A Logical Generalization of Formal Concept Analysis

Sébastien Ferré and Olivier Ridoux

IRISA, Campus Universitaire de Beaulieu, 35042 RENNES cedex,
{ferre,ridoux}@irisa.fr

Abstract. We propose a generalization of Formal Concept Analysis (FCA) in which sets of attributes are replaced by expressions of an almost arbitrary logic. We prove that all FCA can be reconstructed on this basis. We show that from any logic that is used in place of sets of attributes can be derived a contextualized logic that takes into account the formal context and that is isomorphic to the concept lattice. We then justify the generalization of FCA compared with existing extensions and in the perspective of its application to information systems.

1 Introduction

The origin of this work is the search for flexible organisations for managing, updating, querying, or navigating in data. In this context, several roles are played by possibly different people: designer, administrator, and end-user. Hierarchical organisations are not flexible, and updating, querying and navigation are difficult to conciliate (see for instance the view update problem in data-bases). The literature shows that *Formal Concept Analysis* (*FCA*) is a good candidate for supporting querying and navigation. However, we feel it is not flexible enough as far as the description of data is concerned, and the literature on FCA insists more on analysing a given context than on managing contexts. In this article, we present an extension to FCA that allows for flexible descriptions, and we sketch an organisation that handles updating, querying, or navigation in data at the same time.

Given a *formal context* $(\mathcal{O}, \mathcal{A}, I)$, where $\mathcal{O}$ is a set of *objects*, $\mathcal{A}$ is a set of *attributes*, I is a relation between objects and attributes (i.e., a subset of $\mathcal{O} \times \mathcal{A}$), *Formal Concept Analysis* (FCA [Wil82,GW99a], Chapter 11 in [DP90a]) defines *concepts* as maximal sets of objects that share the same attributes. More formally, a concept is a pair (O, A) where O is a subset of $\mathcal{O}$, A is a subset of $\mathcal{A}$, such that the following relation holds:

$\sigma(O) = A$ and $\tau(A) = O$

where $\sigma : 2^{\mathcal{O}} \to 2^{\mathcal{A}}$ $\quad \sigma(O) := \{a \in \mathcal{A} \mid \forall o \in O : (o, a) \in I\}$

and $\tau : 2^{\mathcal{A}} \to 2^{\mathcal{O}}$ $\quad \tau(A) := \{o \in \mathcal{O} \mid \forall a \in A : (o, a) \in I\}$

The application σ returns for every set of objects the set of attributes that are shared by all these objects. The application τ returns for every set of attributes the set of objects that owns all these attributes. The O part of a concept is called

its *extent*, and the A part is called its *intent*. The fundamental theorem of FCA is that the set of all concepts that can be built on a given formal context forms a complete lattice when it is ordered by set-inclusion of concept extensions. In fact, the pair $\sigma - \tau$ forms a Galois connection.

FCA has received attention for its application in many domains such as in software engineering [Sne98,Lin95,KS94]. The interest of FCA as a navigation tool in general has also been recognized [GMA93,Lin95,VW95].

The various application domains bring the need for more sophisticated formal contexts than the mere presence/absence of attributes. For instance, many application domains use numerical values (e.g., lengths, prices, ages), and the need to express negation and disjunction is often felt. In a much more specialized scope, it is imaginable to use the type of software components instead of attributes. Several enrichments to the attribute structure have been proposed: e.g., many valued attributes [GW99a], and first-order terms [CM98]. However, not a single extended FCA framework covers all the concrete domains, and can pretend covering all the concrete domains to come. So, we propose to construct a more general framework for concept analysis, Logical Concept Analysis (LCA), in which the logic of attributes becomes a parameter. This will allow for instantiating the general framework by merely filling in a dedicated logic.

In the rest of this article, we will refer to the original form of concept analysis by FCA or *standard* CA, while we will refer to the concept analysis as it is developed in this article by LCA or *generalized* CA. The term CA will be used to talk about both forms at once. Section 2 goes from standard CA (FCA) to its generalized form, i.e., LCA. Section 3 defines how a contextualized logic can be derived from a logical context. Section 4 studies the contribution of LCA compared with existing extensions of FCA. Section 5 explains how LCA can provide an interesting organization framework for information systems in which objects are described by logical formulas.

In this article, results are given without proofs. They can be found in [FR99].

2 From FCA to LCA

2.1 General presentation

We start on reformulating standard CA so that it can be generalized more naturally. The first step of this reformulation consists in replacing context $(\mathcal{O}, \mathcal{A}, I)$ by $(\mathcal{O}, 2^{\mathcal{A}}, i)$, where $2^{\mathcal{A}}$ is the power-set of $\mathcal{A}$ and i is a mapping from $\mathcal{O}$ to $2^{\mathcal{A}}$ defined by $i(o) := \{a \in \mathcal{A} \mid (o,a) \in I\}$. No information is lost, nor added, and both representations are equivalent: $(o,a) \in I \iff i(o) \supseteq \{a\}$.

Then, if applications σ and τ are reformulated with mapping i rather than relation I, the following equalities are easily obtained:

$$(1) \quad \sigma(O) = \bigcap_{o \in O} i(o) \qquad\qquad (2) \quad \tau(A) = \{o \in \mathcal{O} \mid i(o) \supseteq A\}$$

The rest of FCA theory is kept unchanged. Now, by carefully studying proofs in FCA, and considering the connection between algebraic structures and lat-

tices [DP90a], we observe that a necessary and sufficient condition to FCA is that $\langle 2^{\mathcal{A}}; \supseteq \rangle$ be a lattice whose *supremum* (least upper bound) is $\cap$, and *infimum* (greatest lower bound) is $\cup$. Then, $2^{\mathcal{A}}$ can be considered as a logic where $\supseteq$ is the deduction relation, $\cap$ is disjunction, and $\cup$ is conjunction.

From this interpretation of FCA, it becomes natural to generalize it by replacing the power-set of the set of attributes $2^{\mathcal{A}}$ by an arbitrary set of formulas $\mathcal{L}$, to which are associated a deduction relation $\dot{\models}$, a disjunctive operation $\dot{\vee}$, and a conjunctive operation $\dot{\wedge}$ (dots on symbols are only aimed at differentiating them from those of the meta-language used in this document). So as to keep FCA results in its generalized form (LCA), it is necessary and sufficient that $\langle \mathcal{L}; \dot{\models} \rangle$ be a lattice whose supremum and infimum are respectively $\dot{\vee}$ and $\dot{\wedge}$.

Example 1. An example of a logic usable in LCA is propositional logic, $\mathcal{P}$. On the syntactic side, the set of propositions $\mathcal{P}$ contains atomic propositions (taken in a set $\mathcal{A}$), formulas 0 and 1, and is closed under binary connectors $\wedge$ and $\vee$ and unary connector $\neg$. On a semantical side, interpretations are subsets of the set of atomic propositions $\mathcal{A}$. Logic $\langle \mathcal{P}; \vee, \wedge, \models \rangle$ satisfies LCA conditions, because its semantics $\langle 2^{2^{\mathcal{A}}}; \subseteq \rangle$ is a lattice whose supremum is $\cup$, and infimum is $\cap$.

2.2 Context and Galois connection

In this section, we apply the idea introduced in the last section to give a formulation of CA that is generalized to an almost arbitrary logic: Logical Concept Analysis (LCA). Transposition from FCA to LCA consists in reformulating each occurrence of I with i, and replacing $\supseteq$, $\cap$, and $\cup$ respectively by $\dot{\models}$, $\dot{\vee}$, and $\dot{\wedge}$. Proofs are obtained from the chapter 11 in [DP90a], and by applying the above transposition.

Definition 1 (context) *A* (formal) context *is a triple* $(\mathcal{O}, \mathcal{L}, i)$ *where:*

- $\mathcal{O}$ *is a finite set of objects,*
- $\langle \mathcal{L}; \dot{\models} \rangle$ *is a (possibly infinite) lattice of formulas, whose supremum is* $\dot{\vee}$*, and whose infimum is* $\dot{\wedge}$*;* $\mathcal{L}$ *denotes a logic whose deduction relation is* $\dot{\models}$*, and whose disjunctive and conjunctive operations are respectively* $\dot{\vee}$ *and* $\dot{\wedge}$*,*
- i *is a mapping from* $\mathcal{O}$ *to* $\mathcal{L}$ *that associates to each object a formula that describes the object.*

If $f \dot{\models} g$ and $g \dot{\models} f$, f and g are called logically equivalent; we will consider them as different representations of the same equivalence class, and in fact we will consider that elements of $\mathcal{L}$ are the equivalence classes. Also, operations $\dot{\vee}$ and $\dot{\wedge}$ can be either connectors (as in propositional logic $\mathcal{P}$), either algebraic operations (as in the so-called cube logic [CM98]). We just assume, for practical reasons, that operations $\dot{\models}$, $\dot{\wedge}$, and $\dot{\vee}$ are computable. A word of caution is necessary. The logic $\mathcal{L}$ is completely independent from the set of objects. If the logic $\mathcal{L}$ is rich enough to have quantifiers, the individuals upon which formulas are quantified can never be the objects of the formal context. For instance, if $i(o) \dot{=} \forall x.p(x)$, this

tells nothing about the fact that all objects have property p. It only says that if $i(o')\dot{=}p(1)$ then o can be considered as being more generic than o'. A good interpretation of a description like $i(o)\dot{=}\forall x.p(x)$ is to read it as a polymorphic type declaration. According to the Curry-Howard isomorphism, types can be considered as formulas.

FCA derives from a context two applications σ and τ that form a Galois connection. Here, the Galois connection is between sets of objects, and formulas. Its definition is obtained by transposing equalities (1) and (2):

Definition 2 *Let $(\mathcal{O},\mathcal{L},i)$ be a context, $O\subseteq\mathcal{O}$, and $f\in\mathcal{L}$.*
$\sigma: 2^{\mathcal{O}}\to\mathcal{L}, \quad \sigma(O):=\dot{\bigvee}_{o\in O} i(o) \qquad \tau:\mathcal{L}\to 2^{\mathcal{O}}, \quad \tau(f):=\{o\in\mathcal{O}\mid i(o)\dot{\models} f\}$

Lemma 1 *Let $(\mathcal{O},\mathcal{L},i)$ be a context, $O,O_j\subseteq\mathcal{O}$, and $f,f_j\in\mathcal{L}$ for all $j\in J$, where J is a finite set of indices.*

(i) $O\subseteq\tau(\sigma(O))$	*(i')* $\sigma(\tau(f))\dot{\models} f$
(ii) $O_1\subseteq O_2\Longrightarrow\sigma(O_1)\dot{\models}\sigma(O_2)$	*(ii')* $f_1\dot{\models} f_2\Longrightarrow\tau(f_1)\subseteq\tau(f_2)$
(iii) $\sigma(O)\dot{=}\sigma(\tau(\sigma(O)))$	*(iii')* $\tau(f)=\tau(\sigma(\tau(f)))$
(iv) $\sigma(\bigcup_{j\in J}O_j)\dot{=}\dot{\bigvee}_{j\in J}\sigma(O_j)$	*(iv')* $\tau(\dot{\bigwedge}_{j\in J}f_j)=\bigcap_{j\in J}\tau(f_j)$
(v) $\forall o\in\mathcal{O}:\sigma(\tau(i(o)))\dot{=}i(o)$	

Results expressed in this lemma are a transposition from those of standard CA (see Lemma 11.4 in [DP90b]), except Lemma 1.(v) whose demonstration uses the actual definitions of σ and τ.

Example 2. An example of a formal context will illustrate the rest of our development on LCA. Context K_{ex} is deliberately small and simple as it is aimed at illustrating theoretic notions, and not at showing a realistic application of LCA. The logic used in this context is propositional logic $\mathcal{P}$ with a set of atomic propositions $\mathcal{A}=\{a,b,c\}$. We define context K_{ex} by $(\mathcal{O}_{ex},\mathcal{P},i_{ex})$, where $\mathcal{O}_{ex}=\{x,y,z\}$, and where $i_{ex}=\{x\mapsto a, y\mapsto b, z\mapsto c\wedge(a\vee b)\}$.

2.3 Concepts

In this section, we show that using our definitions of a context and Galois connection σ-τ, we retrieve all existing results about concepts. First, we recall the definition of concepts (see Chapter 11 in [DP90a]), just adapting notations.

Definition 3 (concept) *In a context $(\mathcal{O},\mathcal{L},i)$, a* concept *is a pair $c=(O,f)$ where $O\subseteq\mathcal{O}$, and $f\in\mathcal{L}$, such that $\sigma(O)\dot{=}f$ and $\tau(f)=O$. The set of objects O is the concept* extent *(written $ext(c)$), whereas formula f is its* intent *(written $int(c)$).*

The main difference with standard CA is that the intent is now a formula of the logic $\mathcal{L}$. The set of all concepts that can be built in a context $(\mathcal{O},\mathcal{L},i)$ is denoted by $\mathcal{C}(\mathcal{O},\mathcal{L},i)$, and is partially ordered by $\leq^c$ defined as follows.

Definition 4 (order $\leq^c$) *Let (O_1,f_1) and (O_2,f_2) be in $\mathcal{C}(\mathcal{O},\mathcal{L},i)$,*
$(O_1,f_1)\leq^c(O_2,f_2)\Longleftrightarrow O_1\subseteq O_2$

This order is compatible with order on intents.

Proposition 1 $(O_1, f_1) \leq^c (O_2, f_2) \Longleftrightarrow (f_1 \dot{\models} f_2)$

As in FCA, Definitions 3 and 4 lead to the following *fundamental theorem.*

Theorem 1 *Let $(\mathcal{O}, \mathcal{L}, i)$ be a context, and let J be a set of indices. The ordered set $\langle \mathcal{C}(\mathcal{O}, \mathcal{L}, i); \leq^c \rangle$ is a finite lattice, whose supremum (least upper bound) and infimum (greatest lower bound) are as follows:*

$\bigvee^c_{j \in J}(O_j, f_j) =^c (\tau(\sigma(\bigcup_{j \in J} O_j)), \dot{\bigvee}_{j \in J} f_j)$

$\bigwedge^c_{j \in J}(O_j, f_j) =^c (\bigcap_{j \in J} O_j, \sigma(\tau(\dot{\bigwedge}_{j \in J} f_j)))$

Example 3. Figure 1.(a) represents the Hasse diagram of the concept lattice of context K_{ex} (introduced in Example 2). Concepts are represented by a number and a box containing their extent on the left, and their intent on the right. The higher concepts are placed in the diagram the greater they are for order $\leq^c$. It can be observed that the concept lattice is not isomorphic to the power-set lattice of objects $\langle 2^{\mathcal{O}}; \subseteq \rangle$. Indeed, set $\{x, y\}$ is not the extent of any concept, because $\tau(\sigma(\{x, y\})) = \tau(a \vee b) = \{x, y, z\}$.

2.4 Labelling of concept lattices

Similarly to standard CA, it is possible to label concept lattices with objects and formulas.

In the perspective of our application, the designer will choose a logic $\mathcal{L}$, an administrator will manage a formal context $(\mathcal{O}, \mathcal{L}, i)$, and a end-user will navigate, query, and consult/create/delete designated objects by using arbitrary formulas as labels. The end-user knows $\mathcal{L}$, but he does not necessarily know the formal context. For instance, he might be discovering it through navigation. In the following, we will ignore the possibility of labelling concepts with objects. It is fully described in [FR99].

Definition 5 *Let $(\mathcal{O}, \mathcal{L}, i)$ be a context. One defines a mapping μ labelling concepts with formulas:*

$\mu : \mathcal{L} \to \mathcal{C}(\mathcal{O}, \mathcal{L}, i) \qquad \mu(f) := (\tau(f), \sigma(\tau(f)))$.

Images of mapping μ are indeed concepts from Definition 3 of concepts, and from properties of applications σ and τ (Lemma 1). The next lemma gives interesting properties of these labellings of concept lattices.

Lemma 2 *Let $(\mathcal{O}, \mathcal{L}, i)$ be a context, and $o \in \mathcal{O}$, $f \in \mathcal{L}$, $c \in \mathcal{C}(\mathcal{O}, \mathcal{L}, i)$.*

(1) $c \leq^c \mu(f) \Longleftrightarrow int(c) \dot{\models} f$ *(2) μ is surjective*

(3) $\mu(int(c)) =^c c$ *(4)* $int(\mu(f)) \dot{\models} f$.

Lemma 2.(1) shows that $\mu(f)$ is the greatest concept whose intent logically entails f; and Lemma 2.(2) establishes that every concept is labelled at least by one formula. Regarding relation between concept intents and concept labels,

Lemma 2.(3) shows that every concept is labelled by its intent; and Lemma 2.(4) adds that every formula labelling a concept is logically entailed by the concept intent. This means that concepts can be characterized by several formulas, the most precise one being its intent. This idea is developed in Section 3 and useful for the application sketched in Section 5.

Example 4. Figure 1.(b) represents the same concept lattice as Figure 1.(a), but it does not associate the same information to concepts. The number of each concept is reused in its box so as to identify it; formulas of the form $\bigvee A$ where $A \subseteq \mathcal{A}$ ($\bigvee \emptyset \equiv 0$) are placed on the right of their labelled concept. For instance, concept 1 is labelled by formula a (i.e., $\mu(a) =^c 1$). In Figure 1.(b) we have restricted labels to be formulas of the form $\bigvee A$, but it is only to have a finite number of labels that are not all in the formal context; recall that every formula in $\mathcal{P}$ labels something. It is important for the applications we have in mind (i.e., querying and navigation) not to restrict labels to be (sub)formulas of the logical context. This last point implies that some concepts are necessarily labelled by several formulas, because there is a finite number of concepts (this is indeed observed with concept 6).

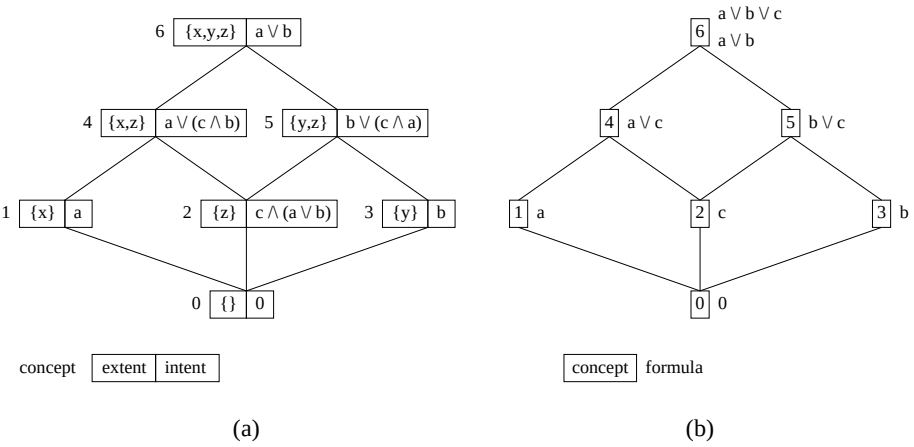

Fig. 1. The concept lattice of context K_{ex} (a) and its labelling (b).

3 Contextualized logic

In a particular context, it is possible to order some properties, although they are not comparable by $\models$. For instance, if in some context every bird flies, then we can say that property "bird" contextually entails the property "fly", although we have not necessarily $bird \models fly$ in $\mathcal{L}$. We introduce a *contextualized deduction*

relation as a generalization of implications between attributes, that can be used in standard CA for knowledge acquisition processes [GW99a,Sne98].

Definition 6 (contextualized deduction) *Let $K = (\mathcal{O}, \mathcal{L}, i)$ be a context, and $f, g \in \mathcal{L}$. One says that f* contextually entails *g in context K, which is noted $f \dot{\models}^K g$, if $\tau(f) \subseteq \tau(g)$, i.e., if every object that satisfies f also satisfies g.*

Relation $\dot{\models}^K$ is a preorder, and the associated equivalence relation is noted $\dot{\equiv}^K$.

Definition 7 (contextualized logic) *Let $K = (\mathcal{O}, \mathcal{L}, i)$ be a context. The term* contextualized logic *denotes the partially ordered set $\langle \mathcal{L}^K; \dot{\models}^K \rangle$ where*

$\mathcal{L}^K := \mathcal{L}/_{\dot{\equiv}^K}$

$\forall F, G \in \mathcal{L}^K : F \dot{\models}^K G \iff \exists f \in F, g \in G : f \dot{\models}^K g.$

Elements in $\mathcal{L}^K$ (equivalence classes of $\mathcal{L}$ modulo $\dot{\equiv}^K$) are called contextualized formulas*, and the class of a formula $f \in \mathcal{L}$ is denoted by $(f)^K$ (or more simply by f^K, if non-ambiguous).*

Lemma 3 *Let $K = (\mathcal{O}, \mathcal{L}, i)$ be a context, $f, g \in \mathcal{L}$, and $a, b \in \mathcal{C}(K)$.*

(1) $f \dot{\models} g \Longrightarrow f \dot{\models}^K g$ *(2)* $\forall o \in \mathcal{O} : i(o) \dot{\models} f \iff i(o) \dot{\models}^K f$

(3) $f \dot{\models}^K g \iff \mu(f) \leq^c \mu(g)$ *(4)* $a \leq^c b \iff int(a) \dot{\models}^K int(b)$

(5) $\sigma(\tau(f)) \dot{\equiv}^K f$

Lemma 3.(1) shows that contextualized deduction $\dot{\models}^K$ is only an enlargement of initial deduction $\dot{\models}$. Lemma 3.(2) shows that object properties are not altered by context: it does not add nor delete any property to objects. Lemmas 3.(3-4) show that μ and *int* are order-embeddings between $\langle \mathcal{L}; \dot{\models}^K \rangle$ and $\langle \mathcal{C}(K); \leq^c \rangle$. Lemma 3.(5) spots $\sigma(\tau(f))$ as the intent contextually equivalent to f.

A context plays the role of a theory extending the deduction relation and enabling new entailments. Contextualized logic can also be seen as a means for extracting knowledge from contexts. Two kinds of knowledge can thus be extracted: knowledge about context by deduction, and knowledge on the domain (from which context is extracted) by induction (e.g., generalizing from $bird \dot{\models}^K fly$).

Example 5. With morphism μ between contextualized deduction and order on concepts (Lemma 3.(3)), it is possible to use the labelled concept lattice (see Figure 1.(b)) to study contextualized logic in context K_{ex}. For instance, as concept 2 is smaller than concept 5 relation $c \models^{K_{ex}} b \vee c$ stands, which is already true in $\mathcal{P}$. More generally, it can be seen that all valid deductions in $\mathcal{P}$ are retrieved in contextualized deductions. Examination of the labelled concept lattice shows that the context adds new valid deductions between formulas: e.g., $c \models^{K_{ex}} a \vee b$, $a \vee b \vee c \equiv^{K_{ex}} a \vee b$.

We now consider connections between contextualized logic and concept lattice. The aim is to show that the concept lattice forms a new logic, derived from $\mathcal{L}$, and adapted to the context: the contextualized logic. For this, we use labelling mappings μ and int to establish a connection between formulas and concepts.

Theorem 2 *$\langle \mathcal{L}^K; \dot{\models}^K \rangle$ and $\langle \mathcal{C}(K); \leq^c \rangle$ are isomorphic, with μ^K as an isomorphism from contextualized formulas to concepts, and int^K as an isomorphism from concepts to contextualized formulas, defined by*

$\mu^K : \mathcal{L}^K \to \mathcal{C}(K) \qquad \mu^K(F) := \mu(f)$ *where* $f \in F$

$int^K : \mathcal{C}(K) \to \mathcal{L}^K \qquad int^K(c) := int(c)^K$

Algebraic structures $\langle \mathcal{C}(K); \vee^c, \wedge^c \rangle$ and $\langle \mathcal{L}^K; \dot{\vee}^K, \dot{\wedge}^K \rangle$ (where $\dot{\vee}^K, \dot{\wedge}^K$ denote supremum and infimum of $\langle \mathcal{L}^K; \dot{\models}^K \rangle$) are then isomorphic.

To summarize, there are three ways of considering a concept lattice, which is a sign of flexibility: (1) extents ($\tau(\sigma(2^{\mathcal{O}}))$) ordered by set inclusion ($\subseteq$), (2) intents ($\sigma(\tau(\mathcal{L}))$) ordered by logical deduction ($\dot{\models}$), (3) contextualized formulas ($\mathcal{L}^K$) ordered by contextualized deduction ($\dot{\models}^K$).

Finally, we consider connections between contextualized logic $\mathcal{L}^K$ and initial logic $\mathcal{L}$. The following relations stand between operations of both logic.

Theorem 3 *Let $f, g \in \mathcal{L}$,*

$f^K \dot{\wedge}^K g^K \dot{=}^K (f \wedge g)^K$ *and* $f^K \dot{\vee}^K g^K \dot{=}^K \sigma(\tau(f) \cup \tau(g))^K$

It should be noticed that the mapping $(.)^K$ is a morphism from formulas to contextualized formulas for the conjunctive operation, but not for the disjunctive one. From this it follows that the concept lattice is not isomorphic to the initial logic $\mathcal{L}$. Therefore, some properties of this logic can be lost in concept lattice. For instance, this is the case of the distributive property in context K_{ex} (see Example 2). Indeed, propositional logic $\mathcal{P}$ is distributive, while the following counter-example can be found in the concept lattice of context K_{ex} (see Figure 1): $2 \wedge^c (1 \vee^c 3) =^c 2$, and $(2 \wedge^c 1) \vee^c (2 \wedge^c 3) =^c 0$.

A concept lattice inherits of all properties of $\mathcal{L}$ (modulo $=^c$) only if relation $f^K \dot{\vee}^K g^K \dot{=}^K (f \dot{\vee} g)^K$ holds. For this, it is sufficient that $\tau(f \dot{\vee} g) = \tau(f) \cup \tau(g)$ (because this implies $\sigma(\tau(f) \cup \tau(g)) \dot{\equiv}^K \sigma(\tau(f \dot{\vee} g)) \dot{\equiv}^K f \dot{\vee} g$). In more concrete terms, that amounts to laying down the following condition on formulas describing objects:

$$(3) \quad \forall o \in \mathcal{O} : i(o) \dot{\models} f \dot{\vee} g \iff i(o) \dot{\models} f \vee i(o) \dot{\models} g$$

This condition amounts to considering there is no form of disjunction in object descriptions. Back to context K_{ex}, it is observed that the description of object z satisfies proposition $a \vee b$ but satisfies neither a nor b, which falsifies condition (3). This explains why the distributive property is lost in the concept lattice.

4 Related works

The first related work to consider is FCA itself since any given finite lattice is isomorphic to a concept lattice (see p. 27 in [GW99a]), and a formal context can be reconstructed from any concept lattice. This amounts to compiling $\mathcal{L}$ into a FCA formal context. But this compilation is not feasible because $\mathcal{L}$ is generally an infinite and incomplete lattice. However, $\mathcal{O}$ is finite in all applications we have in mind (managing concrete data collections). So, a finite number of $\mathcal{L}$ formulas is actually used in any logical context. These formulas ordered by $\dot{\models}$ form a diagram which could be completed in a finite lattice, that could be compiled into a FCA formal context. This works for defining formal concepts, but it does not work for labelling since any formula of $\mathcal{L}$ can be a label. It does not work either for $\mathcal{L}^K$. It is also not a good idea to compile the finite number of actually used formulas because the context is bound to change as often as, say, the state of a file system changes. So, we believe that LCA can be considered as an extension to FCA because it permits to handle arbitrary descriptions when we need them.

The need for using as a formal context a more refined structure than the object-attribute relation has already been remarked by previous authors. Two directions have been followed for extending FCA. The first one is to replace attributes with more complex data. For instance, Chaudron and Maille replace attributes by first-order terms with free variables [CM98]. The underlying logic is that of unification and anti-unification [Plo70]. Kuznetsov replaces attributes by graphs that fit well his application in automated learning, and he mentions that such a construction can be done for all sorts of formal contexts [Kuz99]. In fact LCA gives a framework for this direction, which comprises labelling and contextualized logic. In his thesis [Mai99], Maille defines independently an extension of FCA that is similar to LCA. His Section 6.3 is parallel to our Sections 2.2 and 2.3. In fact, both works are inspired by Davey and Priestley's book [DP90a]. Maille does not describe labelling and contextualized logic, but he describes the concrete handling of a logic with constraints.

The second direction is to transform a more refined context into a classical formal context. The first paragraph of this section shows an example of this direction. This idea has been applied to an object-attribute-value relation [GW99a,Pre97]. The motivation is that multi-valued-contexts (in fact object-attribute-value relations) are widely used in the real world (e.g., in databases). There exists two methods for extracting a classical formal context from a multi-valued formal context.

One method uses *conceptual scales*. They are formal contexts whose objects are values and where attributes express some properties about values (e.g., 2 has the property ≤ 3). The concepts that follow from such formal contexts define a hierarchy among scale attributes. Then a mono-valued context can be derived by replacing values by scale attributes according to this hierarchy. Our generalized context $(\mathcal{O}, \mathcal{L}, i)$ can be seen as a multi-valued context where i is the only attribute and $\mathcal{L}$ is its domain of values. Then, one has to find a scale for retrieving the ordering $\dot{\models}$ on $\mathcal{L}$. But this is not feasible in general, as already explained in the first part of this section.

The other method is called *logical scaling*. It amounts to expressing attribute-value relations in a formalized language (e.g., SQL) using a finite (preferably small) collection of unary predicates. Names are given to all the predicates. Then a classical formal context can be constituted by replacing values associated to every object by the names of all the predicates that are satisfied by the values. As in our framework, logical scaling can be applied to an arbitrary logic. The big difference is that we manage all formulas of the logic, which is in general infinite. It is of particular importance for applications we have in mind (see Section 5).

Some authors have also proposed to derive a logic from a standard formal context. E.g., Wille and Ganter propose to consider the formal context as defining prefered interpretations for the attributes considered as atomic propositions [GW99b]. A propositional formula is true (*a compound attribute is all-extensional* in Wille and Ganter's work) in the derived logic if it is true in all the prefered interpretations. This derived logic is in fact the contextualized logic of LCA when $\mathcal{L}$ is the logic of propositions $\mathcal{P}$, and every object description is a *complete conjunctive clause*.

5 Application to logical information systems

We plan to use LCA in the design of flexible information systems. In these systems, LCA is not used only for structuring *a posteriori* a formal context, but rather for structuring *a priori* incoming data. These systems try to merge a querying facility with a navigation facility.

5.1 Querying and navigating

Navigating usually means following links from places to places for reaching an objective. Links can be directories, URLs, file system links, etc. A fundamental concept in navigation is the *path*. A path is an ordered list of links that must be followed to find a place or an object. If the information system has a tree structure every object is accessible through only one path, and the game of navigating is to find this path. If the structure is a graph there may be several paths, and the game is to find one. Navigation requires erudition to know useful paths, judgement for recognizing that a path may lead to what is looked for, and some luck. Maintaining the information system structure, and keeping it navigable (i.e., ensuring objects are in the proper place), is difficult. However, it seems like the natural thing to do in information systems who have a structure *a priori* (e.g., given by a classification).

Querying usually means to elaborate a query that selects sufficiently few objects so that the relevant objects can be recognized among the answers. It supposes that every object have been indexed in some way. Indexes (and accordingly queries) can be sets of key-words, full-text words, etc. Just like navigation, querying requires erudition, judgement, and luck. Maintaining the information system structure is easier since the structure is essentially flat. Conversely, non-flat structures are not well-represented.

A concept lattice offers both navigation and querying but maintains the coherence automatically. Navigation amounts to following down-links in the concept lattice; concepts are considered as places, and objects that label a concept are considered to be *there* [GMA93,Lin95,VW95]. Querying amounts to conjunct the intent of the current path with some query, and to find which concept this conjunction labels. So doing, querying and navigation can be mixed in any order, and the coherence between both is maintained by the coherence of intents and extents. We have applied this idea to the design of a conceptual file system/shell.

Compared with previous works, the main originalities of our proposal is to adopt the familiar interface of many Unix shells (i.e., commands *ls*, *cd*, *rm*, *mv* are transposed from a hierarchical setting to a conceptual setting), and to handle updates (e.g., via commands *rm* and *mv*). Typical applications are the management of personal directories, folders, or software development environments. We also envisage layman applications like catalogs or cookbooks (see [FR99] for an experiment with a Vietnamese cookbook). An important feature is also that concept lattices are used to give the semantics of the commands, but not in the implementation. Instead, it is the contextualized logic of Section 3 that is used.

A short presentation of the conceptual shell is given in the next section, but more details can be found in [FR00].

5.2 A conceptual shell

A conceptual shell can be described informally by comparison with a classical shell as follows: *files* become *objects*, *paths* become *logical formulas*, *directories* become *concepts* or *contextualized formulas*, the *root* becomes the concept $\top^c$ or formula $\dot{\top}$, and the *working directory* becomes a *working concept*.

The shell commands *cd* and *ls* are transposed in the conceptual shell for querying and navigating. Command *cd* merely maintains a *working query*, *wq*, labelling a *working concept* similar to a *working directory*, and command *ls* actually does the querying/navigation as follows. A command *ls q* returns a list of objects whose description is contextually equivalent to $q\dot{\wedge}wq$ (i.e., the objects labelling the concept $\mu(q\dot{\wedge}wq)$), plus a list of derived queries that possibly characterize strictly smaller but non empty concepts. The principle is that if a derived query q' is returned, the following holds:

$$(4) \quad \bot^c <^c \mu(q'\dot{\wedge}q\dot{\wedge}wq) <^c \mu(q\dot{\wedge}wq)$$

The fact that answers to queries may contain other queries that are relevant (i.e., satisfy the above inequation) is interesting in itself, and could be used in a natural language man-machine-interface: e.g., in a bookshop, *"Q: Do you have the Jungle Book? A: Yes we have, and also an illustrated version in the children department"*. Here, *Jungle Book* is the query, *Yes we have* means that there is an object with a contextually equivalent description (e.g., *Rudyard Kipling's best-known book*), and *illustrated version in the children department* means that there is also an object with a contextually strictly stronger description. According to the context, both *illustrated* and *children* can be derived queries. If the bookshop

had only the illustrated version, the answer could have been *"Yes, in the children department"*.

In fact, derived queries act as sub-directories of the working directory. Just like the standard *ls* command, the conceptual logic *ls* command has an option *-r* which tells it to search "recursively" a directory and its sub-directories. However, in its details the conceptual query *ls -r q* is not recursive at all since it simply returns the list of objects whose description satisfies $q\dot{\wedge}wq$ (i.e., the extent of the concept $\mu(q\dot{\wedge}wq)$).

Implementing a logical information system such as described above seems to require building the concept lattice, and being able to compute $<^c$ and μ. In fact, since the contextualized logic is isomorphic to the concept lattice by μ, operation $<^c$ on concepts can be replaced by operation $\dot{\models}^K$ on formulas (μ becomes unnecessary), whose computation can be reduced to the computation of τ (cf. Definition 6). Because of Definition 2, it is therefore sufficient to memorize a Hasse subdiagram of the ordering of formulas ($\dot{\models}$), containing at least all the descriptions of objects ($i(\mathcal{O})$) to place them in the diagram, and possibly others like past queries.

This gives an approximation of the concept lattice whose size is in the order of the number of objects instead of an exponential of this number. The concept lattice can be seen as representing answers to all possible queries, while the Hasse diagram is a cache of answers to past queries. Our working hypothesis is that the end-user will navigate progressively, and will repetitively use the same idioms. One advantage of this structure is that it is not sensitive to the actual contents of the formal context; in particular it is not sensitive to context changes (i.e., *rm* and *mv*), but only to navigation steps.

All operations of the conceptual shell can be implemented using this data structure and the following primitive functions.

Extent of a query The extent of a query q is noted $\tau(q)$ and returns all objects whose description satisfies q. Operation τ is the same as in the Galois connection of LCA (cf. Definition 2). It is computed as the union of all objects accessible in the Hasse diagram starting from the node labelled q. It is similar to *ls -r q* in a hierarchy.

Objects of a query The objects of a query q, noted $t(q)$, are objects whose description is contextually equivalent to q (i.e., has the same extent): $t(q) = \{o \in \mathcal{O} \mid \tau(i(o)) = \tau(q)\}$. It is computed as the union of first objects accessible in the Hasse diagram starting from the node labelled q. It is similar to *ls q* in a hierarchy.

Derived queries $Inc(q) := \lceil\{x \in X \mid \emptyset \subsetneq \tau(x\dot{\wedge}q) \subsetneq \tau(q)\}\rceil$, where $\lceil E\rceil$ denotes the set of greatest elements of E according to the order $\dot{\models}$. Set X is chosen freely, but one shows that if every description of objects in $\tau(q)$ can be expressed as a conjunction of formulas in X, then for every object in $\tau(q)$ there is an x in $Inc(q)$ such that the object is in $\tau(x\dot{\wedge}q)$. This property ensures that every object is accessible through navigation. At any given moment in a navigation, the set of formulas of the diagram has this property.

None of these operations actually computes $\dot{\models}$ because it is entirely cached in the Hasse diagram. It is in command *cd q* that the position of $q \dot{\wedge} wq$ is searched in the diagram, or inserted if necessary. It is only there that $\dot{\models}$ is used.

6 Conclusion

We have shown how FCA can be reconstructed when the formal context is not restricted to be an object-attribute relation. We show how to use an almost arbitrary logic instead. A by-product of this reconstruction is the derivation of a contextualized logic that adds to the logic of the formal context deductions that are only valid in the formal context. The contextualized logic plays the same role as the concept lattice, and corresponds to attribute implication [GW99a].

We propose to exploit contextualized logic for navigating in a conceptual shell. The main interest of concept analysis in this application is in tightly combining querying and navigation: contextualized formulas are at the same time queries and places where it is possible to read and write. Note that in this application, it is important that no information is lost in converting an extended formal context into a standard one. On one hand, the operations of the conceptual shell make an intensive use of labelling functions and of the contextualized logic. So, it is important that concept analysis handles extended contexts directly. On the other hand, these operations never use the concept lattice as such. Its role is played by the contextualized logic. This makes it easy to update a formal context.

The logic used in the extended formal context is almost arbitrary, but new constraints apply on it when placed in the perspective of implementing a logical information system. Indeed, the primitive operations (e.g., deduction, conjunction, computation of the extent of a formula) must be tractable. For instance, we plan to apply LCA to the domain of software engineering. In this case, description logics (DL [DLNS96,DLNN97]), which are expressive though tractable logics can be used for version and configuration management [Zel98]; and the logic of type isomorphisms [Di 95] can be used for navigating in software component libraries.

At a more theoretical level, two directions for future works are to study the possibility of having relations in the extended formal context, and to study the possibility given by *views* (e.g., like in data-bases) to hide details. This would be extremely useful with extended formal contexts that could be overloaded by details that do not concern every user.

References

[CM98] L. Chaudron and N. Maille. 1st order logic formal concept analysis: from logic programming to theory. *Computer and Information Science*, 13(3), 1998.

[Di 95] R. Di Cosmo. *Isomorphisms of Types: from λ-calculus to information retrieval and language design.* Progress in theoretical computer science. Birkhäuser, 1995.

[DLNN97] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1–58, 10 April 1997.

[DLNS96] F.-M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 191–236. CSLI Publications, Stanford (CA), USA, 1996.

[DP90a] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[DP90b] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*, chapter 11 – Formal Concept Analysis, pages 221–236. Cambridge University Press, 1990.

[FR99] S. Ferré and O. Ridoux. Une généralisation logique de l'analyse de concepts logique. Technical Report RR-3820, Inria, Institut National de Recherche en Informatique et en Automatique, December 1999. An english version is available at http://www.irisa.fr/lande/ferre.

[FR00] Sébastien Ferré and Olivier Ridoux. A file system based on concept analysis. In Yehoshua Sagiv, editor, *International Conference on Rules and Objects in Databases*, 2000. To appear.

[GMA93] R. Godin, R. Missaoui, and A. April. Experimental comparison of navigation in a Galois lattice with conventional information retrieval methods. *International Journal of Man-Machine Studies*, 38(5):747–767, 1993.

[GW99a] B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer, 1999.

[GW99b] Bernhard Ganter and Rudolf Wille. Contextual attribute logic. *Lecture Notes in Computer Science*, 1640:377–388, July 1999.

[KS94] M. Krone and G. Snelting. On the inference of configuration structures from source code. In *Proceedings of the 16th International Conference on Software Engineering*, pages 49–58. IEEE Computer Society Press, May 1994.

[Kuz99] S. O. Kuznetsov. Learning of simple conceptual graphs from positive and negative examples. In Jan M. Żytkow and Jan Rauch, editors, *Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-99)*, volume 1704 of *LNAI*, pages 384–391, Berlin, September 15–18 1999. Springer.

[Lin95] C. Lindig. Concept-based component retrieval. In *IJCAI95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs*, 1995.

[Mai99] Nicolas Maille. *Modèle logico-algébrique pour la fusion symbolique et l'analyse formelle*. PhD thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, November 1999.

[Plo70] G. D. Plotkin. A note on inductive generalization. *Machine Intelligence, Edinburgh Univ. Press*, 5:153–163, 1970. Edinburgh Univ. Press, Edinburgh.

[Pre97] Susanne Prediger. Logical scaling in formal concept analysis. *Lecture Notes in Computer Science*, 1257:332–341, August 1997.

[Sne98] G. Snelting. Concept analysis — A new framework for program understanding. *ACM SIGPLAN Notices*, 33(7):1–10, July 1998.

[VW95] F. Vogt and R. Wille. TOSCANA — a graphical tool for analyzing and exploring data. *Lecture Notes in Computer Science*, 894:226–??, 1995.

[Wil82] Rudolf Wille. *Ordered Sets*, chapter Restructuring lattice theory: an approach based on hierarchies of concepts, pages 445–470. Reidel, Dordrecht Boston, 1982.

[Zel98] A. Zeller. Versioning system models through description logic. *Lecture Notes in Computer Science*, 1439:127–132, 1998.