Mineau G., Missaoui R., Arfi A., and Chau T.-T. (1998) Design of Class Hierarchies Based on Concept (Galois) Lattices. *Theory And Practice of Object Systems*, 4(2).

[11] Huchard M. and Leblanc H. (2000) Computing Interfaces in Java. *proceedings of IEE International conference on Automated Software Engineering (ASE'2000)*, pp 317-320, 11-15 September, Grenoble, France.

[12] Lieberherr K. J., Bergstein P., Silva-Lepe I. (1991) From objects to classes: Algorithms for optimal object-oriented design *Journal of Software Engineering*, pages 205–228

[13] Mineau G., Gecsei J., and Godin R. (1990) Structuring Knowledge Bases Using Automatic Learning. *Proceedings of the sixth International Conference on Data Engineering*, pages 274–280.
[14] Moore I. and Clement T. (1996) A Simple and Efficient Algorithm for Inferring Inheritance Hierarchies. *TOOLS Europe 1996 Proceedings, Prentice-Hall.*
[15] Stroustrup B. (1998) *The C++ programming language, third edition.* Addison–Wesley.
[16] Wille R. (1982) Restructuring lattice theory: An approach based on hierarchies of concepts. *Ordered Sets, in I. Rivals (Eds)*, 23.

Communicated by (The editor will be set by the publisher).
June 2000.