Searching for Objects and Properties with Logical Concept Analysis

Sébastien Ferré** and Olivier Ridoux

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

Abstract Logical Concept Analysis is Formal Concept Analysis where logical formulas replace sets of attributes. We define a Logical Information System that combines navigation and querying for searching for objects. Places and queries are unified as formal concepts represented by logical formulas. Answers can be both extensional (objects belonging to a concept) and intensional (formulas refining a concept). Thus, all facets of navigation are formalized in terms of Logical Concept Analysis. We show that the definition of being a refinement of some concept is a specific case of Knowledge Discovery in a formal context. It can be generalized to recover more classical KD operations like machine-learning through the computation of necessary or sufficient properties (modulo some confidence), or data-mining through association rules.

1 Introduction

Information systems offer means for organizing data, and for navigating and querying. Though navigation and querying are not always distinguished because both involve queries and answers, we believe they correspond to very different paradigms of human-machine communication. In fact, the difference can be clarified using the intension/extension duality.

Navigation implies a notion of place, and of a relation between places (e.g., file system directories, and links or subdirectory relations). Through navigation, a user may ask for the contents of a place, or ask for related places. The ability to ask for related places implies that answers in the navigation-based paradigm belong to the same language as queries. In terms of the intension/extension duality, a query is an intension, and answers are extensions for the contents part, and intensions for the related places.

In very casual terms. we consider navigation with possibly no map, i.e., no *a priori* overview of the country. Related places form simply the landscape from a given place as shown by a "viewpoint indicator". However, our proposal is compatible with any kind of *a priori* knowledge from the user.

With querying, answers are extensions only. A simulation of navigation is still possible, but forces the user to infer what could be a better query from the unsatisfactory answer to a previous query; i.e., infer an intension from an

** This author is supported by a scholarship from CNRS and Région Bretagne extension. This is difficult because there is no simple relation between a variation in the query, and the corresponding variation in the answer. The experience shows that facing a query whose extension is too vast, a user may try to refine it, but the resulting extension will often be either almost as vast as the former or much too small. In the first case, the query lacks of precision (i.e., number of relevant items in the answer divided by total number of items in the answer), whereas in the second case, the query recall (i.e., number of relevant items in the answer divided by number of relevant items in the system) is too low.

Godin *et al.* [GMA93] have shown that Formal Concept Analysis is a good candidate for reconciliating navigation and querying. We follow this opinion, but we believe that care must be taken to make formal contexts as close to the description languages of the end-users, and we have proposed Logical Concept Analysis (LCA) where formal descriptions are logical formulas instead of being sets of attributes [FR00b]. So doing, one may consider that the contents of an information system is a formal context in which items are associated to formulas that describe them in a user-oriented way. We call this a Logical Information System (LIS). Then, Concept Analysis automatically organizes the contents of the information system as a lattice of concepts.

Our goal in this article is to show how a form of navigation and querying can be defined, so that a user who knows neither the contents of a Logical Information System, nor the logic of its descriptions, can navigate in it and discover the parts of the contents and the parts of the logic that are relevant to his quest. Note that a more expert user may know better and may navigate more directly to his goal, but since almost everybody has his *Terra Incognita*, the no-knowledge assumption is the safest one to do.

We will present in the sequel formal means for navigating in a Logical Information System in order to find relevant objects (answers in the extensional language), and relevant properties of the formal context (answers in the intensional language). We will show how this latter point is related to data-mining, knowledge-engineering, and machine-learning. The core of the formal means used in this work is (Logical) Concept Analysis.

The article is organized as follows. Section 2 presents a guided tour of Logical Concept Analysis. Section 3 presents the notion of Logical Information System. Sections 4 and 5 present the details of the navigation in a LIS, and of the extraction of properties. Sections 6 and 7 present conclusions and perspectives.

2 Logical Concept Analysis

We recall the main definitions and results about Logical Concept Analysis (LCA). More explanations and results can be found in [FR00b].

Definition 1 (context) *A* logical 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 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 it.*

Given a formal context K, one can form a Galois connection between sets of objects (extents) and formulas (intents) with two applications σ and τ.

Definition 2 *Let $K = (\mathcal{O}, \mathcal{L}, i)$ be a logical context,*
$\sigma^K : \mathcal{P}(\mathcal{O}) \to \mathcal{L},\ \sigma^K(O) := \dot{\bigvee}_{o \in O} i(o)$
$\tau^K : \mathcal{L} \to \mathcal{P}(\mathcal{O}),\ \tau^K(f) := \{o \in \mathcal{O} | i(o) \dot{\models} f\}$

Formal concepts can be derived from logical contexts.

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

The set of all concepts that can be built in a context K is denoted by $\mathcal{C}(K)$, and is partially ordered by $\leq^c$ defined as follows.

Definition 4 $(O_1, f_1) \leq^c (O_2, f_2) :\Longleftrightarrow O_1 \subseteq O_2 \quad (\Longleftrightarrow f_1 \dot{\models} f_2)$

Definitions 3 and 4 lead to the following *fundamental theorem.*

Theorem 1 *Let $K = (\mathcal{O}, \mathcal{L}, i)$ be a context. The ordered set $\langle \mathcal{C}(K); \leq^c \rangle$ is a finite lattice with supremum $\vee^c$ and infimum $\wedge^c$.*

It is possible to label concept lattices with formulas (resp. objects) through a labelling map μ (resp. γ).

Definition 5 *Let K be a logical context,*
$\mu^K : \mathcal{L} \to \mathcal{C}(K),\quad \mu^K(f) := (\tau^K(f), \sigma^K(\tau^K(f)))$
$\gamma^K : \mathcal{O} \to \mathcal{C}(K),\quad \gamma^K(o) := (\tau^K(\sigma^K(\{o\})), \sigma^K(\{o\})) = (\tau^K(i(o)), i(o))$

We introduce a *contextualized deduction relation* as a generalization of the implications between attributes that are used in FCA for knowledge acquisition processes [GW99,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 $f \dot{\models}^K g$, iff $\tau^K(f) \subseteq \tau^K(g)$, i.e., iff every object that satisfies f also satisfies g.*

Relation $\dot{\models}^K$ is a preorder, whose associated equivalence relation is noted $\dot{\equiv}^K$. Formulas ordered by this deduction relation form a new logic adapted to the context: the *contextualized logic.* It is connected to the concept lattice by the following theorem.

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

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

3 A Logical Information System

A Logical Information System (LIS) is essentially a logical formal context (see Definition 1) equipped with navigation and management tools. In this article, we are mostly interested in the end-user perspective. So, we will insist on navigation tools, and will only briefly allude to the management of a logical formal context.

For illustration purpose, we will present a bibliographical reference system whose principle is the following. Objects are bibliographical references, whose contents are BibTEX entries, and descriptions are composed of titles, lists of authors, etc, extracted from the contents, and appreciations (e.g., "theoretical" or "practical") and status (e.g., "read") given by a user. Logical formulas used for descriptions and queries are sets of valued attributes. For instance, numerical fields can be described in an interval logic, and string fields can be described in a boolean logic based on the absence/presence of substrings in a string: e.g., `title: Logic & -(Concept|Context)/year: 1990..1995`. In an advanced version of this system, we also introduced regular expressions and modalities for representing incomplete knowledge.

Our LIS needs an end-user interface. We will use a shell-based interface, though this is not the most modern thing to do. This is because we believe that shell interfaces (like in UNIX or MS-DOS) are familiar to many of us, and because this abstraction level exposes properly the dialogue of queries and answers. A higher-level interface like a graphical one would hide it, whereas lower-level interfaces, like a file system, would expose irrelevant details.

A prototype of the bibliographical system has been built for experimentation purpose. It has been implemented in Prolog as a generic system in which a theorem-prover and a syntax analyzer can be plugged-in for every logic used in descriptions. It is not meant to be efficient, though it can handle several hundred entries. Contrary to other tools based on concept analysis, it does not create the concept lattice. It only manages a Hasse diagram of the formulas used so far [FR00a]. In our experiments with the logic presented above, this diagram has an average of 5 nodes per object, 3 arcs per node, and a height of about 5.

3.1 Building a Logical Context

The first task one needs to do in a LIS is to build a Logical Context $K = (\mathcal{O}, \mathcal{L}, i)$; this amounts to create and logically describe objects. To each object o is associated a *content* $c(o)$ (here, the BibTEX reference) and a logical *description* $i(o)$. We distinguish content and description in order to hide some BibTEX fields (e.g., `publisher`) and add some non-BibTEX fields (e.g., `status`). We give an example of an object by displaying its content and its description in the following table. A "minus" sign before string formulas represents negation, and the "less" and "greater" signs around strings denote an "all I know" modality that means that the strings are closed (they do not contain anything else): e.g., `title: <"logic">` $\dot{\models}$ `title: - "context"` whereas `title: "logic"` $\not\dot{\models}$ `title: - "context"`.

```
c(o) = @InProceedings{FerRid2000b,
  author = {Sébastien Ferré and Olivier Ridoux},
  title = {A Logical Generalization of Formal Concept Analysis},
  booktitle = {International Conference on Conceptual Structures},
  editor = Guy Mineau and Bernhard Ganter,
  series = LNCS 1867,
  publisher = Springer,
  year = {2000},
  keywords = {concept analysis, logic, context, information system} }
i(o) =
  type: <"InProceedings">/
  author: <"Sébastien Ferré and Olivier Ridoux">/
  title: <"A Logical Generalization of Formal Concept Analysis">/
  year: 2000/
  keywords: <"concept analysis, logic, context, information system">/
  status: "read"
```

For all examples given in the following sections, we consider as context K all ICCS publications until the year 1999, which consists in 209 objects. For this context, the Hasse diagram has 954 nodes and 2150 arcs. In the following experiments of this paper, all response times are shorter than 5 seconds.

3.2 Navigating in a Logical Context

Once objects have been logically described and recorded in a logical context K, one wants to retrieve them. One way to do this is *navigating* in the context. As already said, this way of searching is particularly useful in a context where the logic or the contents are unknown. The aim of navigation is thus to guide the user from a *root place* to a *target place*, which contains the object(s) of interest. For this, a LIS offers to the user 3 basic operations (the corresponding UNIX-like command names are placed between parenthesis): (1) to ask to LIS what is the current place (command `pwd`), (2) to go in a certain "place" (command `cd`), (3) to ask to LIS ways towards other "places" (command `ls`).

In a hierarchical file system, a "place" is a directory. But in our case, a "place" is a concept, which can be seen as a coherent set of objects (extent) and properties (intent) (cf. Definition 3). In large contexts, concepts cannot be referred to by enunciating either their extent or their intent, because both are generally too large. Formulas of the logic $\mathcal{L}$ can play this role because every formula refers to a concept through the labelling map μ (cf. Definition 5), and every concept is referred to by one or several formulas, which are often simpler than the intent.

We now describe the 3 navigation operations listed above. First of all, going from place to place implies to remember the current place, which corresponds to the working directory. In our LIS, we introduce the *working query*, wq, and the *working concept*, $wc := \mu^K(wq)$; we say that wq refers to wc. This working query is taken into account in the interpretation of most of LIS commands, and is initialized to the formula $\dot{\top}$, which refers to the concept whose extent is the set of all objects. Command `pwd` displays the working query to the user.

Command `cd` takes as argument a query formula q saying in which place to go, and it changes the working query accordingly. We call l_{wq} (elaboration of wq) the mapping that associates to the query q a new working query according to the current working query wq. The query q can be seen as a *link* between the current and the new working query. Usually, `cd` is used to refine the working concept, i.e., to select a subset of its extent. In this case, the mapping l_{wq} is defined by $l_{wq}(q) := wq \dot{\wedge} q$, which is equivalently characterized by

$$\mu^K(l_{wq}(q)) =^c wc \wedge^c \mu^K(q) \text{ and } \tau^K(l_{wq}(q)) = \tau^K(wq) \cap \tau^K(q).$$

However, it is useful to allow other interpretations of the query argument. For instance, we can allow the distinction between *relative* and *absolute* query, similarly to relative and absolute paths in file systems. The previous definition of the mapping l_{wq} concerns relative queries, but can be extended to handle absolute queries by $l_{wq}(/q) := q$, where '/' denotes the absolute interpretation of queries. This allows to forget the working query. We can also imagine less usual interpretations of queries like $l_{wq}(|\ q) := wq \dot{\vee} q$. Finally, the special argument `..` for the command `cd` enables to go back in the history of visited queries/concepts. This works much like the "Back" button in *Web* browsers.

Command `ls` is intended to guide the user towards his goal. More precisely, it must suggest some relevant links that could act as queries for the command `cd` to refine the working query. These links are formulas of $\mathcal{L}$. A set of links given by `ls` should be finite, of course (whereas $\mathcal{L}$ is usually infinite), even small if possible, and complete for navigation (i.e., each object of the context must be accessible by navigating). We postpone the development of this issue to Section 4.

As navigation aims at finding objects, command `ls` must not only suggest some links to other places, but also present the object belonging to the current place, called the *object of wq* or the *local object*. It is defined as the object labelling the working concept through the labelling map γ (cf. Definition 5). Formally, the object of a query q is defined by

$$t^K(q) := o \in \mathcal{O} \text{ such that } \gamma^K(o) =^c \mu^K(q).$$

There can be no local object, and there cannot be several ones because this would be equivalent to have two different objects described by exactly the same formula, which would make them undistinguishable. Another interesting thing to notice is that the working query can be, and is often, much shorter than the whole description of the local object (which is also the intent of the working concept), as in the following example where the formula on the first line is contextually equivalent to the description on the four other lines for accessing the object.

```
i(t(author: Mineau & Missaoui)) =
  type: <InProceedings>/
  title: <"The Representation of Semantic Constraints in Conceptual Graph Systems">/
  author: <"Guy W. Mineau and Rokia Missaoui">/
  year: 1997
```

3.3 Updating and Querying a Logical Context

Updating is done *via* shell commands like `mv` or `cp`. With option `-r`, every object of the working concept is concerned, while in the opposite case, only the local object is (considering it exists, cf. Section 3.2).

```
(1) cd /author: Mineau & Missaoui
(2) mv . status: "to be read"
(3) cd /keywords: FCA | GC
(4) mv -r status: "to be read"    status: "read"
```

Contents can also be changed with a LIS-command `chfile`. This changes indirectly descriptions, but the ensuing reorganization of the formal concept lattice is automatic and transparent. In fact, it costs not so much since the concept lattice is not actually represented.

Extensional queries can be submitted to a logical information system using the `-r` option with command `ls`. The answer to query `ls -r q` is simply $\tau^K(l_{wq}(q))$, i.e., the extent of the concept refered to by $l_{wq}(q)$ (cf. Section 3.2).

```
(1) ls -r /title: Logic & -(Concept | Context)/year: 1990..1995
3 B. R. Gaines. "Representation, discourse, logic and truth: situating knowledge technology".
INPROC, 1993.
2 J. F. Sowa. "Relating diagrams to logic". INPROC, 1993.
72 H. van den Berg. "Existential Graphs and Dynamic Predicate Logic". INPROC, 1995.
3 object(s)
```

4 Searching for Objects

We now define more precisely than in Section 3.2 the dialogue between a user and our LIS through commands `cd` and `ls`. Let us recall that `cd` enables the user to traverse a link from the working concept to another one, and that `ls` enables him to get from LIS a set of relevant links to refine the working concept.

Navigation Links. The following notion of refinement corresponds to the case where the elaboration mapping satisfies $l_{wq}(q) = wq \dot{\wedge} q$. To avoid to go in the concept $\perp^c$ whose extent is empty, we must impose the following condition on a link x: $\tau^K(wq \dot{\wedge} x) \neq \emptyset$. As $\mathcal{L}$ is a too wide search space, we consider a finite subset X of $\mathcal{L}$ in which links are selected. The content of X is not strictly determined but it should contain simple formulas, some frequently used formulas, and more generally, all formulas that users expect to see in `ls` answers. X can be finite because terminology and used formulas are (because the context is finite). Furthermore, we keep only greatest links (in the deduction order) as they correspond to smallest refinement steps. We can now define the set of links of a working query wq by $Link^K(wq) := \lceil \{x \in X | \tau^K(wq \dot{\wedge} x) \neq \emptyset\} \rceil$, where $\lceil E \rceil$ denotes the set of greatest elements of E according to the order $\dot{\models}$.

Increments vs. Views. We can distinguish two kinds of links: *increments* that strictly restrict the working concept (i.e., $\tau^K(wq \dot{\wedge} x) \neq \tau^K(wq)$), and *views* that are properties shared by all the objects of the working concept (i.e., $\tau^K(wq \dot{\wedge} x) = \tau^K(wq)$). Only increments are useful as arguments of `cd`, because the application of `cd` to a view would not change the working concept.

Why not use only increments if views are not useful for refinement? Because sets of increments appear to be too large and heterogeneous: in the BibTEX example, they mix author names, title words, years, etc. Sets of links are smaller because many increments are subsumed by views, and then are not returned as links. For instance, the view (`author: *`) is returned as a summary of a large set of author name increments. So, if views are not useful for selecting objects, they are useful for selecting increments. We introduce a working view wv, similar to the working query, under which links must be searched for. For instance, if the working view is (`author: *`), links will be author name increments. We modify the definition of *Link* to take this into account.

Definition 7 $Link^K(wq, wv) := \lceil\{x \in X | x \dot{\models} wv, wv \dot{\not\models} x, \tau^K(wq \dot{\wedge} x) \neq \emptyset\}\rceil$.

Definition 8 *The completeness of Link is formally expressed by*
$\forall wq \in \mathcal{L} : \forall o \in \tau^K(wq) : o \neq t^K(wq) \Rightarrow \exists x \in Link^K(wq, wv) : o \in \tau^K(wq \dot{\wedge} x)$.

In English, this means that for all working query wq and for all object o of its extent, if o is not yet the local object then it exists a link that enables to restrict the working extent while keeping o in it.

Theorem 3 *$Link^K(wq, wv)$ is complete for all wq and for all wv being a view for wq iff every object description belongs to X.*

In fact, X acts as the vocabulary that LIS uses in its answers. In our prototype, X is in a large part automatically generated from the context (i.e., from object descriptions) every time an object is created (command `mkfile`) or updated (commands `chfile` and `mv`); and from queries in order to incorporate the user vocabulary in the LIS one. This automatic generation guarantees the completeness of the navigation according to the above condition, while favouring small links such as author names, title words, keywords (in fact, whole object descriptions very rarely appear as links, cf. Table 1). Furthermore, the user can adjust the LIS vocabulary by adding and removing formulas in X with commands `mkdir` and `rmdir`.

To summarize, the view-based variant of command `ls` takes as argument a view v, sets the working view to $l_{wv}(v)$ (where l_{wv} works similarly to l_{wq}), shows the local object if it exists, displays each link x of the set $Link^K(wq, wv)$ along with the size of its selected extent $\tau^K(wq \dot{\wedge} x)$, and finally displays the size of the working extent $\tau^K(wq)$. Increments and views are distinguished according to their cardinality compared to the working size; views simply have the same size as the working concept, whereas increments have strictly smaller sizes.

User/LIS Dialogue. We now show how commands `cd` and `ls` compose a rather natural dialogue between the user and LIS. The user can refine the working concept with command `cd`, and asks for suggested links with the command `ls`. LIS displays to the user relevant increments for forthcoming `cd`'s, and relevant views for forthcoming `ls`'s. Commands `cd` (resp. increments) are *assertions* from the user (resp. from LIS): "I want this kind of object!" (resp. "I have this kind of object!"). Commands `ls` (resp. views) are *questions* from the user (resp. from LIS): "What kind of object do you have?" (resp. "What kind of object do you want?"). It should also be noticed that both the user and LIS can answer to questions both by assertions and by questions.

A complete example of a dialogue is given in Table 1. The left part of this table shows what is really displayed by our prototype, and the right part is an English translation of the dialogue. Notice that this translation is rather systematic and could thus be made automatic. `(n)` is the prompt for the n-th query from the user. On the 2nd query, the question of the user is so open, that LIS only answers by questions. On the 3rd query, the user replies to one of these questions (`title: *`) by an assertion; but on the 4th query, he sends back to LIS another of these questions (`author: *`) to get some relevant suggestions. On the 5th query, he just selects a suggested author, `"Wille"`, and then gets his co-authors on Concept Analysis with the 6th query. On the 7th query, he selects a co-author and finally finds an object at the 8th query.

`(1) pwd`	`(1) What is currently selected?`
`/`	`All objects.`
`(2) ls`	`(2) What do you have?`
`209 type: *`	`What kind of type do you want?`
`209 author: *`	`What kind of author do you want?`
`209 year: ..`	`What kind of year do you want?`
`209 title: *`	`What kind of title do you want?`
`209 object(s)`	`209 objects are currently selected.`
`(3) cd title: "Concept Analysis"`	`(3) I want objects whose title contains "Concept Analysis"!`
`(4) ls author: *`	`(4) What kind of author do you have (for this)?`
`1 author: "Mineau"`	`I have 1 object with author "Mineau"!`
`1 author: "Lehmann"`	`I have 1 object with author "Lehmann"!`
`1 author: "Stumme"`	`I have 1 object with author "Stumme"!`
`1 author: "Prediger"`	`I have 1 object with author "Prediger"!`
`3 author: "Wille"`	`I have 3 objects with author "Wille"!`
`4 object(s)`	`4 objects are currently selected.`
`(5) cd author: Wille`	`(5) I want objects with author "Wille"!`
`(6) ls`	`(6) What kind of author do you have (yet)?`
`1 author: "Mineau"`	`I have 1 object with author "Mineau"!`
`1 author: "Lehmann"`	`I have 1 object with author "Lehmann"!`
`1 author: "Stumme"`	`I have 1 object with author "Stumme"!`
`3 author: "Wille"`	`What kind of author "Wille" do you want?`
`3 object(s)`	`3 objects are currently selected.`
`(7) cd author: Mineau`	`(7) I want objects with author "Mineau"!`
`(8) ls`	`(8) What do you have?`
`200 Guy W. Mineau and Gerd Stumme and Rudolf Wille. "Conceptual Structures Represented by Conceptual Graphs and Formal Concept Analysis". INPROCEEDINGS, 1999.`	
`1 object(s)`	`1 object is currently selected.`
`(9) pwd`	`(9) What is currently selected?`
`author: "Wille" & "Mineau"/`	`Objects with authors "Wille" and "Mineau",`
`title: "Concept Analysis"`	`and whose title contains "Concept Analysis".`

Table1. Example of User/LIS Dialogue in the BibTEX context.

Related Work. We finish this section by comparing our LIS navigation with other kinds of navigation based on Concept Analysis. Lindig [Lin95] designed a concept-based component retrieval based on sets of *significant keywords* which are equivalent to our increments for the logic of attributes underlying FCA. Godin et al. [GMA93] propose a direct navigation in the lattice of concepts, which is in fact very similar to Lindig's approach except that only greatest significant keywords, according to the contextualized deduction on attributes, are displayed to the user. They have also notions common to our LIS such as working query, direct query specification, and history of selected queries. Cole and Stumme [CS00] developed a Conceptual Email Manager (CEM) where the navigation is based on Conceptual Scales [Pre97,PS99]. These scales are similar to our views in the sense that they select some attributes acting as increments and displayed, as for us, with the size of the concept they select. A difference with our LIS is that these increments are ordered according to concept lattices of scales, but it could also be done in LIS by a post-treatment on answers of command `ls` if we had a GUI. But the main difference with all of these approaches is that we use an (almost) arbitrary logic to express properties. This enables us to have automatic subsumption relations (e.g., (`author: Wille & -Mineau`) $\dot{\models}$ (`author: Wille`) $\dot{\models}$ (`author: *`)), and thus some implicit views (e.g., `author: *`, `year: ..`).

5 Searching for Properties

In previous sections, Concept Analysis (CA) is used to specify navigation and querying in a LIS. However, in the past CA has been often applied in domains such as *data-analysis*, *data mining*, and *learning*.

Data-analysis consists in structuring data in order to help their understanding. These data are often received as tables or relations and structured by partitions, hierarchies, or lattices. With CA, formal contexts (binary relations between objects and attributes) are structured in concept lattices [GW99]. This is applied for instance in software engineering for configuration analysis [KS94]. *Data-mining* is used to extract properties from large amount of data. These properties are association rules verified (exactly or approximately) by the data. This is analogous to implications between attributes in FCA (cf. p. 79 in [GW99]), and to contextualized logic in LCA [FR00b]. *Unsupervised learning* is similar to data-analysis in the sense that one tries to discover some properties, and to understand some data, whereas *supervised learning* is similar to data-mining as some rules are searched for between known properties and the property to be learned. For instance, Kuznetsov applied CA to the learning of a positive/negative property from positive and negative instances [Kuz99].

The issue of this section is to show whether these features of Knowledge Discovery (KD) can be incorporated in our LIS, and how. Our aim is not to fully implement them in the LIS itself, but to offer primitives that could be combined for building more sophisticated KD features. First, we show how each of the three above kinds of KD can be formally expressed with the only notion of contextualized logic (cf. Section 2).

KD through Contextualized Logic. A context K plays the role of a theory by extending the deduction relation and enabling new entailments (e.g., $bird \dot{\models}^K fly$ when every bird flies in the context). All these contextual entailments are gathered with logical entailments to form the contextualized logic, which is thus a means for extracting some knowledge from the context. Two kinds of knowledge can be extracted: knowledge about the context by deduction ("Every bird of this context *do* fly"), and knowledge about the domain (which the context belongs to) by induction ("Every bird of the domain *may* fly").

Concept lattices produced by data-analysis are isomorphic to contextualized logics (cf. Theorem 2). Associations rules produced by data-mining or supervised learning match the contextualized deduction relation, possibly qualified by a confidence defined by $conf(f \overset{\cdot}{\models}^K g) = \frac{|\tau^K(f) \cap \tau^K(g)|}{|\tau^K(f)|}$.

Considering two properties $f, g \in \mathcal{L}$, their contextual relation is determined by the sizes of 3 sets of objects $\pi_l^K(f,g) := |\tau^K(f) \setminus \tau^K(g)|$, $\pi_c^K(f,g) := |\tau^K(f) \cap \tau^K(g)|$ and $\pi_r^K(f,g) := |\tau^K(g) \setminus \tau^K(f)|$. For instance, f contextually entails g iff $\pi_l^K(f,g) = 0$, f and g are contextually separated iff $\pi_c^K(f,g) = 0$, or x is an increment of wq (cf. Section 4) iff $\pi_c^K(x,wq) \neq 0$ and $\pi_r^K(x,wq) \neq 0$.

Generalizing the LIS navigation to KD. The links of navigation defined in Section 4 can be defined on such contextual relations w.r.t. the working query, as it can be seen on the following reformulation of *Link*:

$Link^K(wq, wv) =$

$\lceil\{(x, \pi_c^K(x,wq)) \in X \times \mathbb{N} \mid x \overset{\cdot}{\models} wv, wv \overset{\cdot}{\not\models} x, \pi_c^K(x,wq) > 0\}\rceil$.

We propose to generalize the search for links of navigation into the search for some contextual properties *Propr*.

Definition 9 $Propr^K(wq, wv) :=$

$\lceil\{(x, label^K(x,wq)) \in X \times Label \mid x \overset{\cdot}{\models} wv, wv \overset{\cdot}{\not\models} x, propr^K(x,wq)\}\rceil$.

where *propr* and *label* are respectively a predicate and an application defined with π_l, π_c, and π_r. The predicate *propr* specifies which contextual properties are searched for. The application *label* associates to each property a value belonging to *Label* and indicating to the user the relevance of this property. Properties can then be ordered according to their label. Now, for each kind of contextual property of interest, we can define a new LIS command similar to the command `ls` for searching for this kind of properties in the context. As an illustration, the following paragraph defines two such commands. It is important to notice that sets of properties displayed by these new commands depends strongly on the working query. So, they cannot produce an exhaustive knowledge like data-analysis, but are only useful for discovering some knowledge underlying a context through a navigation-like process. This *human-centered knowledge discovery process* is rather different from the most common approach of data analysis and data-mining, but has already been advocated [HSWW00,KGLB00].

Searching for necessary or sufficient properties. First, we define some common notions such as *support* and *confidence* [HSWW00] of a rule $f \rightarrow g$ in a context K:

$supp^K(f \rightarrow g) := \pi_c^K(f,g)$ and $conf^K(f \rightarrow g) := \dfrac{\pi_c^K(f,g)}{\pi_l^K(f,g) + \pi_c^K(f,g)}$.

We define a *necessary property* as a property x entailed by the working query wq with a confidence greater than $conf_{min}$. This leads to the following instantiations of *propr* and *label* in the definition of *Nec*:

$propr^K(x,wq) :\Longleftrightarrow conf^K(wq \rightarrow x) \geq conf_{min}$,

$label^K(x,wq) := conf^K(wq \rightarrow x) \in [0,1]$.

This results in the new command `nec` that takes as argument $conf_{min}$ (set to > 0.0 by default) in addition to a view. This command enables the user to navigate among properties common to all objects of the working concept, and so in its intent. In the left part of Table 2, we search for the necessary properties of articles written by Sowa or Mineau. The 2nd query shows that all of these articles have fields `type`, `author`, `year`, and `title` defined. The 3rd query lists title words with confidence greater than 0.15: “Conceptual Graph” or “CG” appears in more than half of the considered articles (note that the formula `title: "Conceptual Graph" | "CG"` has been added to X manually with command `mkdir`, unlike other formulas), which is not very surprising, but “Formal” and “Context” also appear in more than a quarter of them.

```
(1) cd /author: Sowa | Mineau
(2) nec /
  1.000 type: *
  1.000 author: *
  1.000 year: ..
  1.000 title: *
  18 object(s)
(3) nec 0.15 title: *
  0.167 title: "Definition"
  0.167 title: "Constraints"
  0.222 title: "Represent"
  0.222 title: "Processes"
  0.278 title: "Context"
  0.278 title: "Formal"
  0.500 title: "Graph"
  0.556 title: "Concept"
  0.556 title: "Conceptual Graph" | "CG"
  18 object(s)
```

```
(1) cd /title: Knowledge
(2) suf /
  27 0.129 type: *
  27 0.129 author: *
  27 0.129 year: ..
  27 0.129 title: *
  27 object(s)
(3) suf 2 0.2 author: *
  2 0.286 author: "Ellis"
  2 0.400 author: "Angelova"
  2 0.667 author: "Bontcheva"
  2 1.000 author: "Gaines"
  3 0.375 author: "Dick"
  3 0.429 author: "Lukose"
  3 0.500 author: "Cyre"
  4 0.667 author: "Martin"
  27 object(s)
```

Table2. Examples of Knowledge Extraction in the BibTEX context.

We define a *sufficient property* as a property x that entails the working query wq with a support greater than $supp_{min}$, and with a confidence greater than $conf_{min}$. This leads to the following instantiations of *propr* and *label* in the definition of Suf:

$$propr^K(x, wq) :\Leftrightarrow supp^K(x \to wq) \geq supp_{min} \wedge conf^K(x \to wq) \geq conf_{min},$$
$$label^K(x, wq) := (supp^K(x \to wq), conf^K(x \to wq)) \in \mathbb{N} \times [0, 1].$$

This results in the new command `suf` that takes as argument $supp_{min}$ (set to 1 by default) and $conf_{min}$ (set to > 0.0 by default) in addition to a view. This command extracts some properties with which the working query property is always, or at least often, associated. Therefore, it can help to build some decision procedure for the working property, which is the issue of supervised learning. In the right part of Table 2, we search for the sufficient properties of articles whose title contains the word “Knowledge”. The 2nd query just shows these articles represent 12.9% of the whole context, and serves mainly as a menu of the available fields. Then, the 3rd query lists author names with support greater than 2, and confidence greater than 0.2: e.g., Martin has written 4 articles about “Knowledge”, which consists in 2/3 of his articles, and Gaines has written 2 articles, both talking about knowledge.

Commands `nec` and `suf` both enable the search for association rules whose left or right part is fixed on the working query. In an application built on top of LIS, they could be combined to find all association rules between two sets of properties (ex. between authors and title words, in the BibTEX context).

6 Future Work

Our most practical perspective is to design a *logical file system*, which would implement the ideas we have presented in this article. The expected advantage is to offer the services described here at a standard system level that is accessible for every application. So doing, even applications that do not know about logical information systems (like e.g., compilers) would benefit from it.

A graphical user-interface to logical file systems would allow to display in an integrated fashion the working query, the working view, and the corresponding extent and set of links. For instance, a graphical interface for keeping trace of navigation, like what is becoming standard for file browsers, has been already experimented for a simple logic (attributes with values) but should be developed further. This amounts to keep a trace of the path from the start of the navigation to the current place. Moreover, the set of links could be presented graphically as a diagram of ordered formulas. A further refinement is to take into account the contextualized deduction, to get something similar to concept lattices derived from scales [CS00]. This amounts to represent an overview of possible future navigations.

The *Web* can also be explored using our techniques if one considers answers to web-queries as a formal context into which to navigate.

There is also a connection with natural language that we wish to explore further (see Section 4). We believe that a logical information system can provide the rational for a human-machine interface in natural language, in which both the human being and the machine could submit assertions and queries. In this case, logics that have been widely used for representing natural language semantics seem to be the right choice. One such logic is the deduction relation of Conceptual Graphs [Sow84,Sow99].

7 Conclusion

We have presented the specifications of a Logical Information System based on (Logical) Concept Analysis. As opposed to previous attempt of using Concept Analysis for organizing data, we do not propose to navigate directly in the concept lattice. Instead, we use the contextualized logic (i.e., the logical view of the concept lattice) to evaluate the relevance of navigation links. Those that do not narrow the focus of the search are called *views*. They only restrict the language of available navigation links. Other links, that do narrow the focus of the search, are called *increments*. They can be used to come closer to some place of interest.

In this way, standard commands of a file system shell can be mimicked in a logical context. However, a simple generalization of the definition of links forms a framework in which operations of data-analysis or data-mining can be expressed. Using this framework, purely symbolic navigation as well as statistical exploration can be integrated smoothly as variants of the same generic operation.

References

[CS00] Richard Cole and Gerd Stumme. CEM - a conceptual email manager. In Guy Mineau and Bernhard Ganter, editors, *International Conference on Conceptual Structures*, LNCS 1867, pages 438–452. Springer, 2000.

[FR00a] Sébastien Ferré and Olivier Ridoux. A file system based on concept analysis. In Yehoshua Sagiv, editor, *International Conference on Rules and Objects in Databases*, LNCS 1861, pages 1033–1047. Springer, 2000.

[FR00b] Sébastien Ferré and Olivier Ridoux. A logical generalization of formal concept analysis. In Guy Mineau and Bernhard Ganter, editors, *International Conference on Conceptual Structures*, LNCS 1867, pages 371–384. Springer, 2000.

[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.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations.* Springer, 1999.

[HSWW00] Joachim Hereth, Gerd Stumme, Rudolf Wille, and Uta Wille. Conceptual knowledge discovery and data analysis. In Guy Mineau and Bernhard Ganter, editors, *International Conference on Conceptual Structures*, LNCS 1867, pages 421–437. Springer, 2000.

[KGLB00] Pascale Kuntz, Fabrice Guillet, Rémi Lehn, and Henri Briand. A user-driven process for mining association rules. In D.A. Zighed, J. Komorowski, and J. Zytkow, editors, *Principles of Knowledge Discovery and Data mining*, LNAI 1910, pages 483–489. Springer-Verlag, 2000.

[KS94] M. Krone and G. Snelting. On the inference of configuration structures from source code. In *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, *Principles of Data Mining and Knowledge Discovery*, LNAI 1704, pages 384–391, Berlin, 1999. Springer.

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

[Pre97] Susanne Prediger. Logical scaling in formal concept analysis. *LNCS 1257*, pages 332–341, 1997.

[PS99] Susanne Prediger and Gerd Stumme. Theory-driven logical scaling. In *International Workshop on Description Logics*, volume 22, Sweden, 1999.

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

[Sow84] John Sowa. *Conceptual structures. Information processing in man and machine.* Addison-Wesley, Reading, US, 1984.

[Sow99] John Sowa. Conceptual graphs: Draft proposed American National Standard. In William Tepfenhart and Walling Cyre, editors, *International Conference on Conceptual Structures*, LNAI 1640, pages 1–65, Berlin, 1999. Springer.