HAL
open science

Une généralisation logique de l'analyse de concepts formels

Sébastien Ferré, Olivier Ridoux

To cite this version:

Sébastien Ferré, Olivier Ridoux. Une généralisation logique de l'analyse de concepts formels. [Rapport de recherche] RR-3820, INRIA. 1999. inria-00072838

HAL Id: inria-00072838

https://inria.hal.science/inria-00072838v1

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Une généralisation logique de l'analyse de concepts formels

Sébastien Ferré, Olivier Ridoux

N°3820

Décembre 1999

THÈME 2

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--3820--FR

Une généralisation logique de l'analyse de concepts formels

Sébastien Ferré, Olivier Ridoux

Thème 2 — Génie logiciel
et calcul symbolique
Projet Lande

Rapport de recherche n° 3820 — Décembre 1999 — 26 pages

Résumé : Nous proposons une généralisation de l'analyse de concepts formels (ACF) dans laquelle les ensembles d'attributs sont remplacés par des expressions d'une logique presque arbitraire. Nous prouvons que toute l'ACF peut être reconstruite sur cette base. Nous montrons qu'à partir de toute logique utilisée à la place des ensembles d'attributs, on peut dériver une logique contextualisée qui prend en compte le contexte formel et qui est isomorphe au treillis de concepts. Nous comparons ensuite la généralisation de l'ACF aux extensions qui y ont déjà été apportées. Enfin, nous présentons nos perspectives d'application aux systèmes d'information.

Mots-clé : analyse de concepts, treillis de concepts, logique, contexte, systèmes d'information, interrogation, navigation.

(Abstract: pto)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

A Logical Generalization of Formal Concept Analysis

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 contextual logic that takes into account the formal context and that is isomorphic to the concept lattice. We then compare the generalization of FCA to existing extensions. Endly, we present our perspectives of application to information systems.

Key-words: concept analysis, concept lattice, logic, context, information systems, querying, browsing.

1 Introduction

Étant donné un *contexte formel* $(\mathcal{O},\mathcal{A},I)$, où $\mathcal{O}$ est un ensemble fini d'*objets*, $\mathcal{A}$ est un ensemble fini d'*attributs*, I est une relation entre les objets et les attributs (c-à-d., un sous-ensemble de $\mathcal{O} \times \mathcal{A}$), l'*Analyse de Concepts Formels* (ACF [Wil82, GW99a], Chapitre 11 dans [DP90]) définit des *concepts* comme des ensembles maximaux d'objets partageant les mêmes attributs. Plus formellement, un concept est un couple (O,A) où O est un sous-ensemble de $\mathcal{O}$, A est un sous-ensemble de $\mathcal{A}$, tel qu'on a la relation suivante :

$$\begin{array}{ll} \sigma(O) = A \text{ et } \tau(A) = O & \\ \text{où,} \quad \sigma : 2^{\mathcal{O}} \to 2^{\mathcal{A}} & \sigma(O) := \{a \in \mathcal{A} | \forall o \in O : (o,a) \in I\} \\ \text{et,} \quad \tau : 2^{\mathcal{A}} \to 2^{\mathcal{O}} & \tau(A) := \{o \in \mathcal{O} | \forall a \in A : (o,a) \in I\} \end{array}$$

La fonction σ donne pour tout ensemble d'objets l'ensemble des attributs partagés par ces objets. La fonction τ donne pour tout ensemble d'attributs l'ensemble des objets qui possèdent tous ces attributs. La composante O d'un concept est appelée son *extension*, et la composante A est appelée son *intention*. Le théorème fondamental de l'ACF est que l'ensemble de tous les concepts qui peuvent être construits sur un contexte formel donné forme un treillis complet quand il est ordonné par l'inclusion ensembliste des extensions de concept. En fait, la paire $\sigma - \tau$ constitue une connection de Galois. Rappelons que deux applications $\sigma : P \to Q$ et $\tau : Q \to P$, où $\langle P; \leq_P \rangle$ et $\langle Q; \leq_Q \rangle$ sont des ensembles partiellement ordonnés, forment une connection de Galois[1] si

$$p \leq_P \tau(q) \iff \sigma(p) \leq_Q q.$$

L'ACF est connue pour ses applications dans de nombreux domaines, tels que la représentation de la structure modulaire des logiciels [Sne98], la navigation dans la documentation logicielle [Lin95], dans le génie logiciel [KS94] et dans plusieurs applications en sciences humaines. L'intérêt de l'ACF comme outil de navigation en général a aussi été reconnu [GMA93].

Ces divers domaines d'application apportent le besoin de contextes formels plus sophistiqués que la simple présence/absence d'attributs. Par exemple, de nombreux domaines d'application utilisent des valeurs numériques (ex. longueurs, prix, âges) et on ressent assez souvent le besoin d'exprimer la négation et la disjonction. Dans un cadre beaucoup plus spécialisé, on peut aussi imaginer utiliser les types de composants logiciels au lieu des attributs[2]. Plusieurs enrichissements des attributs ont déjà été proposés : par exemple, des attributs multivalués [GW99a] et des termes du 1er ordre [CM98]. Cependant, aucune extension de l'ACF ne couvre tous ces domaines concrets, et ne peut prétendre couvrir tous les domaines concrets imaginables. Donc, nous proposons de construire un cadre général de l'analyse de concepts, l'Analyse de Concepts Logique (ACL), dans lequel la logique des attributs devient un paramètre. Cela permettra d'instancier le cadre général simplement en l'appliquant à une logique dédiée.

Dans la suite de ce rapport, on désignera par ACF ou AC *classique* la forme originale de l'analyse de concepts, tandis qu'on désignera par ACL ou AC *généralisée* l'analyse de concepts telle qu'elle est développée dans ce rapport. Le terme AC sera employé pour parler de ces deux formes à la fois. La section 2 part de l'AC classique (ACF) pour arriver à sa forme généralisée qu'est l'ACL. La section 3 définit comment une logique contextualisée peut être dérivée d'un contexte formel. La section 4 discute de caractéristiques possibles de la logique employée en paramètre de l'ACL. La section 5 montre la contribution de l'ACL comparé à des extensions existantes de l'AC classique. La section 6 explique comment l'ACL peut fournir un modèle d'organisation intéressant pour des systèmes d'informations dans lesquels les objets sont décrits par des formules logiques.

2 De l'ACF à l'ACL

2.1 Présentation générale

On commence tout d'abord par reformuler l'AC classique de telle sorte que l'AC généralisée en découle plus naturellement, par un simple isomorphisme. Cette reformulation commence par remplacer le contexte $(\mathcal{O},\mathcal{A},I)$ par $(\mathcal{O},2^{\mathcal{A}},i)$, où $2^{\mathcal{A}}$ est l'ensemble des parties de $\mathcal{A}$ et i est une fonction de $\mathcal{O}$ dans $2^{\mathcal{A}}$ définie par $i(o) := \{a \in \mathcal{A} | (o,a) \in I\}$. On n'a ni perdu, ni ajouté d'information et les deux représentations sont équivalentes :

$$(o,a) \in I \iff i(o) \supseteq \{a\}.$$

1. Les connections de Galois sont aussi employées en interprétation abstraite [CC92]. Dans ce domaine, les applications σ et τ correspondent respectivement à des fonctions d'abstraction et de concrétisation.

2. Voir Di Cosmo [Di 95] pour un traitement formel de l'idée d'utiliser les types pour la récupération de fonctions dans une librairie.

Ensuite, si on reformule les fonctions σ et τ avec la fonction i plutôt que la relation I, on obtient facilement les égalités suivantes :

$$\sigma(O) = \bigcap_{o \in O} i(o) \quad (1)$$

$$\tau(A) = \{o \in \mathcal{O} | i(o) \supseteq A\} \quad (2)$$

Le reste de la théorie de l'ACF est inchangé.

Maintenant, en consultant attentivement les preuves de l'ACF et en considérant la connection entre structures algébriques et treillis [DP90], on constate que le pré-requis nécessaire et suffisant à l'ACF est que $\langle 2^{\mathcal{A}}; \supseteq \rangle$ soit un treillis dont le supremum (plus petit majorant) est $\cap$ et l'infimum (plus grand minorant) est $\cup$. On peut donc considérer $2^{\mathcal{A}}$ comme une logique où $\supseteq$ est la relation de déduction, $\cap$ est l'opération de disjonction et $\cup$ est l'opération de conjonction.

A partir de cette interprétation de l'ACF, il devient naturel de la généraliser en remplaçant le domaine des ensembles d'attributs $2^{\mathcal{A}}$ par un domaine de formules arbitraire $\mathcal{L}$ auquel on associe une relation de déduction $\dot{\models}$, une opération de disjonction $\dot{\vee}$ et une opération de conjonction $\dot{\wedge}$ (les points sur les symboles sont seulement destinés à les différencier de ceux de la métalangue utilisée dans ce document). Pour que les résultats de l'ACF soient maintenus dans la forme généralisée (ACL), il est nécessaire et suffisant que $\langle \mathcal{L}; \dot{\models} \rangle$ soit un treillis et que $\dot{\vee}$ et $\dot{\wedge}$ soient respectivement son supremum et son infimum. D'après la correspondance existant entre structures algébriques et treillis [DP90], cela revient à poser les conditions suivantes sur $\mathcal{L}$:

- $\langle \mathcal{L}; \dot{\vee}, \dot{\wedge} \rangle$ est une structure algébrique satisfaisant au moins les propriétés suivantes pour tout $f, g, h \in \mathcal{L}$:
  - $f \dot{\vee} (g \dot{\vee} h) \doteq (f \dot{\vee} g) \dot{\vee} h$ $\quad f \dot{\wedge} (g \dot{\wedge} h) \doteq (f \dot{\wedge} g) \dot{\wedge} h$ (associativité)
  - $f \dot{\vee} g \doteq g \dot{\vee} f$ $\quad f \dot{\wedge} g \doteq g \dot{\wedge} f$ (commutativité)
  - $f \dot{\vee} f \doteq f$ $\quad f \dot{\wedge} f \doteq f$ (idempotence)
  - $f \dot{\vee} (f \dot{\wedge} g) \doteq f$ $\quad f \dot{\wedge} (f \dot{\vee} g) \doteq f$ (absorption)
- $\forall f, g \in \mathcal{L} : f \dot{\models} g \iff f \dot{\vee} g \doteq g \iff f \dot{\wedge} g \doteq f$

Exemple. Un exemple de logique utilisable dans l'ACL est la logique propositionnelle, où les formules sont appelées *propositions*. Syntaxiquement, l'ensemble des propositions $\mathcal{P}$ contient des propositions atomiques (prises dans un ensemble $\mathcal{A}$ [3]), les formules particulières 0 et 1 et est clos par les connecteurs binaires $\wedge$ et $\vee$ et par le connecteur unaire $\neg$. Sémantiquement, les interprétations sont des parties de l'ensemble des propositions atomiques $\mathcal{A}$. L'ensemble des modèles d'une proposition P est noté $M(P)$ et est défini inductivement sur la structure des propositions de la façon suivante :

$$\begin{array}{lcl} M(a) & = & \{A \in 2^{\mathcal{A}} | a \in A\}, \text{ pour tout } a \in \mathcal{A} \\ M(0) & = & \emptyset \\ M(1) & = & 2^{\mathcal{A}} \\ M(P \wedge Q) & = & M(P) \cap M(Q) \\ M(P \vee Q) & = & M(P) \cup M(Q) \\ M(\neg P) & = & 2^{\mathcal{A}} \setminus M(P). \end{array}$$

On définit alors une relation de déduction $\models$ sur $\mathcal{P}$ par $P \models Q \iff M(P) \subseteq M(Q)$. La relation d'équivalence sur les propositions est noté $\equiv$ et est défini par $P \equiv Q \iff M(P) = M(Q)$. Le tableau suivant présente de façon synthétique la relation d'instanciation de la logique arbitraire $\mathcal{L}$ de l'ACL vers la logique propositionnelle $\mathcal{P}$:

$\mathcal{L}$	$\dot{\vee}$	$\dot{\wedge}$	$\dot{\models}$	$\doteq$
$\mathcal{P}$	$\vee$	$\wedge$	$\models$	$\equiv$

La logique $\langle \mathcal{P}; \vee, \wedge, \models \rangle$ satisfait bien les conditions de l'ACL, car sa sémantique $\langle 2^{2^{\mathcal{A}}}; \subseteq \rangle$ est un treillis de supremum $\cup$ et d'infimum $\cap$.

2.2 Contexte et connection de Galois

Dans cette section, on applique l'idée introduite à la section précédente pour donner une formulation de l'AC généralisée à une logique presque arbitraire : l'Analyse de Concepts Logique (ACL). La transposition de

3. Dans cet exemple $\mathcal{A}$ désigne un ensemble de propositions atomiques et pas un ensemble d'attributs comme dans l'ACF. Cependant, il est possible d'établir une correspondance formelle entre les deux [GW99b] (cf. section 5.3).

l'ACF à l'ACL consiste à reformuler les occurrences de I avec i et à remplacer $\supseteq$, $\cap$ et $\cup$ respectivement par $\dot{\models}$, $\dot{\vee}$ et $\dot{\wedge}$. Les preuves sont obtenues à partir du chapitre 11 de [DP90] et en appliquant cette transposition.

Définition 1 (contexte) *Un* contexte (formel) *est un triplet* $(\mathcal{O},\mathcal{L},i)$ *où :*

- $\mathcal{O}$ *est un ensemble fini d'objets,*
- $\langle\mathcal{L};\dot{\models}\rangle$ *est un treillis de formules, de supremum* $\dot{\vee}$ *et d'infimum* $\dot{\wedge}$ *;* $\mathcal{L}$ *dénote une logique dont la relation de déduction est* $\dot{\models}$ *et dont les opérations de disjonction et de conjonction sont respectivement* $\dot{\vee}$ *et* $\dot{\wedge}$*,*
- i *est une fonction de* $\mathcal{O}$ *dans* $\mathcal{L}$ *qui associe à chaque objet une formule qui décrit l'intention*[4] *de l'objet.*

L'ACF définit à partir d'un contexte deux fonctions σ et τ, qui constituent une connection de Galois. La connection de Galois se fait ici entre des ensembles d'objets (les extensions) et des formules (les intentions). On obtient leur expression en transposant les égalités (1) et (2) :

Définition 2 *Soit* $(\mathcal{O},\mathcal{L},i)$ *un contexte,* $O \subseteq \mathcal{O}$ *et* $f \in \mathcal{L}$*.*

$$\begin{array}{lll} \sigma : 2^{\mathcal{O}} \to \mathcal{L}, & \sigma(O) & := \dot{\bigvee}_{o\in O} i(o) \\ \tau : \mathcal{L} \to 2^{\mathcal{O}}, & \tau(f) & := \{o \in \mathcal{O} | i(o) \dot{\models} f\} \end{array}$$

Lemme 1 *Soit* $(\mathcal{O},\mathcal{L},i)$ *un contexte,* $O,O_j \subseteq \mathcal{O}$ *et* $f,f_j \in \mathcal{L}$ *pour tout* $j \in J$*, où* J *est un ensemble d'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)	$\forall o \in \mathcal{O} : \sigma(\tau(i(o))) \dot{=} i(o)$		

Démonstrations :

(i) $\tau(\sigma(O)) = \{o \in \mathcal{O} | i(o) \dot{\models} \dot{\bigvee}_{o'\in O} i(o')\}$.
Pour tout $o \in O$, $i(o) \dot{\models} \dot{\bigvee}_{o'\in O} i(o') \Longrightarrow o \in \tau(\sigma(O))$, donc $O \subseteq \tau(\sigma(O))$,

(i') $\sigma(\tau(f)) = \dot{\bigvee}_{o\in\mathcal{O}|i(o)\dot{\models}f} i(o) \Longrightarrow \sigma(\tau(f)) \dot{\models} f$,

(ii) $O_1 \subseteq O_2 \Longrightarrow \dot{\bigvee}_{o\in O_1} i(o) \dot{\models} \dot{\bigvee}_{o\in O_2} i(o) \Longrightarrow \sigma(O_1) \dot{\models} \sigma(O_2)$,

(ii') $f_1 \dot{\models} f_2$
$\Longrightarrow \forall o \in \mathcal{O} : (i(o) \dot{\models} f_1) \Longrightarrow (i(o) \dot{\models} f_2)$ (transitivité de $\dot{\models}$)
$\Longrightarrow \forall o \in \mathcal{O} : o \in \tau(f_1) \Longrightarrow o \in \tau(f_2)$
$\Longrightarrow \tau(f_1) \subseteq \tau(f_2)$,

(iii) $O \subseteq \tau(\sigma(O))$ (lemme 1.(i)) $\Longrightarrow \sigma(O) \dot{\models} \sigma(\tau(\sigma(O)))$ (lemme 1.(ii))
et $\sigma(\tau(\sigma(O))) \dot{\models} \sigma(O)$ (lemme 1.(i')),

(iii') $\sigma(\tau(f)) \dot{\models} f$ (lemme 1.(i')) $\Longrightarrow \tau(\sigma(\tau(f))) \subseteq \tau(f)$ (lemme 1.(ii'))
et $\tau(f) \subseteq \tau(\sigma(\tau(f)))$ (lemme 1.(i)),

(iv)

$$\begin{array}{lll} \dot{\bigvee}_{j\in J} \sigma(O_j) & \dot{=} & \dot{\bigvee}_{j\in J} \dot{\bigvee}_{o\in O_j} i(o) \\ & \dot{=} & \dot{\bigvee}_{o\in\bigcup_{j\in J} O_j} i(o) \text{ (associativité et commutativité de } \dot{\vee}) \\ & \dot{=} & \sigma(\bigcup_{j\in J} O_j), \end{array}$$

(iv')

$$\begin{array}{lll} \bigcap_{j\in J} \tau(f_j) & = & \bigcap_{j\in J} \{o \in \mathcal{O} | i(o) \dot{\models} f_j\} \\ & = & \{o \in \mathcal{O} | \forall j \in J : i(o) \dot{\models} f_j\} \\ & = & \{o \in \mathcal{O} | i(o) \dot{\models} \dot{\bigwedge}_{j\in J} f_j\} \\ & = & \tau(\dot{\bigwedge}_{j\in J} f_j), \end{array}$$

(v) $\sigma(\tau(i(o)) \dot{=} \sigma(\{o' \in \mathcal{O} | i(o') \dot{\models} i(o)\}) \dot{=} (\dot{\bigvee}_{o'\in\mathcal{O}|i(o')\dot{\models}i(o)} i(o')) \dot{=} i(o)$
(car $o \in \{o' \in \mathcal{O} | i(o') \dot{\models} i(o)\}$). ■

Les résultats exprimés dans ce lemme sont simplement une transposition de ceux de l'AC classique, excepté le lemme 1.(v) dont la démonstration ne fait intervenir que les définitions de σ et τ.

4. "intention" doit être pris ici dans un sens informel. Ces formules ne font que véhiculer les caractéristiques des objets que l'utilisateur a l'intention d'utiliser. A l'opposé, le terme "intention" sera repris de façon formelle dans la définition des concepts (cf. définition 3).

objet	description
x	a
y	b
z	$c \wedge (a \vee b)$

TAB. 1 – *Descriptions des objets du contexte K_{ex}.*

Exemple. On donne ici un exemple de contexte formel qui permettra d'illustrer la suite du développement de l'ACL. Le contexte K_{ex} donné ci-après est volontairement petit et simple, car il est destiné à illustrer des notions théoriques et non à présenter une application réaliste de l'ACL. La logique employée est la logique propositionnelle $\mathcal{P}$ (présentée en exemple à la page 4) pour un ensemble de propositions atomiques $\mathcal{A} = \{a,b,c\}$. Le contexte K_{ex} est défini par $(\mathcal{O}_{ex},\mathcal{P},i_{ex})$, où $\mathcal{O}_{ex} = \{x,y,z\}$ et où la fonction i_{ex} qui décrit les objets est définie dans la table 1.

2.3 Les concepts

Dans cette sous-section, on montre qu'en utilisant nos définitions d'un contexte et de la connection de Galois σ–τ, on retrouve tous les résultats existants concernant les concepts. Tout d'abord, on rappelle la définition des concepts (cf. chapitre 11 dans [DP90]), simplement en adaptant les notations.

Définition 3 (concept) *Dans un contexte $(\mathcal{O},\mathcal{L},i)$, un* concept *est une paire $c = (O,f)$ où $O \subseteq \mathcal{O}$ et $f \in \mathcal{L}$, tel que $\sigma(O) \doteq f$ et $\tau(f) = O$.*

*L'ensemble d'objets O est l'*extension *du concept ($ext(c)$), tandis que la formule f en est l'*intention *($int(c)$).*

La grande différence avec l'AC classique est que l'intention est maintenant une formule de la logique considérée $\mathcal{L}$. L'ensemble de tous les concepts existant dans un contexte $(\mathcal{O},\mathcal{L},i)$ est noté $\mathcal{C}(\mathcal{O},\mathcal{L},i)$, et est ordonné partiellement par $\leq^c$ défini comme suit.

Définition 4 *Soient (O_1,f_1) et (O_2,f_2) dans $\mathcal{C}(\mathcal{O},\mathcal{L},i)$*

$$(O_1,f_1) \leq^c (O_2,f_2) \Longleftrightarrow O_1 \subseteq O_2$$

Comme dans l'AC classique, cet ordre est compatible avec l'ordre sur les intentions.

Proposition 1 *Soient (O_1,f_1) et (O_2,f_2) dans $\mathcal{C}(\mathcal{O},\mathcal{L},i)$*

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

Démonstration :

- $O_1 \subseteq O_2 \Longrightarrow \sigma(O_1) \dot{\models} \sigma(O_2)$ (lemme 1.(ii))
  $\Longrightarrow f_1 \dot{\models} f_2$ (car (O_1,f_1) et (O_2,f_2) sont des concepts).
- $f_1 \dot{\models} f_2 \Longrightarrow \tau(f_1) \subseteq \tau(f_2)$ (lemme 1.(ii'))
  $\Longrightarrow O_1 \subseteq O_2$ (car (O_1,f_1) et (O_2,f_2) sont des concepts). ■

Comme pour l'AC classique, les définitions 3 et 4 mènent au *théorème fondamental* suivant.

Théorème 1 *Soient $(\mathcal{O},\mathcal{L},i)$ un contexte et J un ensemble d'indices. L'ensemble ordonné $\langle \mathcal{C}(\mathcal{O},\mathcal{L},i); \leq^c \rangle$ est un treillis complet, dont le supremum (plus petit majorant) et l'infimum (plus grand minorant) sont comme suit :*

$$\begin{array}{lcl} \bigvee^c_{j\in J}(O_j,f_j) & =^c & (\tau(\sigma(\bigcup_{j\in J} O_j)),\dot{\bigvee}_{j\in J} f_j) \\ \bigwedge^c_{j\in J}(O_j,f_j) & =^c & (\bigcap_{j\in J} O_j,\sigma(\tau(\dot{\bigwedge}_{j\in J} f_j))) \end{array}$$

Démonstration :

- ($\leq^c$ est un ordre partiel)
  $\leq^c$ est clairement un ordre partiel sur $\mathcal{C}(\mathcal{O},\mathcal{L},i)$ d'après sa définition et car $\subseteq$ est lui-même un ordre partiel sur les ensembles.
- ($\vee^c$ est un supremum dans $\mathcal{C}(\mathcal{O},\mathcal{L},i)$)
  Soit $(O_j,f_j) \in \mathcal{C}(\mathcal{O},\mathcal{L},i)$ pour tout $j \in J$.
  Par le lemme 1.(i') on a $\sigma(\tau(\dot{\bigvee}_{j\in J} f_j)) \dot{\models} \dot{\bigvee}_{j\in J} f_j$.
  De même, $\forall k \in J : f_k \dot{\models} \dot{\bigvee}_{j\in J} f_j$
  $\Longrightarrow \forall k \in J : \tau(f_k) \subseteq \tau(\dot{\bigvee}_{j\in J} f_j)$ (lemme 1.(ii'))
  $\Longrightarrow \forall k \in J : f_k \dot{\models} \sigma(\tau(\dot{\bigvee}_{j\in J} f_j))$ (lemmes 1.(ii), (iii) et car (O_k,f_k) est un concept)
  $\Longrightarrow \dot{\bigvee}_{j\in J} f_j \dot{\models} \sigma(\tau(\dot{\bigvee}_{j\in J} f_j))$. Donc, $\dot{\bigvee}_{j\in J} f_j \dot{=} \sigma(\tau(\dot{\bigvee}_{j\in J} f_j))$.
  Puis, par le lemme 1.(iv): $\tau(\dot{\bigvee}_{j\in J} f_j) = \tau(\dot{\bigvee}_{j\in J} \sigma(\tau(f_j))) = \tau(\sigma(\bigcup_{j\in J} \tau(f_j))) = \tau(\sigma(\bigcup_{j\in J} O_j))$.
  Il s'en suit que $(\tau(\sigma(\bigcup_{j\in J} O_j)), \dot{\bigvee}_{j\in J} f_j)$ est un concept. Appelons-le c.
  c est un majorant de $\{(O_j,f_j) | j \in J\}$ car $\forall k \in J : f_k \dot{\models} \dot{\bigvee}_{j\in J} f_j$.
  Si (O,f) est un majorant quelconque de $\{(O_j,f_j) | j \in J\}$, alors $\forall j \in J : f_j \dot{\models} f$, donc $\dot{\bigvee}_{j\in J} f_j \dot{\models} f$ ce qui donne $c \leq^c (O,f)$.
  Donc, on obtient $\bigvee^c_{j\in J}(O_j,f_j) =^c c$.
- ($\wedge^c$ est un infimum dans $\mathcal{C}(\mathcal{O},\mathcal{L},i)$)
  Soit $(O_j,f_j) \in \mathcal{C}(\mathcal{O},\mathcal{L},i)$ pour tout $j \in J$.
  Par le lemme 1.(i), $\bigcap_{j\in J} O_j \subseteq \tau(\sigma(\bigcap_{j\in J} O_j))$.
  De même, $\forall k \in J : \bigcap_{j\in J} O_j \subseteq O_k$
  $\Longrightarrow \forall k \in J : \sigma(\bigcap_{j\in J} O_j) \dot{\models} \sigma(O_k)$ (lemme 1.(ii))
  $\Longrightarrow \forall k \in J : \tau(\sigma(\bigcap_{j\in J} O_j)) \subseteq O_k$ (lemmes 1.(ii'), (iii') et car (O_k,f_k) est un concept)
  $\Longrightarrow \tau(\sigma(\bigcap_{j\in J} O_j)) \subseteq \bigcap_{j\in J} O_j$. Donc $\tau(\sigma(\bigcap_{j\in J} O_j)) = \bigcap_{j\in J} O_j$.
  Puis, par le lemme 1.(iv): $\sigma(\bigcap_{j\in J} O_j) \dot{=} \sigma(\bigcap_{j\in J} \tau(\sigma(O_j))) \dot{=} \sigma(\tau(\dot{\bigwedge}_{j\in J} \sigma(O_j))) \dot{=} \sigma(\tau(\dot{\bigwedge}_{j\in J} f_j))$.
  Il s'ensuit que $(\bigcap_{j\in J} O_j, \sigma(\tau(\dot{\bigwedge}_{j\in J} f_j)))$ est un concept. Appelons-le c.
  c est un minorant de $\{(O_j,f_j) | j \in J\}$ car $\forall k \in J : \bigcap_{j\in J} O_j \subseteq O_k$.
  Si (O,f) est un minorant quelconque $\{(O_j,f_j) | j \in J\}$, alors $\forall j \in J : O \subseteq O_j$ alors $O \subseteq \bigcap_{j\in J} O_j$ ce qui donne $(O,f) \leq^c c$.
  Donc, on obtient $\bigwedge^c_{j\in J}(O_j,f_j) =^c c$. ■

Exemple. La figure 1 représente le diagramme de Hasse du treillis de concept du contexte K_{ex} (introduit en exemple à la page 6). Les concepts y sont représentés par un numéro et une boîte contenant l'extension à gauche et l'intention à droite. Les concepts placés le plus haut sont les plus grands pour la relation d'ordre $\leq^c$. On observe que le treillis de concepts n'est pas isomorphe au treillis des parties de $\mathcal{O}_{ex}$. En effet, l'ensemble d'objets $\{x,y\}$ n'est pas une extension, car $\tau(\sigma(\{x,y\})) = \tau(a \vee b) = \{x,y,z\}$.

2.4 Étiquetage du treillis de concepts

De manière analogue à l'AC classique, on peut étiqueter le treillis de concepts par les objets et les formules.

Définition 5 *Soit $(\mathcal{O},\mathcal{L},i)$ un contexte formel.*
On définit une fonction d'étiquetage γ du treillis de concepts par les objets, et une fonction d'étiquetage μ du treillis de concepts par les formules :

$$\begin{array}{ll} \gamma : \mathcal{O} \to \mathcal{C}(\mathcal{O},\mathcal{L},i) & \gamma(o) := (\tau(\sigma(\{o\})),\sigma(\{o\})) \\ \mu : \mathcal{L} \to \mathcal{C}(\mathcal{O},\mathcal{L},i) & \mu(f) := (\tau(f),\sigma(\tau(f))). \end{array}$$

Les images des fonctions γ et μ sont bien des concepts d'après la définition 3 des concepts et les propriétés des fonctions σ et τ (lemme 1). Le lemme suivant donne des propriétés intéressantes de ces étiquetages du treillis de concepts. Avant de l'énoncer, on rappelle tout d'abord la définition de *sup-dense* [DP90].

Définition 6 *Un sous-ensemble P d'un treillis $\mathcal{P}$ est appelé* sup-dense *dans $\mathcal{P}$ si pour tout élément $p \in \mathcal{P}$ il existe un sous-ensemble A de P tel que $p = \bigvee_{\mathcal{P}} A$.*

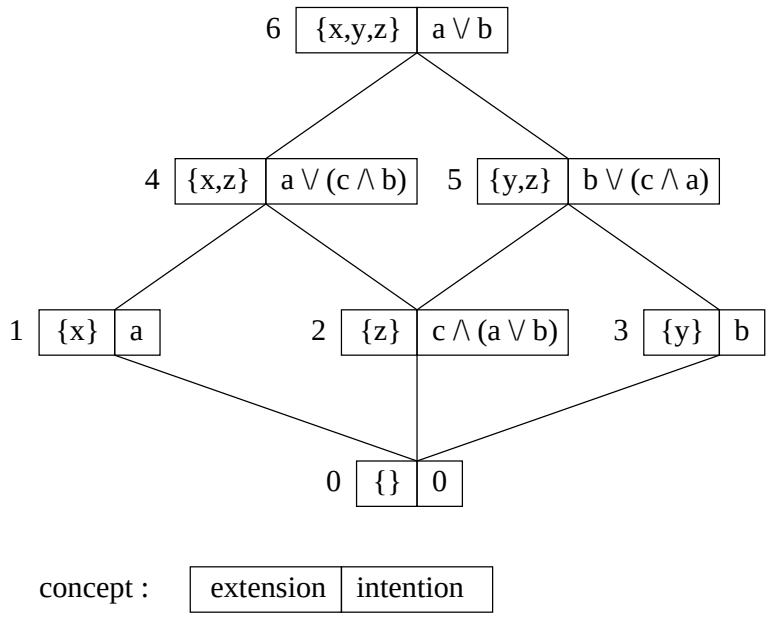

FIG. 1 – *Treillis des concepts formels du contexte K_{ex}.*

Lemme 2 *Soit $(\mathcal{O},\mathcal{L},i)$ un contexte et $o \in \mathcal{O}$, $f \in \mathcal{L}$, $c \in \mathcal{C}(\mathcal{O},\mathcal{L},i)$.*
(1) $\gamma(o) \leq^c c \iff o \in ext(c)$
(2) $\gamma(\mathcal{O})$ est sup-dense dans $\mathcal{C}(\mathcal{O},\mathcal{L},i)$
(3) $c \leq^c \mu(f) \iff int(c) \dot{\models} f$
(4) μ est surjective
(5) $i(o) \dot{\models} f \iff \gamma(o) \leq^c \mu(f)$
(6) $\mu(int(c)) =^c c$
(7) $int(\mu(f)) \dot{\models} f$.

Démonstrations :

(1) $\gamma(o) \leq^c c \iff ext(\gamma(o)) \subseteq ext(c) \iff \tau(\sigma(\{o\})) \subseteq ext(c)$ (définition de γ)
$\iff \{o\} \subseteq ext(c) \iff o \in ext(c)$ (lemmes 1.(i),(ii) et (ii')).

(2) Pour tout $(O,f) \in \mathcal{C}(\mathcal{O},\mathcal{L},i)$,

$$\begin{aligned}
\bigvee \gamma(O) &= \bigvee_{o\in O} \gamma(o) = \bigvee_{o\in O} (\tau(\sigma(\{o\})),\sigma(\{o\})) \\
&= (\tau(\sigma(\textstyle\bigcup_{o\in O} \tau(\sigma(\{o\})))),\dot{\bigvee}_{o\in O}\sigma(\{o\})) \\
&= (\tau(\sigma(\textstyle\bigcup_{o\in O} \tau(\sigma(\{o\})))),\sigma(\textstyle\bigcup_{o\in O}\{o\})) && \text{(lemme 1.(iv))} \\
&= (\tau(\sigma(\textstyle\bigcup_{o\in O} \tau(\sigma(\{o\})))),\sigma(O)) \\
&= (\tau(\sigma(\textstyle\bigcup_{o\in O} \tau(\sigma(\{o\})))),f) && \text{(car } (O,f) \text{ est un concept)}
\end{aligned}$$

(O,f) et $\bigvee \gamma(O)$ sont deux concepts de même intention, alors $\bigvee \gamma(O) = (O,f)$ et donc, $\gamma(\mathcal{O})$ est sup-dense dans $\mathcal{C}(\mathcal{O},\mathcal{L},i)$.

(3) $c \leq^c \mu(f) \iff int(c) \dot{\models} int(\mu(f)) \iff int(c) \dot{\models} \sigma(\tau(f))$ (définition de μ)
$\iff int(c) \dot{\models} f$ (lemmes 1.(i'),(ii) et (ii')).

(4) Pour tout $(O,f) \in \mathcal{C}(\mathcal{O},\mathcal{L},i)$,
$\mu(f) \doteq (\tau(f),\sigma(\tau(f))) \doteq (O,\sigma(O)) \doteq (O,f)$ (car (O,f) est un concept) et donc, μ est surjective (tout concept a au moins son intention comme étiquette).

(5)

$$\begin{aligned}
i(o) \dot{\models} f &\iff \dot{\bigvee}_{o\in\{o\}} i(o) \dot{\models} f \iff \sigma(\{o\}) \dot{\models} f \\
&\iff \tau(\sigma(\{o\})) \subseteq \tau(f) \text{ (lemme 1)} \\
&\iff (\tau(\sigma(\{o\})),\sigma(\{o\})) \leq^c (\tau(f),\sigma(\tau(f))) \\
&\iff \gamma(o) \leq \mu(f).
\end{aligned}$$

(6) $\mu(int(c)) =^c (\tau(int(c)),\sigma(\tau(int(c)))) =^c (ext(c),int(c)) =^c c$.

(7) $int(\mu(f)) \doteq \sigma(\tau(f))$ (par définition de μ)
donc $int(\mu(f)) \dot{\models} f$ (lemme 1.(i')). ■

Concernant la fonction d'étiquetage γ, le lemme 2.(1) montre que $\gamma(o)$ est le plus petit des concepts dont l'extension contient o ; et le lemme 2.(2) établit que tout concept est le supremum de concepts étiquetés par au moins un objet. Concernant la fonction d'étiquetage μ, le lemme 2.(3) montre que $\mu(f)$ est le plus grand des concepts dont l'intention entraîne logiquement f ; et le lemme 2.(4) établit que tout concept est étiqueté par au moins une formule. Le lemme 2.(5) montre qu'un objet quelconque o satisfait toutes les formules étiquetant les concepts plus grands que $\gamma(o)$, et seulement celles-là. De façon analogue, ce lemme montre qu'une formule quelconque f a dans son extension $\tau(f)$ tous les objets étiquetant les concepts plus petits que $\mu(f)$, et seulement ceux-là. L'étiquetage du treillis de concepts permet donc de retrouver toutes les informations du contexte. Concernant la relation entre intention et étiquette d'un concept, le lemme 2.(6) montre que tout concept est étiqueté par son intention ; et le lemme 2.(7) ajoute que toute formule étiquetant un concept est déductible logiquement de l'intention de ce concept.

Exemple. La figure 2 représente le même treillis de concepts que la figure 1, mais n'associe pas les mêmes informations aux concepts. Le numéro de chaque concept a été repris pour pouvoir l'identifier ; et on a placé à gauche de chaque concept les objets et à droite les formules de la forme $\bigvee A$ avec $A \subseteq \mathcal{A}$ ($\bigvee \emptyset \equiv 0$) qui l'étiquettent. Par exemple, le concept 1 est étiqueté par l'objet x (c-à-d. $\gamma(x) =^c 1$) et par la formule a (c-à-d. $\mu(a) =^c 1$). On a restreint l'étiquetage par les formules aux disjonctions, car il y a une infinité de formules dans $\mathcal{P}$ et tout de même 256 classes d'équivalence modulo $\equiv$. Ce dernier point implique que des concepts sont nécessairement étiquetés par plusieurs formules car le nombre des concepts est fini (ce qu'on observe effectivement sur le concept 6).

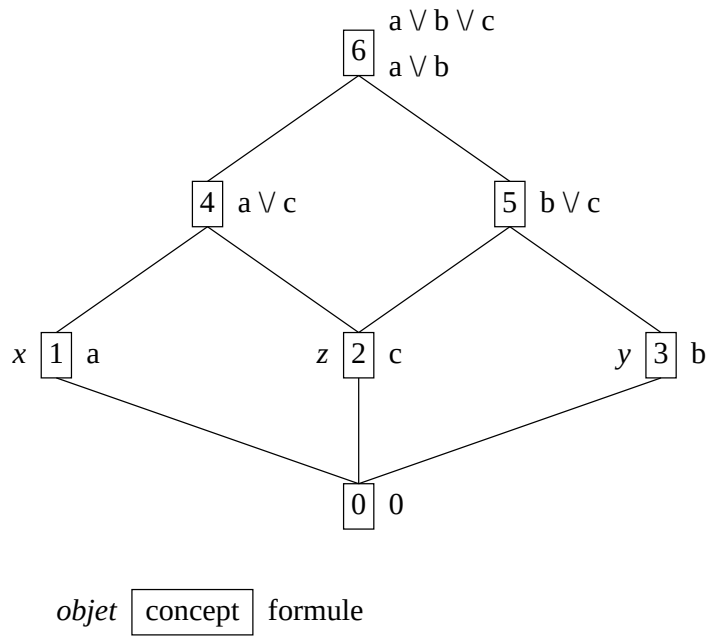

FIG. 2 – *Etiquetage du treillis de concepts.*

3 Logique contextualisée

Dans un contexte particulier, il est possible d'ordonner certaines propriétés, bien qu'elles ne soient pas comparables par $\dot{\models}$. Par exemple, si tous les oiseaux volent dans un contexte donné, alors on peut dire que la propriété "oiseau" entraîne contextuellement la propriété "vole", bien que l'on ait pas nécessairement $oiseau \dot{\models} vole$ (cela dépend de la logique choisie). On introduit une *relation de déduction contextualisée* qui concrétise cette

idée. C'est une généralisation des implications entre attributs, qui peuvent être utilisées dans l'AC classique pour des processus d'acquisition de connaissance [GW99a, Sne98].

Définition 7 (déduction contextualisée) *Soient $K = (\mathcal{O},\mathcal{L},i)$ un contexte et $f,g \in \mathcal{L}$.*
On dit que f entraîne contextuellement *g dans le contexte K, et on note $f \dot{\models}^K g$, si $\tau(f) \subseteq \tau(g)$, c-à-d. si tout objet qui satisfait f satisfait aussi g.*

La relation $\dot{\models}^K$ est clairement un pré-ordre, et on note $\dot{\equiv}^K$ la relation d'équivalence associée.

Définition 8 (logique contextualisée) *Soit $K = (\mathcal{O},\mathcal{L},i)$ un contexte.*
On appelle logique contextualisée[5] *l'ensemble partiellement ordonné $\langle \mathcal{L}^K; \dot{\models}^K \rangle$ où*

$$\mathcal{L}^K := \mathcal{L}/_{\dot{\equiv}^K}$$
$$\forall F,G \in \mathcal{L}^K : F \dot{\models}^K G \Longleftrightarrow \exists f \in F, g \in G : f \dot{\models}^K g.$$

Les éléments de $\mathcal{L}^K$ (classes d'équivalence de $\mathcal{L}$ modulo $\dot{\equiv}^K$) sont appelées formules contextualisées, *et la classe d'une formule $f \in \mathcal{L}$ est notée f^K.*

Lemme 3 *Soient $K = (\mathcal{O},\mathcal{L},i)$ un contexte, $f,g \in \mathcal{L}$ et $a,b \in \mathcal{C}(K)$.*
(1) $f \dot{\models} g \Longrightarrow f \dot{\models}^K g$
(2) $f \dot{\models}^K g \Longleftrightarrow \mu(f) \leq^c \mu(g)$
(3) $a \leq^c b \Longleftrightarrow int(a) \dot{\models}^K int(b)$
(4) $\sigma(\tau(f)) \dot{\equiv}^K f$
(5) $\forall o \in \mathcal{O} : i(o) \dot{\models}^K f \Longleftrightarrow i(o) \dot{\models} f$

Démonstration :
(1) $f \dot{\models} g \Longrightarrow \forall h \in \mathcal{L} : (h \dot{\models} f \Rightarrow h \dot{\models} g)$ (car $\dot{\models}$ est un ordre et est donc transitif)
$\Longrightarrow \forall o \in \mathcal{O} : (i(o) \dot{\models} f \Rightarrow i(o) \dot{\models} g) \Longrightarrow \forall o \in \mathcal{O} : (o \in \tau(f) \Rightarrow o \in \tau(g))$
$\Longrightarrow \tau(f) \subseteq \tau(g) \Longrightarrow f \dot{\models}^K g.$
(2) $f \dot{\models}^K g \Longleftrightarrow \tau(f) \subseteq \tau(g)$ (par définition de $\dot{\models}^K$)
$\Longleftrightarrow ext(\mu(f)) \subseteq ext(\mu(g)) \Longleftrightarrow \mu(f) \leq^c \mu(g).$
(3) $a \leq^c b \Longleftrightarrow \mu(int(a)) \leq^c \mu(int(b))$ (lemme 2.(6))
$\Longleftrightarrow int(a) \dot{\models}^K int(b)$ (lemme 3.(2)).
(4) $\mu(int(\mu(f))) =^c \mu(f)$ (lemme 2.(6)) $\Longrightarrow int(\mu(f)) \dot{\equiv}^K f$ (lemme 3.(2))
$\Longrightarrow \sigma(\tau(f)) \dot{\equiv}^K f$ (par définition de μ).
(5)
- $i(o) \dot{\models} f \Longrightarrow i(o) \dot{\models}^K f$ (lemme 3.(1))
- $i(o) \dot{\models}^K f \Longrightarrow \tau(i(o)) \subseteq \tau(f)$
  $\Longrightarrow \sigma(\tau(i(o))) \dot{\models} \sigma(\tau(f)) \dot{\models} f$ (lemme 1.(ii), (i'))
  $\Longrightarrow i(o) \dot{\models} f$ (lemme 1.(v)). ■

Le lemme 3.(1) montre que la déduction contextualisée $\dot{\models}^K$ est seulement une extension de la déduction initiale $\dot{\models}$. Les lemmes 3.(2) et (3) montrent que μ et *int* sont des morphismes entre $\langle \mathcal{L}; \dot{\models}^K \rangle$ et $\langle \mathcal{C}(K); \leq^c \rangle$. Le lemme 3.(4) identifie $\sigma(\tau(f))$ comme l'intention contextuellement équivalente à f. Le lemme 3.(5) montre que le contexte ne modifie pas les propriétés des objets : il n'ajoute ni ne supprime quelque propriété que ce soit.

Le contexte joue le rôle d'une théorie étendant la relation de déduction et permettant de nouvelles déductions. La logique contextualisée peut aussi être vue comme un moyen d'extraire des connaissances à partir d'un contexte. Deux types de connaissances peuvent ainsi être extraites : des connaissances sur le contexte par déduction, et des connaissances sur le domaine d'où est extrait le contexte par induction.

5. Cette logique contextualisée est de nature différente de la logique contextuelle introduite par Prediger [Pre98] : la première est une logique sur les formules, tandis que la deuxième est une logique sur les contextes.

Exemple. Grâce au morphisme μ entre la déduction contextualisée et l'ordre sur les concepts (lemme 3.(2)), on peut s'appuyer sur le treillis de concepts étiqueté (cf. figure 2) pour étudier la logique contextualisée dans le contexte K_{ex}. Par exemple, comme le concept 2 est inférieur au concept 5 on a la relation $c \models^{K_{ex}} b \vee c$, ce qui est déjà vrai dans $\mathcal{P}$. De manière générale, on vérifie bien que toutes les déductions valides dans $\mathcal{P}$ sont retrouvées avec la déduction contextualisée. L'examen du treillis étiqueté montre que le contexte ajoute de nouvelles déductions valides entre les formules : ex., $c \models^{K_{ex}} a \vee b$, $a \vee b \vee c \equiv^{K_{ex}} a \vee b$.

On considère maintenant les correspondances entre la logique contextualisée et le treillis de concepts. L'objectif est de montrer que le treillis de concepts forme une nouvelle logique, dérivée de $\mathcal{L}$ et adaptée au contexte : la logique contextualisée. Pour cela, on utilise les fonctions μ et int pour établir une correspondance entre les formules contextualisées et les concepts.

Théorème 2 *$\langle \mathcal{L}^K; \dot{\models}^K \rangle$ et $\langle \mathcal{C}(K); \leq^c \rangle$ sont isomorphes, avec μ^K comme isomorphisme des formules contextualisées vers les concepts et int^K comme isomorphisme des concepts vers les formules contextualisées, définies par*

$$\begin{array}{ll} \mu^K : \mathcal{L}^K \to \mathcal{C}(K) & \mu^K(F) := \mu(f) \text{ où } f \in F \\ int^K : \mathcal{C}(K) \to \mathcal{L}^K & int^K(c) := int(c)^K \end{array}$$

Démonstration :

Tout d'abord, la fonction μ^K est bien définie car pour tout $f_1, f_2 \in \mathcal{L}$ on a $f_1^K \dot{=}^K f_2^K \Longrightarrow f_1 \dot{\equiv}^K f_2 \Longrightarrow \mu(f_1) =^c \mu(f_2)$ (lemme 3.(2)). Montrons maintenant que μ^K et int^K sont mutuellement inverses et constituent une paire d'isomorphismes entre $\langle \mathcal{L}^K; \dot{\models}^K \rangle$ et $\langle \mathcal{C}(K); \leq^c \rangle$.

- $\forall F,G \in \mathcal{L}^K : F \dot{\models}^K G$
  $\Longrightarrow \exists f \in F, g \in G : f \dot{\models}^K g$
  $\Longrightarrow \exists f \in F, g \in G : \mu(f) \leq^c \mu(g)$ (lemme 3.(2))
  $\Longrightarrow \mu^K(F) \leq^c \mu^K(G)$ (définition de μ^K).
- $\forall a,b \in \mathcal{C}(K) : a \leq^c b$
  $\Longrightarrow int(a) \dot{\models}^K int(b)$ (lemme 3.(3))
  $\Longrightarrow int(a)^K \dot{\models}^K int(b)^K$
  $\Longrightarrow int^K(a) \dot{\models}^K int^K(b)$ (définition de int^K).
- $\mu^K(int^K(a)) =^c \mu^K(int(a)^K) =^c \mu(int(a)) =^c a$ (lemme 2.(6)).
- $\mu(int(\mu(f))) =^c \mu(f)$ (lemme 2.(6))
  $\Longrightarrow int(\mu(f)) \dot{\equiv}^K f$ (lemme 3.(3))
  $\Longrightarrow int(\mu(f))^K \dot{=}^K f^K$
  $\Longrightarrow int^K(\mu^K(f^K)) \dot{=}^K f^K$ (définitions de μ^K et de int^K). ■

Les structures algébriques $\langle \mathcal{C}(K); \vee^c, \wedge^c \rangle$ et $\langle \mathcal{L}^K; \dot{\vee}^K, \dot{\wedge}^K \rangle$ (où $\dot{\vee}^K, \dot{\wedge}^K$ dénotent le supremum et l'infimum de $\langle \mathcal{L}^K; \dot{\models}^K \rangle$) sont donc homomorphiques[6] [DP90]. Alors, on a les relations suivantes pour tout $F,G \in \mathcal{L}^K$ et pour tout $a,b \in \mathcal{C}(K)$:

$$\begin{array}{ll} \mu^K(F) \wedge^c \mu^K(G) =^c \mu^K(F \dot{\wedge}^K G) & int^K(a) \dot{\wedge}^K int^K(b) \dot{=}^K int^K(a \wedge^c b) \\ \mu^K(F) \vee^c \mu^K(G) =^c \mu^K(F \dot{\vee}^K G) & int^K(a) \dot{\vee}^K int^K(b) \dot{=}^K int^K(a \vee^c b) \end{array}$$

En résumé, il y a trois façon de considérer le treillis de concepts, ce qui est un signe de richesse et de souplesse :

- les extensions $(\tau(\sigma(2^{\mathcal{O}})))$ ordonnées par l'inclusion ensembliste $(\subseteq)$,
- les intentions $(\sigma(\tau(\mathcal{L})))$ ordonnées par la déduction logique $(\dot{\models})$,
- les formules contextualisées $(\mathcal{L}^K)$ ordonnées par la déduction contextualisée $(\dot{\models}^K)$.

Enfin, on considère les correspondances entre la logique contextualisée $\mathcal{L}^K$ et la logique initiale $\mathcal{L}$. On a les relations suivantes entre les opérateurs des deux logiques.

Théorème 3 *Soient $f,g \in \mathcal{L}$.*

$$\begin{array}{l} f^K \dot{\wedge}^K g^K \dot{=}^K (f \dot{\wedge} g)^K \\ f^K \dot{\vee}^K g^K \dot{=}^K \sigma(\tau(f) \cup \tau(g))^K \end{array}$$

6. Deux structures algébriques sont dites homomorphiques si il existe une bijection entre elles qui préserve les opérations.

Démonstration :

- $\tau(f) \cap \tau(g) = \tau(f \dot{\wedge} g)$ (lemme 1.(iv'))
  $\Longrightarrow \mu(f) \wedge^c \mu(g) =^c \mu(f \dot{\wedge} g)$ (définitions de μ et $\wedge^c$)
  $\Longrightarrow \mu^K(f^K) \wedge^c \mu^K(g^K) =^c \mu^K((f \dot{\wedge} g)^K)$ (définition de μ^K)
  $\Longrightarrow f^K \dot{\wedge}^K g^K \dot{=}^K (f \dot{\wedge} g)^K$ (homomorphisme entre $\mathcal{C}(K)$ et $\mathcal{L}^K$).
- $\tau(\sigma(\tau(f) \cup \tau(g))) = \tau(\sigma(\tau(f) \cup \tau(g)))$
  $\Longrightarrow \mu(f) \vee^c \mu(g) =^c \mu(\sigma(\tau(f) \cup \tau(g)))$ (définitions de μ et $\vee^c$)
  $\Longrightarrow \mu^K(f^K) \vee^c \mu^K(g^K) =^c \mu^K(\sigma(\tau(f) \cup \tau(g))^K)$ (définition de μ^K)
  $\Longrightarrow f^K \dot{\vee}^K g^K \dot{=}^K \sigma(\tau(f) \cup \tau(g))^K$ (homomorphisme entre $\mathcal{C}(K)$ et $\mathcal{L}^K$). ■

On remarque donc que la fonction $(.)^K$ est un morphisme des formules vers les formules contextualisées pour la conjonction, mais pas pour la disjonction. De ce fait, le treillis de concepts n'est pas homomorphe à celui de la logique de départ ($\mathcal{L}$), et donc, certaines propriétés que pouvait avoir cette logique peuvent être perdues dans le treillis de concepts. C'est le cas par exemple de la propriété de distributivité dans le contexte K_{ex} (cf. page 6). En effet, la logique $\mathcal{P}$ est distributive (trivial d'après la sémantique de $\mathcal{P}$), tandis que dans le treillis de concepts on peut trouver le contre-exemple suivant (cf. figure 1) :

$$\begin{aligned} 2 \ \wedge^c (\ 1 \ \vee^c \ 3\ ) &=^c 2\ , \\ (\ 2 \ \wedge^c \ 1\ ) \vee^c (\ 2 \ \wedge^c \ 3\ ) &=^c 0\ . \end{aligned}$$

Pour que le treillis de concepts hérite de toutes les propriétés de $\mathcal{L}$ (modulo $=^c$), il faut que l'on ait $f^K \dot{\vee}^K g^K \dot{=}^K (f \dot{\vee} g)^K$. Pour celà, il suffit d'avoir $\tau(f \dot{\vee} g) = \tau(f) \cup \tau(g)$ (car alors $\sigma(\tau(f) \cup \tau(g)) \dot{\equiv}^K \sigma(\tau(f \dot{\vee} g)) \dot{\equiv}^K f \dot{\vee} g$). Plus concrètement, celà revient à poser la condition suivante sur les formules décrivant les objets :

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

Cette condition revient à considérer qu'il n'y a pas de forme de disjonction dans la description des objets. Si l'on revient sur le contexte K_{ex}, on constate que la description de l'objet z satisfait la proposition $a \vee b$ mais ne satisfait ni a, ni b, ce qui rend faux la condition 3. Ceci explique qu'on ait perdu la propriété de distributivité dans le treillis de concepts.

4 Discussion sur la logique $\mathcal{L}$

L'objet de cette section est de discuter de l'introduction de quelques éléments utiles dans $\mathcal{L}$. En effet, pour l'instant, on a seulement considéré que cette logique était pourvue des opérations de disjonction et de conjonction.

4.1 Top et bottom

On considère ici le cas où $\mathcal{L}$ est borné par $\dot{\top}$ (top) et $\dot{\bot}$ (bottom). On a donc la proposition suivante :

$$f \dot{\models} \dot{\top} \text{ et } \dot{\bot} \dot{\models} f, \text{ pour tout } f \in \mathcal{L}.$$

Ces formules $\dot{\top}$ et $\dot{\bot}$ correspondent naturellement aux formules habituelles *vrai* et *faux*.
On s'intéresse maintenant à l'existence et à la valeur des top et bottom du treillis de concepts.

Théorème 4 *Si $K = (\mathcal{O}, \mathcal{L}, i)$ est un contexte formel tel que $\forall o \in \mathcal{O} : i(o) \dot{\neq} \dot{\bot}$, alors $\mathcal{C}(K)$ est borné par $\top^c :=^c (\mathcal{O}, \dot{\bigvee}_{o \in \mathcal{O}} i(o))$ et $\bot^c :=^c (\emptyset, \dot{\bot})$.*

Démonstration :

- ($\top^c$ est un concept)
  $\sigma(ext(\top^c)) \dot{=} \sigma(\mathcal{O}) \dot{=} \dot{\bigvee}_{o \in \mathcal{O}} i(o) \dot{=} int(\top^c)$,
  $\tau(int(\top^c)) = \tau(\dot{\bigvee}_{o \in \mathcal{O}} i(o)) = \tau(\sigma(\mathcal{O})) \supseteq \mathcal{O}$ (lemme 1)
  or $\tau(\sigma(\mathcal{O})) \subseteq \mathcal{O}$ (par définition de τ)
  donc $\tau(int(\top^c)) = \mathcal{O} = ext(\top^c)$.
- ($\top^c$ est le top des concepts)
  Pour tout concept (O, f), $O \subseteq \mathcal{O} \Longrightarrow O \subseteq ext(\top^c) \Longrightarrow (O, f) \leq^c \top^c$.

- ($\bot^c$ est un concept)
  $\sigma(ext(\bot^c)) \doteq \sigma(\emptyset) \doteq \dot{\bigvee}\emptyset \doteq \dot{\bot}$ (cf. remarque 2.2 dans [DP90]),
  $\tau(int(\bot^c)) = \tau(\bot^c) = \{o \in \mathcal{O} | i(o) \dot{\models} \dot{\bot}\} = \emptyset$ (par hypothèse du théorème).
- ($\bot^c$ est le bottom des concepts)
  Pour tout concept (O,f), $\emptyset \subseteq O \Longrightarrow ext(\bot^c) \subseteq O \Longrightarrow \bot^c \leq^c (O,f)$. ■

Exemple. Dans la logique propositionnelle $\mathcal{P}$ (introduite p. 4), les formules $\dot{\top}$ et $\dot{\bot}$ sont respectivement instanciées par les propositions 1 et 0.

Maintenant, comme le treillis de concepts et la logique contextualisée sont isomorphes (cf. théorème 2), on peut borner cette dernière par un top $\dot{\top}^K$ et un bottom $\dot{\bot}^K$ (distincts a priori de $(\dot{\top})^K$ et $(\dot{\bot})^K$) et montrer les relations suivantes (d'après l'homomorphisme entre concepts et formules contextualisées, cf. section 3) :

$$\begin{array}{ll} \top^c =^c \mu^K(\dot{\top}^K) & \dot{\top}^K \doteq^K int^K(\top^c) \\ \bot^c =^c \mu^K(\dot{\bot}^K) & \dot{\bot}^K \doteq^K int^K(\bot^c) \end{array}$$

Enfin, on peut compléter le théorème 3 par les relations du théorème suivant (qui justifient les notations choisies pour le top et le bottom de la logique contextualisée).

Théorème 5

$$\begin{array}{l} \dot{\top}^K \doteq^K (\dot{\top})^K \\ \dot{\bot}^K \doteq^K (\dot{\bot})^K \end{array}$$

Démonstration :

- $\mathcal{O} = \tau(\dot{\top})$
  $\Longrightarrow ext(\top^c) = ext(\mu(\dot{\top}))$ (définitions de $\top^c$ et μ)
  $\Longrightarrow \top^c =^c \mu(\dot{\top})$
  $\Longrightarrow \mu^K(\dot{\top}^K) =^c \mu^K((\dot{\top})^K)$ (homomorphisme et définition de μ^K)
  $\Longrightarrow \dot{\top}^K \doteq^K (\dot{\top})^K$ (isomorphisme entre concepts et formules contextualisées, théorème 2).
- $\emptyset = \tau(\dot{\bot})$
  $\Longrightarrow ext(\bot^c) = ext(\mu(\dot{\bot}))$ (définitions de $\bot^c$ et μ)
  $\Longrightarrow \bot^c =^c \mu(\dot{\bot})$
  $\Longrightarrow \mu^K(\dot{\bot}^K) =^c \mu^K((\dot{\bot})^K)$ (homomorphisme et définition de μ^K)
  $\Longrightarrow \dot{\bot}^K \doteq^K (\dot{\bot})^K$ (isomorphisme entre concepts et formules contextualisées, théorème 2). ■

4.2 La complémentation relative

Le domaine d'application envisagé est celui des systèmes d'information dans lesquels le treillis de concept jouerait le rôle de structure d'organisation et de navigation. Les éléments de $\mathcal{L}$ servent à la fois à décrire les propriétés des objets (par la fonction i du contexte) et à formuler les requêtes.

Deux fonctionnalités de ces systèmes d'informations rendent nécessaire la possibilité de "retrancher" une formule à une autre. La première est le déplacement d'un objet du concept étiqueté par *origine* (dont l'extension contient donc l'objet en question) au concept étiqueté par *destination*. Dans ce cas, il faut modifier la description de l'objet en lui "retranchant" la formule *origine* puis en lui "ajoutant" la formule *destination* (au cours de cette opération, il faut noter que le treillis de concept peut changer car la composante i du contexte est modifiée). La deuxième est la navigation qui doit proposer pour chaque requête un ensemble de requêtes plus précise : les *sous-requêtes* (analogues des sous-répertoires dans les systèmes de fichiers). En fait, ce que l'on voudrait donner à l'utilisateur, ce ne sont pas les sous-requêtes elles-mêmes mais des formules plus simples, obtenues par "retranchement" de la requête initiale aux sous-requêtes.

Pour l'"ajout" d'une formule à une autre, l'opération de conjonction $\dot{\wedge}$ est satisfaisante (en effet, on a $f \dot{\wedge} g \dot{\models} f$ et $f \dot{\wedge} g \dot{\models} g$). Pour le "retranchement" d'une formule à une autre, on propose d'introduire la notion de *complément relatif*. Pour celà, il faut ajouter aux conditions posées sur $\mathcal{L}$ pour l'ACL (cf. section 2), celle de la *complémentation relative*

$$\forall f,g \in \mathcal{L} : \exists! x : (f \dot{\wedge} x \dot{\models} g) \wedge \neg \exists x' : (x' \dot{\neq} x) \wedge (x \dot{\models} x') \wedge (f \dot{\wedge} x' \dot{\models} g). \quad (4)$$

Cette condition dit que pour toutes formules f et g, il existe une formule x la plus générale telle que la conjonction de f et de x est plus précise que g. La formule x correspond au complément de g relativement à f. Nous définissons maintenant une nouvelle opération binaire $\dot{\Rightarrow}$ dans $\mathcal{L}$ qui rend le complément relatif de deux formules.

Définition 9 *Soient $f,g \in \mathcal{L}$, on caractérise le* complément relatif *$f \dot{\Rightarrow} g$ par*

$$\forall f,g \in \mathcal{L} : (f \dot{\wedge} (f \dot{\Rightarrow} g) \dot{\models} g) \wedge \neg \exists x' : (x' \dot{\neq} (f \dot{\Rightarrow} g)) \wedge ((f \dot{\Rightarrow} g) \dot{\models} x') \wedge (f \dot{\wedge} x' \dot{\models} g).$$

$f \dot{\Rightarrow} g$ a bien une valeur, et une valeur unique, d'après la condition 4.

Lemme 4 *Soient $f,f',g \in \mathcal{L}$,*

1. $g \dot{\models} f \dot{\Rightarrow} g$ *(axiome K de Hilbert)*
2. $f \dot{\wedge} (f \dot{\Rightarrow} g) \dot{=} g \dot{\wedge} (g \dot{\Rightarrow} f) \dot{=} f \dot{\wedge} g$ *(forme de* modus ponens*)*
3. $g \dot{\models} f \Longrightarrow f \dot{\wedge} (f \dot{\Rightarrow} g) \dot{=} g$ *(cas où le complément relatif est* exact*)*
4. $f \dot{\models} g \Longleftrightarrow (f \dot{\Rightarrow} g) \dot{=} \dot{\top}$
5. $f' \dot{\models} f \Longrightarrow f \dot{\Rightarrow} g \dot{\models} f' \dot{\Rightarrow} g$

Démonstration :

1. g est une solution de $f \dot{\wedge} x \dot{\models} g$
2. $g \dot{\wedge} (g \dot{\Rightarrow} f) \dot{\models} f$ (déf. de $\dot{\Rightarrow}$) et $g \dot{\wedge} (g \dot{\Rightarrow} f) \dot{\models} g$ donc $g \dot{\wedge} (g \dot{\Rightarrow} f) \dot{\models} f \dot{\wedge} g$
   De plus, $f \dot{\models} g \dot{\Rightarrow} f$ (axiome K), et donc $f \dot{\wedge} g \dot{\models} g \dot{\wedge} (g \dot{\Rightarrow} f)$
   Même chose pour $f \dot{\wedge} (f \dot{\Rightarrow} g)$.
3. si $g \dot{\models} f$ on a $f \dot{\wedge} g \dot{=} g$
   donc d'après le lemme 2, $f \dot{\wedge} (f \dot{\Rightarrow} g) \dot{=} g$.
4. $$\begin{aligned} f \dot{\Rightarrow} g \dot{=} \dot{\top} &\Longleftrightarrow (f \dot{\wedge} \dot{\top} \dot{\models} g) \wedge \neg \exists x' > \dot{\top} : (f \dot{\wedge} x' \dot{\models} g) \text{ (par définition)} \\ &\Longleftrightarrow f \dot{\models} g \text{ (par manipulations algébriques)} \end{aligned}$$
5. $f' \dot{\wedge} (f \dot{\Rightarrow} g) \dot{\models} f \dot{\wedge} (f \dot{\Rightarrow} g) \dot{\models} g$ (car $f' \dot{\models} f$)
   donc $f \dot{\Rightarrow} g$ est solution de $f' \dot{\wedge} x \dot{\models} g$
   et donc $f \dot{\Rightarrow} g \dot{\models} f' \dot{\Rightarrow} g$. ■

Le lemme 4.3 est particulièrement intéressant, car il montre que dans le cas où deux formules sont comparables ($g \dot{\models} f$) on peut exprimer exactement le "retranchement" de la plus grande (f) à la plus petite (g) par $f \dot{\Rightarrow} g$. Or, c'est précisément dans ce cas là qu'on se trouve lorsqu'on parle de "retranchement" pour déplacer des objets ou pour naviguer dans un système d'information.

A partir de ce complément relatif, on peut aussi définir un *complément* $\dot{\neg}$ par

$$\forall f \in \mathcal{L} : (\dot{\neg} f) \dot{=} (f \dot{\Rightarrow} \dot{\bot}).$$

En partant de la caractérisation de $\dot{\Rightarrow}$, on aboutit à une caractérisation assez naturelle de $\dot{\neg}$:

$$\forall f \in \mathcal{L} : (f \dot{\wedge} (\dot{\neg} f) \dot{=} \dot{\bot}) \wedge \neg \exists x' : (x' \dot{\neq} (\dot{\neg} f)) \wedge ((\dot{\neg} f) \dot{\models} x') \wedge (f \dot{\wedge} x' \dot{=} \dot{\bot}).$$

Exemple. Dans la logique propositionnelle $\mathcal{P}$ (introduite p. 4), la complémentation relative $\Rightarrow$ existe et est caractérisée par

$$\forall P,Q \in \mathcal{P} : P \Rightarrow Q \equiv \neg P \vee Q.$$

On a aussi pour toute proposition P la propriété $P \Rightarrow 0 \equiv \neg P \vee 0 \equiv \neg P$, ce qui établit une cohérence entre complément et complément relatif.

4.3 Terminologie

Dans l'analyse de concepts, on peut distinguer plusieurs sources de la connaissance contenue dans un contexte formel ou un treillis de concepts (rappelons que contexte et treillis de concepts contiennent exactement la même quantité d'information car on peut passer de l'un à l'autre). La première source est la logique $\mathcal{L}$ qui apporte une connaissance de base sur le domaine choisi. C'est une forme de connaissance absolue, figée, indépendante de la composante extensionnelle du contexte. Cette logique introduit an fait un mécanisme de raisonnement (automatique) sur les intentions.

La deuxième source est la composante extensionnelle du contexte, c-à-d., les objets et les intentions qui leur sont associées. Dans le cadre des systèmes d'information, c'est une forme de connaissance relative, dynamique (création, modification et destruction d'objets). Celle-ci est révélée par le treillis de concepts et la logique contextualisée qui sont d'ailleurs isomorphes (cf. théorème 2).

Enfin, dans les systèmes d'information, on a souvent besoin d'une troisième source correspondant à la connaissance que peut apporter le concepteur ou l'utilisateur du système. Des exemples sont la hiérarchie de types pour les graphes conceptuels [CM92], la TBox (Terminological Box) pour les logiques de description [Nap97], les échelles conceptuelles pour l'analyse de concepts [Pre97] ou les règles d'inférence de Datalog [CGT90]. Cette connaissance, que l'on choisit d'appeler *terminologie* est intermédiaire entre la logique et le contexte puisqu'elle intervient au niveau des intentions, et complète donc la logique de base, mais est dynamique car elle peut être mise à jour et étendue.

De ce point de vue, on peut considérer la terminologie comme un paramètre de la logique. Au lieu d'avoir une simple logique $\mathcal{L}$, on se donne une famille de logiques $(\langle \mathcal{L}_T ; \dot{\models}_T, \dot{\vee}_T, \dot{\wedge}_T \rangle)_{T \in \mathcal{T}}$ paramétrée par une terminologie T prise dans le domaine $\mathcal{T}$. Ce domaine peut être partiellement ordonné par la relation d'ordre $\sqsubseteq$, défini pour tout $T_1, T_2 \in \mathcal{T}$ par :

$$T_1 \sqsubseteq T_2 \iff \langle \mathcal{L}_{T_1} ; \dot{\models}_{T_1} \rangle \subseteq \langle \mathcal{L}_{T_2} ; \dot{\models}_{T_2} \rangle.$$

Intuitivement, cette relation exprime que la terminologie T_2 contient toute la connaissance exprimée dans T_1. On peut ensuite discuter des propriétés du domaine partiellement ordonné des terminologies $\langle \mathcal{T} ; \sqsubseteq \rangle$. Si il possède un bottom $\perp$, alors $\mathcal{L}_\perp$ joue le rôle de logique de base, celle qui est contenue dans toutes les autres et sert de dénominateur commun à la famille $(\mathcal{L}_T)_{T \in \mathcal{T}}$. Si c'est un treillis, on peut associer aux supremum $\sqcup$ et infimum $\sqcap$ des opérations de disjonction et de conjonction dans $\mathcal{T}$. La disjonction de terminologie permettrait en particulier de fusionner deux systèmes d'information ayant chacun sa propre terminologie.

Un exemple de logique avec terminologie est donné à la section 5.2 comparant la notion d'ajustement logique de Prediger [Pre97] à l'ACL.

4.4 Produit de logiques

Dans un système d'information généraliste (ex. système de fichiers), les objets peuvent être de natures très diverses, et on peut alors ressentir le besoin d'associer à ces objets des propriétés de différentes sortes (ex. attributs systèmes tels que la taille ou les droits d'accès, types pour les fonctions d'une librairie, formalisme de représentation de connaissance pour des propriétés de plus haut niveau). Or, il peut être difficile de trouver une logique qui puisse combiner ces différents sortes de propriétés sans qu'elle ne devienne trop complexe. Dans ce cas là, on peut se donner une famille de logiques indépendantes (c-à-d., dont les formules sont incomparables), et associer à chaque objet une formule de chacune de ces logiques. La logique globale obtenue est un *produit de logiques*.

Définition 10 *Soit* $(\langle \mathcal{L}_k ; \dot{\models}_k, \dot{\vee}_k, \dot{\wedge}_k \rangle)_{k \in K}$ *une famille finie de logiques.*
On définit le produit des logiques $(\mathcal{L}_k)_{k \in K}$ *par* $\langle \mathcal{L} ; \dot{\models}, \dot{\vee}, \dot{\wedge} \rangle$ *avec* $\mathcal{L} = \Pi_{k \in K} \mathcal{L}_k$ *et telle que pour tout* $f, g \in \mathcal{L}$ *:*

- $f \dot{\models} g \iff \forall k \in K : f[k] \dot{\models}_k g[k]$

- $\forall k \in K : (f \dot{\vee} g)[k] \dot{=}_k f[k] \dot{\vee}_k g[k]$
- $\forall k \in K : (f \dot{\wedge} g)[k] \dot{=}_k f[k] \dot{\wedge}_k g[k]$.

Théorème 6 *Si tous les éléments de la famille* $(\mathcal{L}_k)_{k \in K}$ *sont des logiques adaptées à l'ACL, alors le produit de ces logiques* $\Pi_{k \in K} \mathcal{L}_k$ *est aussi une logique adaptée à l'ACL.*

Démonstration : Il suffit de montrer que $\langle \mathcal{L}; \dot{\models}, \dot{\vee}, \dot{\wedge} \rangle$ satisfait les conditions de l'ACL, ce qui se fait très facilement. On se contente donc de donner la démonstration pour la condition $f \dot{\models} g \leftrightarrow f \dot{\wedge} g \dot{=} f$, les autres démonstrations étant très semblables.

$\forall k \in K : f[k] \dot{\models}_k g[k] \leftrightarrow f[k] \dot{\wedge}_k g[k] \dot{=}_k f[k]$ (car les $\mathcal{L}_k$ satisfont les conditions de l'ACL)
$\Longrightarrow \forall k \in K : f[k] \dot{\models}_k g[k] \Leftrightarrow \forall k \in K : (f \dot{\wedge} g)[k] \dot{=}_k f[k]$ (par définition de $\dot{\wedge}$)
$\Longrightarrow f \dot{\models} g \Leftrightarrow f \dot{\wedge} g \dot{=} f$ (par définition de $\dot{\models}$). ■

5 Discussion sur des travaux existants

Le besoin d'utiliser comme contexte des données plus riches que la relation objet-attribut a déjà été ressenti, et deux voies ont été empruntées pour étendre l'AC classique. La première consiste à remplacer les attributs par des entités plus complexes, tels que des termes du premier ordre [CM98] (cf. sous-section 5.1). La deuxième consiste à ramener à un contexte formel classique, un contexte plus complexe tel qu'une relation objet-attribut-valeur [Pre97], issue d'une base de donnée relationnelle par exemple (cf. sous-section 5.2). Une troisième voie n'enrichit pas le contexte, mais dérive d'un contexte une logique plus riche que celle qui est sous-jacente à l'ACF [GW99b] (cf. sous-section 5.3). Pour chacune de ces voies, nous montrons dans la suite comment se situe l'ACL et la contribution qu'elle apporte.

5.1 Extension de l'ACF aux termes du 1er ordre

L. Chaudron et N. Maille proposent une extension de l'ACF à la logique du 1er ordre [CM98]. En fait, ils remplacent les attributs par des termes du 1er ordre avec variables existentiellement quantifiées (termes Prolog). Les informations associées aux objets sont donc des ensembles finis de tels termes appelés *cubes logiques*. La logique sous-jacente est celles des conjonctions de litéraux, soit un fragment de la logique du 1er ordre proprement dite.

Le domaine des cubes logiques est noté $\mathcal{C}$ sur lequel on définit une opération de supremum $\cap_c$ pour capturer les traits communs de deux cubes, et une opération d'infimum $\cup_c$ pour réunir les caractéristiques de deux cubes. Avant de donner leur expression littérale, on doit introduire la notion de *réductibilité* des cubes qui permet de supprimer certaines redondances et de normaliser les cubes logiques. Formellement, on dit qu'un cube C est *réductible* s'il existe une substitution θ telle que $C\theta \subsetneq C$. On démontre que tout cube C admet une réduction irréductible unique que l'on note $reduc(C)$. L'ensemble des cubes irréductibles est noté $\mathcal{C}^r$.

Définition 11 *Soient $c_1, c_2 \in \mathcal{C}^r$, les opérations d'infimum et de supremum $\cap_c$ et $\cup_c$ sont définies par :*

$$c_1 \cap_c c_2 := reduc(c_1 \cup c_2)$$
$$c_1 \cup_c c_2 := reduc(anti_unif(c_1, c_2)),$$

où $anti_unif$ dénote l'anti-unification de deux cubes c_1 et c_2, c-à-d. l'union des anti-unifications[7] *des éléments de c_1 avec ceux de c_2.*

On démontre ensuite que $\langle \mathcal{C}^r; \cap_c, \cup_c \rangle$ est un treillis dont l'ordre induit est noté $\geq_c$ et est équivalent à la subsomption sur les cubes. Finalement, il apparaît que l'ACF étendue aux cubes logiques est une instanciation de l'ACL. En effet, pour retrouver l'ACF étendue aux termes du 1er ordre, il suffit d'instancier les paramètres de la logique $\mathcal{L}$ de l'ACL suivant les correspondances suivantes :

général	$\mathcal{L}$	$\dot{\vee}$	$\dot{\wedge}$	$\dot{\models}$
instancié	$\mathcal{C}^r$	$\cap_c$	$\cup_c$	$\geq_c$

La logique des cubes est bien utilisable dans le cadre de l'ACL car $\langle \mathcal{C}^r; \geq_c \rangle$ est bien un treillis de supremum $\cap^c$ et d'infimum $\cup^c$. Enfin, si on instancie les expressions de σ et de τ de l'ACL, on retrouve bien celles données par [CM98].

Dans l'article de Chaudron et Maille, l'ACF a été étendue pour une logique particulière (celle des cubes logiques). Cette extension est motivée par un besoin d'expressivité pour des applications en analyse de données. Aussi, on peut facilement imaginer que pour d'autres applications, dans le domaine de l'analyse de données

7. L'anti-unification [Plo70] de deux termes t_1 et t_2 est définie comme le terme le plus précis qui admette à la fois t_1 et t_2 comme instances.

ou non, on ressente le besoin d'étendre l'ACF vers d'autres logiques. L'ennui est que pour chaque logique particulière, il faut re-définir l'ACF et re-démontrer ses résultats fondamentaux. L'intérêt de l'analyse de concept logique (ACL) est donc de fournir un cadre général qui donne les conditions que doivent satisfaire la logique employée et qui effectue les démonstrations une fois pour toutes. L'utilisation de nouvelles logiques dans le cadre de l'AC devient donc peu coûteux et pourrait encourager des applications plus variées.

5.2 Ajustement logique dans l'ACF

S. Prediger présente deux méthodes pour obtenir un contexte formel classique à partir de contextes multi-valués [Pre97]. Ces méthodes sont motivées par le fait que les contextes multi-valués (en fait, des relations objet-attribut-valeur) sont fréquemment employés pour représenter des problèmes du monde réel (ex. dans les bases de données relationnelles). L'objectif est d'appliquer l'ACF à ces contextes pour en extraire des hiérarchies conceptuelles. Pour celà, il faut dériver un contexte mono-valué à partir des contextes multi-valués. Deux méthodes sont proposées : les *échelles conceptuelles* et l'*ajustement logique*.

5.2.1 Echelles conceptuelles et ajustement logique

Les échelles conceptuelles (*conceptual scales*) sont des contextes formels dont les objets sont des valeurs et les attributs sont des attributs d'échelle. Ces contextes induisent des treillis de concepts qui définissent une hiérarchie sur les attributs d'échelle. A chaque attribut du contexte multi-valué, on associe une telle échelle conceptuelle qui permet en quelque sorte de structurer le domaine de valeurs de cet attribut. Par exemple, pour l'attribut *importance* dont les valeurs sont comprises entre 1 et 5, on peut construire une échelle conceptuelle dont les attributs d'échelle sont "≤ 1" à "≤ 5". Dans ce cas, la valeur 3 aura les attributs d'échelle "≤ 3", "≤ 4" et "≤ 5". Le contexte mono-valué est alors obtenu à partir du contexte multi-valué, en fonction des échelles conceptuelles de chaque attribut. L'ensemble d'objet est conservé, l'ensemble d'attributs est l'union disjointe des attributs d'échelle des échelles conceptuelles et la relation revient à une jointure entre celle du contexte multi-valué et celles des échelles conceptuelles.

Le principe de l'ajustement logique (*logical scaling*) est d'utiliser un langage formel (ex. SQL) pour générer des prédicats unaires à partir des attributs et des valeurs des attributs d'un contexte multi-valué. Une terminologie est un ensemble fini (et de taille modérée) de tels prédicats auxquels on associe un nom. Le contexte mono-valué est dérivé du contexte multi-valué à partir d'une terminologie. L'ensemble d'objets est conservé, l'ensemble d'attributs est l'ensemble des noms des prédicats de la terminologie et un objet est en relation avec un nom si les valeurs de ses attributs sont tels que le prédicat associé au nom est satisfait. Par exemple, si la terminologie contient la définition "$assez_important := importance \leq 4$" ($assez_important$ est le nom du prédicat et *importance* est un attribut) et que la valeur de l'attribut *importance* d'un objet est 3, alors cet objet aura l'attribut $assez_important$ dans le contexte dérivé.

Dans les deux cas, il peut y avoir perte d'information. En effet, dans le premier cas les valeurs sont remplacées par des attributs d'échelle, ce qui correspond en quelque sorte à une abstraction des données, et dans le second cas on s'intéresse seulement à la satisfaction ou non par les objets de quelques prédicats. Pour des applications d'analyse de données, celà peut être utile puisqu'il s'agit de discerner des propriétés générales parmi des données nombreuses et détaillées (ex. implication et dépendance entre attributs [GW99a]).

En revanche, pour les applications que nous envisageons, à savoir les systèmes d'information, il est indispensable de conserver toutes les informations du contexte initial. De plus, les systèmes d'information amènent d'autres contraintes qui ne permettent pas d'employer les méthodes présentées ci-dessus. En particulier, les mises-à-jour et les consultations sont entrelacées, ce qui rend l'utilisation d'une terminologie (au sens de l'ajustement logique) impraticable car on ne connaît pas les prédicats/requêtes à l'avance. De plus, comme nous envisageons de nous appuyer sur le treillis de concepts pour réaliser l'interrogation et la navigation du système d'information, la réduction du contexte multi-valué à un contexte mono-valué revient à réduire le langage de requêtes d'un langage propositionnel multi-valué (de type SQL) à un langage de conjonctions de propositions mono-valuées (de type Altavista), qui est moins expressif.

Finalement, nous estimons que si beaucoup de problèmes du monde réel sont représentés par des relations objet-attribut-valeur, c'est que celà correspond à un besoin d'expressivité et qu'il est donc souhaitable de conserver cette expressivité lorsqu'on passe au treillis de concepts, via l'analyse de concepts. Les éléments d'expressivité qui sont évoqués dans l'article de S. Prediger sont :

- les attributs valués,
- les relations sur les domaines de valeurs (ex. comparaison de valeurs numériques),
- les opérateurs logiques habituels : $\wedge$ (conjonction), $\vee$ (disjonction) et $\neg$ (négation),

– la terminologie, c-à-d. la possibilité de donner un nom à certains prédicats.

Nous proposons donc de définir une logique réunissant tous ces éléments d'expressivité et de montrer que la logique ainsi construite est utilisable dans le cadre défini par l'ACL.

5.2.2 Une logique propositionnelle multi-valuée pour l'ACL

On se donne un ensemble d'attributs $\mathcal{A}$, un ensemble de valeurs $\mathcal{V}$, un ensemble de relations $\mathcal{R}$ sur $\mathcal{V}$ et une terminologie $T = (N,\nu)$ où N est un ensemble de noms et ν associe à chacun de ces noms une formule de la logique $\mathcal{L}_T$, dont on donne maintenant la syntaxe (T est en indice car on considère que c'est un élément variable de la logique, cf. section 4.3) :

$$\begin{array}{lcll} P,Q & ::= & a = v & a \in \mathcal{A},\ v \in \mathcal{V} \\ & | & a\ R\ v & a \in \mathcal{A},\ v \in \mathcal{V},\ R \in \mathcal{R} \\ & | & n & n \in N \\ & | & P \wedge Q & \\ & | & P \vee Q & \\ & | & \neg P & \end{array}$$

Pour donner la sémantique de la logique $\mathcal{L}_T$, on commence par définir une interprétation comme une fonction partielle de $\mathcal{A}$ vers $\mathcal{V}$ (on note $\mathcal{M}$ le domaine des interprétations). La notation $\mu \models P$ signifie que l'interprétation μ satisfait, ou est modèle de la formule P. La propriété d'être modèle pour une interprétation arbitraire μ est définie de la façon suivante :

$$\begin{array}{lll} \mu \models a = v & \text{ssi} & \mu(a) = v \\ \mu \models a\ R\ v & \text{ssi} & (\mu(a),v) \in R \\ \mu \models n & \text{ssi} & \mu \models \nu(n) \\ \mu \models P \wedge Q & \text{ssi} & \mu \models P \text{ et } \mu \models Q \\ \mu \models P \vee Q & \text{ssi} & \mu \models P \text{ ou } \mu \models Q \\ \mu \models \neg P & \text{ssi} & \mu \not\models P \end{array}$$

Ensuite, on définit la relation de déduction sur $\mathcal{L}_T$ par $P \models Q \iff \forall \mu \in \mathcal{M} : \mu \models P \Rightarrow \mu \models Q$, et on note $\equiv$ la relation d'équivalence associée au pré-ordre $\models$. On peut lors facilement montrer que la logique $\langle \mathcal{L}_T/_\equiv ; \models ,\vee,\wedge \rangle$ satisfait les conditions de l'ACL.

Il ne reste plus qu'à montrer comment construire un contexte formel à partir d'une relation objet-attribut-valeur. Une telle relation associe à chaque objet d'un ensemble $\mathcal{O}$ et à chaque attribut de l'ensemble $\mathcal{A}$ au plus une valeur. Celà revient à associer à chaque objet o une interprétation, que l'on note μ_o. Le contexte formel est alors défini par le triplet $(\mathcal{O},\mathcal{L}_T/_\equiv,i)$ où la fonction i est définie pour tout objet $o \in \mathcal{O}$ par

$$i(o) := \bigwedge_{a \in dom(\mu_o)} (a = \mu_o(a)).$$

Le treillis de concepts obtenu par ACL à partir d'un tel contexte est certainement plus complexe et moins lisible que ceux obtenus par les méthodes de S. Prediger, car la logique utilisée étant plus expressive, la discrimination entre les concepts est plus fine et ceux-ci sont en conséquence plus nombreux. Mais si l'on considère comme applications les systèmes d'information, celà n'est nullement gênant dans la mesure où le treillis de concepts n'intervient que comme structure interne d'organisation et de navigation. L'utilisateur du système n'a qu'une vue abstraite et indirecte de ce treillis (cf. section 6 pour plus de détails).

Nous terminerons par une remarque concernant l'ajustement logique présenté précédemment, qui a pour intérêt d'étudier la hiérarchie conceptuelle d'un ensemble de prédicats. En fait, cette fonctionnalité peut aussi être obtenue dans l'ACL grâce à la logique contextualisée (cf. section 3). En effet, il suffit de déterminer les relations de déductions contextualisées entre ces prédicats[8], puis de tracer le diagramme de Hasse de la relation d'ordre obtenue pour obtenir la hiérarchie conceptuelle de ces prédicats (au sens de [Pre97]).

5.3 Logique contextuelle à attributs

Dans le cadre de l'ACF, B. Ganter et R. Wille définissent une Logique Contextuelle à Attributs (LCA) qui a la propriété de dépendre d'un contexte formel [GW99b]. Cette logique leur permet d'extraire des connaissances du contexte en cherchant certaines formules valides : les séquents (clauses disjonctives complètes). Il s'agit d'un enrichissements des implications entre attributs (section 2.3 dans [GW99a]).

8. Rappelons que $P \dot{\models}^K Q$ ssi $\tau(P) \subseteq \tau(Q)$, où $\dot{\models}^K$ est la relation de déduction contextualisée.

A la section 3, nous avons aussi introduit une logique dépendante du contexte : la logique contextualisée. La différence essentielle est que la logique contextualisée est dérivée de la logique $\mathcal{L}$, qui est indépendante du contexte, et elle est donc une conséquence de l'ACL ; tandis que la LCA est un développement *a posteriori* de l'ACF (en particulier, elle n'affecte ni le contexte, ni le treillis de concepts). On va montrer que ces deux approches ne sont pas indépendantes : la LCA est la logique contextualisée obtenue par instanciation de l'ACL à la logique propositionnelle $\mathcal{P}$ (introduite dans l'exemple, p. 4) et par restriction des descriptions des objets aux clauses conjonctives complètes.

Dans la logique propositionnelle $\mathcal{P}$, les propositions atomiques correspondent aux attributs de Ganter et Wille, et les propositions correspondent aux attributs composés. Comme $\mathcal{P}$ satisfait les conditions de l'ACL, on peut en dériver pour tout contexte K une logique contextualisée $\langle \mathcal{P}; \wedge, \vee, \models^K \rangle$, où $P \models^K Q \iff \tau(P) \subseteq \tau(Q)$ (appelons *extension* de P l'expression $\tau(P)$).

Ganter et Wille s'intéressent à certaines propositions (les séquents) qui sont "extensionnellement complètes" (*all-extensional*), c-à-d. dont l'extension est l'ensemble $\mathcal{O}$ des objets du contexte. Dans notre cas, la fonction τ est définie à partir de la logique $\langle \mathcal{P}; \models \rangle$ qui n'intervient pas dans la démarche de Ganter et Wille. Ceux-ci donnent une définition de l'extension des propositions par induction structurelle sur les propositions. Pour comparer ces définitions, nous cherchons à donner une caractérisation inductive de τ qui soit bien sûr compatible avec notre définition. Il apparaît qu'en toute généralité, une telle caractérisation n'existe pas (par exemple, on ne peut pas exprimer $\tau(P \vee Q)$ en fonction de $\tau(P)$ et de $\tau(Q)$, sans utiliser la relation de déduction $\models$). Cependant, si on restreint les descriptions des objets à des clauses conjonctives complètes, c-à-d. à des formules de la forme $i(o) = (\bigwedge_{a \in I(o)} a) \wedge (\bigwedge_{a' \in \mathcal{A} \setminus I(o)} \neg a')$ (où $I \in \mathcal{O} \to 2^{\mathcal{A}}$), ce qui revient à considérer que le contexte est équivalent à ceux de l'ACF, alors on obtient une caractérisation inductive de τ identique à celle de Ganter et Wille, à savoir :

$$\begin{array}{lcl} \tau(a) & = & \{o \in \mathcal{O} | a \in I(o)\} \\ \tau(P \wedge Q) & = & \tau(P) \cap \tau(Q) \\ \tau(P \vee Q) & = & \tau(P) \cup \tau(Q) \\ \tau(\neg P) & = & \mathcal{O} \setminus \tau(P). \end{array}$$

En résumé, la logique contextualisée dérivée de la logique propositionnelle $\mathcal{P}$ recouvre la LCA sur les contextes de type ACF, et l'étend pour les contextes plus généraux de l'ACL (les descriptions des objets sont des propositions quelconques).

Dans le cas où le contexte considéré est le *contexte de test* $T = (2^{\mathcal{A}}, \mathcal{A}, \ni)$, on observe que pour toute proposition P on a l'égalité $\tau(P) = M(P)$ car dans ce cas, l'ensemble des objets $\mathcal{O}$ est l'ensemble des interprétations possibles et pour tout $A \in 2^{\mathcal{A}}$ on a $I(A) = A$. Cela signifie que la logique propositionnelle contextualisée par le contexte de test T est la logique propositionnelle elle-même.

6 Application aux systèmes d'information logiques

Jusqu'à présent, l'ACL a été présentée en toute généralité, indépendamment de toute application. Cependant, bien que l'ACL soit certainement applicable à des domaines variés, elle a été élaborée dans l'objectif d'être appliquée dans le domaine des systèmes d'information. Plus précisément, nous adoptons une approche formelle des systèmes d'information. En particulier, nous voulons décrire les objets du systèmes par des expressions d'une logique formelle (presque) arbitraire. C'est-à-dire que nous ne voulons pas définir un système d'information particulier, mais plutôt un schéma général qui puisse être instancié en fonction des applications, par une logique appropriée. C'est pourquoi nous parlons de systèmes d'information logiques (SIL). Dans la sous-section 6.1, nous montrons l'intérêt qu'apporte l'AC par rapport à d'autres méthodes pour l'organisation et la récupération d'informations : combiner interrogation et navigation. Puis nous présentons à la sous-section 6.2 un premier prototype de SIL qui soutient la faisabilité de notre projet et qui a permis d'effectuer quelques expérimentations.

6.1 Interrogation et navigation

La récupération d'informations englobe la représentation, le stockage, l'organisation et l'accès aux informations. Deux méthodes de récupération d'informations sont largement acceptées et appliquées. La première méthode est la *classification hiérarchique* que l'on trouve fréquemment dans les outils informatiques : systèmes de fichiers, carnets d'adresses internet ou archivage des mails. Dans ce modèle, les recherches s'effectuent par *navigation* dans une structure construite et maintenue manuellement. Cette structure est le plus souvent un arbre (ex. système de fichiers), mais peut aussi être un graphe (ex. liens hypermédias entre les pages Web). Un fait notable en terme de d'accès, dans le cas d'une structure arborescente, est qu'il existe un unique chemin

d'accès à chaque objet, ce qui implique de respecter un ordre bien précis dans la spécification des critères de recherche. Dans le cas des graphes, il peut exister plusieurs chemins d'accès à un objet, mais l'ordre des critères de recherche dans chaque chemin est encore figé. La deuxième méthode est l'*interrogation booléenne* que l'on trouve souvent dans les serveurs d'informations tels que les moteurs de recherche sur le Web (ex. AltaVista). Dans ce modèle, les recherches s'effectuent en posant des *requêtes*, en général exprimées dans une forme de logique propositionnelle. Une difficulté reconnue de ce modèle est la nécessité de bien connaître la terminologie du système d'information et d'avoir une idée assez précise de ce qu'on cherche. Or, il est plus facile de reconnaître un objet que de le décrire, et la nécessité d'exprimer une requête peut être un frein pour les utilisateurs occasionnels.

Alors, quel mode de recherche faut-il privilégier : la navigation ou l'interrogation? En fait, cela dépend des situations et on peut même avoir besoin des deux conjointement. Par exemple, on peut souhaiter commencer la recherche par une requête et ensuite la raffiner par navigation. Des systèmes hybrides combinant un système hiérarchique et un système booléen ont été proposés dans le domaine des systèmes de fichiers (SF) :

- SFS (Semantic File System, [GJSO91]) étend le modèle hiérarchique des SFs classiques par des répertoires virtuels correspondant à des requêtes. Ces requêtes portent sur les propriétés des fichiers qui sont générées automatiquement par des transducteurs (*transducers*), et sont exprimées dans une logique à attributs valués. Il y a donc deux modes d'organisation et de stockage qui co-existent : la hiérarchie classique qui donne un nom aux fichiers et des répertoires virtuels qui permettent des recherches associatives sur les propriétés intrinsèques des fichiers. Malheureusement, ces deux modes ne peuvent pas être utilisés conjointement en toute généralité ; le mode d'accès associatif est une surcouche du mode d'accès hiérarchique qui n'est disponible que pour les accès en lecture. En particulier, les répertoires virtuels ne sont pas des répertoires à part entière (ex. `pwd` échoue dans un tel répertoire, et `mv` n'est même pas défini car les propriétés des fichiers ne sont pas données par l'utilisateur mais par les transducteurs).
- HAC (Hierarchy And Content, [GM99]) reprend l'idée de requêtes pour créer des répertoires basés sur le contenu des fichiers, mais ces répertoires sont ici intégrés dans la hiérarchie. Cela permet de combiner la hiérarchie et le contenu dans les recherches. L'objectif est de conserver toutes les fonctionnalités des SFs existants. Ainsi, les requêtes ne sont qu'une aide pour organiser les fichiers. L'utilisateur a toujours la possibilité de déplacer un fichier dans un répertoire, même s'il ne satisfait pas la requête associée, ce qui pose un problème de cohérence.

Ces systèmes hybrides présentent le désavantage de manquer de cohérence. En effet, ils ont deux modes de recherche incomplètement corrélés, ce qui rend difficile la commutation d'un mode à l'autre et la combinaison des deux dans une même recherche.

A côté de cela, le treillis de concept est un espace de recherche supportant les deux modes de recherche. Dans le mode navigationnel, les concepts sont considérés comme des classes ou répertoires (extension du concept) et les liens sont concrétisés par les relations de généralisation/spécialisation. Dans le mode interrogatif, les concepts sont considérés comme des requêtes (intention du concept) et la sélection d'un concept se fait par spécification directe d'une propriété. Comme les concepts jouent à la fois le rôle de classe et de requête, il est possible de combiner librement les deux modes de recherche. Un autre avantage est que le treillis de concepts est automatiquement constructible à partir du contexte. Il n'est donc pas besoin de maintenance manuelle de l'organisation du système d'information. Par ailleurs, ce contexte peut être généré par des méthodes d'indexation traditionnelles. Le treillis de concepts apparaît donc comme une alternative intéressante aux méthodes conventionnelles, pour la récupération d'informations.

Une comparaison expérimentale des trois méthodes évoquées ci-dessus retient les 3 critères suivant [GMA93] :

le temps de recherche : temps mis par l'utilisateur pour accomplir une recherche ; c'est une bonne mesure de l'effort fourni, si on considère les temps de réponse du système comme non déterminants,

la précision : proportion de réponses pertinentes parmi les réponses données,

le rappel : proportions de réponses données parmi les solutions pertinentes.

Les expériences ont été réalisées [GMA93] sur une petite base de données de 113 courtes descriptions de films, et par 20 personnes. Ces personnes n'étaient pas toutes informaticiennes, et il faut noter qu'elles pouvaient consulter les documents avant de commencer les expériences. On a obtenu les résultats suivants. Pour le temps de recherche, c'est le modèle booléen qui obtient les meilleurs résultats (3,55), tandis que les modèles hiérarchiques et conceptuel obtiennent des résultats semblables (resp. 3,90 et 3,95) [9]. Mais il faut nuancer ces résultats sachant que les modèles booléen et hiérarchiques ont été expérimentés sur des outils commerciaux, alors que le modèle conceptuel a été expérimenté sur un prototype dont l'interface a posé quelques difficultés aux utilisateurs. Pour

9. On ne connaît pas l'unité de ces résultats (minutes?), mais on peut au moins exploiter les rapports entre les valeurs.

le rappel, les modèles booléen et conceptuel donnent des résultats très proches (resp. 80,9% et 79,5%). Cela peut paraître surprenant si on considère que l'expressivité du modèle conceptuel est limité aux conjonctions (pour l'AC classique), mais il semble que la navigation compense ce défaut en permettant un raffinement progressif de la requête. Par contre, le modèle hiérarchique a un taux de rappel nettement moindre (70,5%). Les causes possibles sont la construction manuelle de la hiérarchie qui peut entraîner des erreurs de classement, des ambiguités conceptuelles quant à l'intention des répertoires et le fait qu'il n'y ait qu'un seul chemin d'accès au document cherché. Enfin, pour la précision, les expériences n'ont pas fait apparaitre de différence significative entre les trois modèles.

En conclusion, si le modèle booléen a les meilleurs résultats pour les 3 critères, l'absence de navigation peut le rendre rebutant pour les utilisateurs occasionnels et rend aussi difficile le contrôle du volume des réponses (problème bien connu avec les moteurs de recherche sur Internet). Quant au modèle hiérarchique, la récupération d'informations y est incertaine en raison du faible taux de rappel. Le modèle conceptuel a des résultats peu éloignées du modèle booléen et offre en plus des possibilités de navigation sans avoir à maintenir manuellement une structure de navigation et tout en permettant le mode interrogationnel. Finalement, le treillis de concepts est une alternative intéressante aux méthodes conventionnelles d'organisation et de récupération d'informations.

Cette alternative est suffisamment séduisante pour que nous la choisissions comme principe d'organisation de notre SIL. De plus, nous accordons pour le moment davantage d'importance aux fonctionnalités qu'aux performances. Or, d'après la discussion précédente, le modèle conceptuel est clairement le meilleur des trois sur le plan fonctionnel.

6.2 Un prototype de SIL : le shell conceptuel

La sous-section précédente a montré que l'AC (y compris dans sa forme généralisée qu'est l'ACL) est un moyen séduisant de récupération d'informations, car elle permet de combiner étroitement interrogation et navigation et d'automatiser l'organisation. En vue d'évaluer ce modèle de système d'information, nous avons réalisé une maquette générique, dans le sens où l'on ne fait aucune hypothèse sur la logique employée (mis à part celles posées par l'ACL). Ce sont donc les applications particulières de cette maquette qui doivent définir la logique employée (syntaxe des formules, déduction logiques, opérations de disjonction et de conjonction). L'interface de la maquette est un langage de commandes très proche de celui du système de fichiers UNIX (commandes `cd`, `ls`, `cp`, etc.), ce qui permet de l'apprendre très rapidement. C'est à cause de cette interface que nous appelons notre maquette *shell conceptuel*.

6.2.1 Spécification informelle

Les données de départ sont celles d'un contexte formel $K = (\mathcal{O},\mathcal{L},i)$, où i est la fonction qui associe à chaque objet la description logique de ses propriétés ou carctéristiques. Les commandes du shell conceptuel sont celles du shell UNIX, ré-interprétées dans le cadre de l'ACL. Les changement principaux sont le remplacement des *chemins* (*path*) par des formules de $\mathcal{L}$ et l'utilisation de la relation de déduction contextualisée $\dot{\models}^K$ (isomorphe à l'ordre sur les concepts, cf. section 3) plutôt que la relation de déduction intiale $\dot{\models}$. Pour le reste, les effets des commandes sont très semblables.

Tout d'abord, on introduit quelques expressions utilisées dans la suite. Soit une formule f :

- le *concept* f est le concept associé à f par la fonction d'étiquetage μ (cf. définition 5), c-à-d. $\mu(f)$; les formules sont des représentations des concepts et les concepts sont des classes d'équivalence de formules modulo $\dot{\equiv}^K$ (équivalence contextualisée),
- l'*extension de* f est en fait l'extension du concept f ; c'est aussi l'ensemble des objets dont la description satisfait f,
- l'*objet de* f est l'objet, s'il existe, dont la description est *contextuellement* équivalente à f (il est unique par convention pour éviter les ambiguités d'accès) ; f joue alors le rôle de *description contextuelle* pour cet objet ; un objet peut avoir plusieurs descriptions contextuelles (plusieurs chemins d'accès en quelque sorte), et sa description en fait toujours partie (car $i(o)\dot{\equiv}^K i(o)$).

Avant de décrire nos spécifications, commençons par établir une analogie entre les termes du shell UNIX et ceux de notre shell conceptuel :

shell UNIX	shell conceptuel
fichier	objet
path/chemin	formule logique
nom absolu d'un fichier	description d'un objet
répertoire	formule/concept
racine	formule $\dot{\top}$/concept $\top^c$
répertoire de travail	requête/concept de travail

répertoire de travail :
Il est remplacé par une pile de formules correspodant à l'*historique* de la navigation. Le sommet de cette pile représente la *requête de travail*, que l'on note wq (pour *working query*).

`pwd` :
Affiche la requête de travail wq.

`cd ..` :
Dépile l'historique, à moins que celui-ci ne contienne qu'un élément. Cette commande permet de faire des retours-arrières dans la navigation, et remplace le déplacement vers le répertoire père qui n'est plus possible tel quel puisque la structure de navigation est ici un treillis (cette commande est en fait analogue à la commande "Back" des navigateurs Web).

`cd` to :
Notons $l(to)$ la composition (essentiellement une conjonction) de to avec wq. Cette commande empile $l(to)$ dans l'historique, et permet de se "déplacer" vers son concept. La fonction de composition l nous laisse la possibilité d'une notation qui distingue formule *relative* qu'on combine avec wq (ex. $wq := wq \dot{\wedge} to$) et formule *absolue* qu'on ne combine pas (ex. $wq := to$).

`ls` f :
Affiche tout d'abord l'identité de l'objet de $l(f)$, s'il existe, puis une liste de formules (de préférence simples), appelées *incréments*, qui permettent de raffiner la requête $l(f)$, tout en évitant de mener à la requête vide $\perp^c$ et en assurant que tout élément de l'extension de $l(f)$ soit accessible en utilisant uniquement les incréments donnés par la commande `ls` (condition de complétude).

`ls -r` f :
Affiche pour chaque élément de l'extension de $l(f)$, son identité et sa description auquel on a "retranché" $l(f)$.

`mkfile` f c :
Crée un nouvel objet de contenu c et de description $l(f)$.

`chfile` f c :
Modifie l'objet de $l(f)$, s'il existe, en remplaçant son contenu par c.

`edit` f :
Affiche l'identité, la description et le contenu de l'objet de $l(f)$, s'il existe.

`edit -r` f :
Affiche l'identité, la description et le contenu de tous les éléments de l'extension de $l(f)$.

`rm` f :
Supprime l'objet de $l(f)$, s'il existe.

`rm -r` f :
Supprime tous les éléments de l'extension de $l(f)$.

`cp` $from$ to :
Duplique l'objet de $l(from)$, s'il existe, en dupliquant le contenu et en "translatant" sa description de $l(from)$ vers $l(to)$, c-à-d. en "retranchant" $l(from)$ puis en "ajoutant" $l(to)$ (l'"ajout" correspond à la conjonction et le "retranchement" correspond à la complémentation relative, définie à la sous-section 4.2).

`cp -r` $from$ to :
Duplique chaque élément de l'extension de $l(from)$ en dupliquant le contenu et en "translatant" sa description de $l(from)$ vers $l(to)$.

`ln` $from$ to :
Duplique l'objet de $l(from)$, s'il existe, en partageant le contenu et en "translatant" sa description de $l(from)$ vers $l(to)$.

`ln -r` *from to* :
: Duplique chaque élément de l'extension de $l(from)$ en partageant son contenu et en "translatant" sa description de $l(from)$ vers $l(to)$.

`mv` *from to* :
: Déplace l'objet de $l(from)$, s'il existe, en "translatant" sa description de $l(from)$ vers $l(to)$ (l'identité et le contenu de l'objet sont inchangés).

`mv -r` *from to* :
: Déplace chaque élément de l'extension de $l(from)$ en "translatant" sa description de $l(from)$ vers $l(to)$ (l'identité et le contenu des objets sont inchangés).

Les commandes `cd` et `ls` sont celles qui permettent l'interrogation et la navigation. La requête `cd` est simple et intervient essentiellement dans la gestion de la requête courante. La commande `ls` mérite davantage d'explications. Avec l'option `-r` le résultat est celui que les systèmes d'interrogation donne à une requête : la liste des objets satisfaisant la requête. Sans cette option, la commande `ls` a pour but de permettre la navigation, c-à-d. de permettre la recherche d'objets par raffinements successifs de la requête courante : un incrément x permet ainsi de raffiner la requête courante wq en $wq \dot{\wedge} x$.

Mais la nouvelle requête $wq \dot{\wedge} x$ peut donner autant de réponses que l'ancienne (pas assez de raffinement) ou bien aucune réponse (trop de raffinement). Pour éviter ces cas défavorables, on doit poser la condition suivante pour tout incrément x et pour une requête courante donnée wq (rappelons que $\tau(f)$ dénote l'extension de la requête f) :

$$\emptyset \subset \tau(wq \dot{\wedge} x) \subset \tau(wq).$$

On peut démontrer que cette condition est équivalente à

$$\perp^c <^c \mu(wq) \wedge^c \mu(x) <^c \mu(wq).$$

Cela montre non seulement la pertinence du treillis de concepts pour caractériser la navigation, mais aussi une certaine nécessité car la condition ci-dessus n'est pas exprimable dans la seule logique $\mathcal{L}$. La difficulté est ensuite de trouver un ensemble fini d'incréments *Inc* qui satisfasse cette condition et qui soit *complet*, c-à-d. que pour toute requête courante wq et pour tout objet o de l'extension courante, il existe un incrément x dans *Inc* qui restreint strictement l'extension courante tout en conservant o dans la nouvelle extension. Cette condition de complétude garantit qu'en utilisant seulement les incréments pour raffiner les requêtes, on peut atteindre n'importe quel objet.

Dans les commandes entraînant une modification du contexte (`mkfile`, `chfile`, `rm`, `cp`, `ln`, `mv`) le treillis de concepts est implicitement adapté. En fait, le principe de nos SIL est de fournir des accès basés sur les propriétés des objets et non sur une structure d'organisation figée, quelle qu'elle soit. Le treillis de concept n'échappe pas à la règle : il est intéressant pour modéliser et implémenter nos SIL, mais l'utilisateur n'a pas besoin de le connaître, ni de le visualiser explicitement. Les seules notions utiles pour l'utilisateur sont les notions d'extension et de logique contextualisée.

6.2.2 Quelques éléments d'implémentation

Nous n'entrerons pas dans les détails de l'implémentation, car ce n'est pas le propos de ce rapport. Cependant, nous donnons quand même une idée de la structure choisie pour implémenter l'interrogation et la navigation conceptuelles. Toutefois, il faut noter que cette structure n'est qu'une première proposition, et il reste à l'évaluer et à la comparer à d'éventuelles autres solutions.

Un principe de base du shell conceptuel est d'accéder à des objets (et d'effectuer des opérations sur eux) à partir de formules. Les accès fondamentaux consistent donc à passer d'une formule à un ou des objets et interviennent dans la plupart des commandes du shell. Il y a deux solutions extrêmes pour implémenter ces accès. La première est de ne s'appuyer sur aucune structure et de ré-évaluer ces accès complètement à chaque fois en se basant sur leur définition. Pour $\tau(f)$ (l'extension de f), cela revient à évaluer $i(o) \dot{\models} f$ pour tout objet o du système, ce qui est inacceptable si on considère que le système peut compter un grand nombre d'objets (ex. plusieurs milliers pour un système de fichiers personnel). La deuxième s'appuierait sur une représentation du treillis de concepts. Cependant, bien que ce treillis soit fini, sa taille croît avec la richesse de la logique choisie et le nombre d'objets et elle devient vite inacceptable. Heureusement, une telle représentation n'est pas nécessaire.

La solution que nous proposons consiste à représenter uniquement un sous-diagramme de Hasse, c-à-d. un sous-graphe anti-réflexif et anti-transitif, $(F, \dot{\prec})$ du treillis (infini) des formules $\langle \mathcal{L}; \dot{\models} \rangle$, où F contient au moins $i(\mathcal{O})$ (les descriptions des objets), $\dot{\top}$ et $\perp$ (les bornes de $\mathcal{L}$, cf. sous-section 4.1). Pourquoi utiliser un graphe de

formules plutôt que de concepts? Premièrement, ce qui nous intéresse dans les concepts, ce sont les extensions (rappellons-nous que l'on cherche à passer des formules aux objets), et non les intentions qui peuvent être trop compliquées pour être exploitables (ex. cas où la disjonction est réalisée par un constructeur). Néanmoins, n'importe quelle formule associée à un concept par μ fait l'affaire pour le représenter. De plus, l'extension d'un concept est liée à toute formule le représentant, par la fonction τ qui peut être définie sans faire intervenir la moindre notion sur les concepts ($\forall f \in \mathcal{L} : ext(\mu(f)) = \tau(f)$) : le graphe des formules est donc suffisant pour retrouver les informations relatives aux extensions, et donc aux concepts. Deuxièmement, le graphe des formules est plus simple à utiliser, car les paramètres des commandes sont des formules et non des concepts, et à maintenir, car il est stable (il est défini lors du choix de la logique et en particulier par $\dot{\models}$), alors que le treillis des concepts évolue en même temps que le contexte formel (c-à-d. à chaque fois qu'un objet est ajouté, modifié ou supprimé). Pourquoi choisir un sous-graphe anti-réflexif et anti-transitif? Cela permet d'éviter les redondances et d'alléger la structure de données. De plus, on retrouve facilement la relation $\dot{\models}$ à partir de $\prec$ (par fermeture réflexive et transitive). Pourquoi F doit-il contenir $i(\mathcal{O})$, $\dot{\top}$ et $\dot{\bot}$? Les bornes $\dot{\top}$ et $\dot{\bot}$ facilitent les parcours dans le graphe de formules, et les noeuds du graphe étiquetés par les descriptions d'objets permettent d'y attacher les objets correspondants. Un algorithme permet d'insérer de nouvelles formules dans le graphe et nécessite un démonstrateur automatique pour la logique $\mathcal{L}$; quelques autres algorithmes réalisent les accès élémentaires du shell par simple parcours du graphe de formules.

6.2.3 Quelques expérimentations

Une application concrète du shell conceptuel a été réalisé pour une première expérimentation de notre SIL. Le domaine d'application choisi est la cuisine, et plus précisément la cuisine vietnamienne. Les objets du SIL sont donc des recettes, dont les descriptions logiques donnent ses ingrédients (ex. crevette, céleri), le type du plat (ex. volaille, dessert), les menus composés à partir de ce plat et le nom vietnamien de la recette. Pour cette première expérience, la logique a été choisie simple : il s'agit de la logique à attributs sous-jacente à l'AC classique. Ce domaine est intéressant d'après nous car il n'existe pas d'organisation hiérarchique des recettes qui soit satisfaisante. En fait, dans le livre de recettes d'où nous tirons nos données [RG95], les recettes sont classées en fonction de leur type (ex. volaille, dessert, etc.). Mais cela ne permet ni de choisir un dessert parmi tous ceux qui existent, ni de faire des accès associatif tel que la recherche d'un plat à base de poisson et de gingembre. Le problème avec l'organisation hiérarchique, ici, est que les relations entre les recettes et les ingrédients sont très "croisées", c-à-d. qu'un ingrédient peut être utilisé dans plusieurs recettes et qu'une recette utilise plusieurs ingrédients ; et on observe la même chose pour les relations entre recettes et menus.

Une base de 101 recettes a été entrée avec une centaine d'attributs. Les expériences que nous avons menées montrent que, conformément à ce qui était avancé à la section 6.1, l'organisation d'un système d'information par un treillis de concepts permet une combinaison étroite des modes de recherche interrogationnel et navigationnel. En effet, il est possible de n'utiliser que l'interrogation, que la navigation ou bien toute combinaison des deux, y compris les entrelacements. Comme l'interrogation et la navigation utilisent le même langage (la logique $\mathcal{L}$), l'interrogation peut être vue comme la navigation suivant un chemin connu de l'utilisateur, et la navigation peut être vue comme de l'interrogation à partir des "suggestions" de la commande `ls`. Rappelons que ce n'est pas le cas dans les systèmes hybrides [GJSO91, GM99]. Par rapport aux systèmes hiérarchiques, on peut aussi relever la possibilité pour notre SIL d'effectuer un traitement (ex. déplacement avec la commande `mv -r`) sur un ensemble presque arbitaire d'objets (ce degré d'arbitraire est lié à l'expressivité de la logique utilisée).

Du point de vue de l'utilisateur du shell conceptuel, les concepts n'apparaissent pas explicitement, mais ils se manisfestent néanmoins concrètement. Par exemple, bien que les recettes soient associées en moyenne à une dizaine d'attributs, nous avons observé qu'il suffit le plus souvent d'en spécifier 3 ou 4 (ce qui correspond à 3 ou 4 pas de navigation) pour accéder à n'importe quelle recette. Les concepts permettent donc de concilier précision des descriptions d'objets et concision des requêtes. Une autre manifestation des concepts est que les ensembles d'objets que l'on peut manipuler d'un bloc (par les commandes avec option `-r`, cf. section 6.2.1) correspondent exactement aux extensions des concepts, car toute formule a pour extension, l'extension du concept qu'elle désigne (formellement $\tau(f) = ext(\mu(f))$).

7 Conclusion

Nous avons montré comment l'ACF peut être reconstruite quand les ensembles d'attributs sont remplacés par des formules d'une logique (presque) arbitraire. Un sous-produit de cette reconstruction est la dérivation d'une logique contextualisée qui ajoute aux déductions qui sont valides dans la logique, de nouvelles déductions

qui sont valides uniquement pour le contexte formel. Cette logique contextualisée joue le même rôle que le treillis de concepts et correspond à l'étude des implications entre attributs [GW99a]. C'est la logique contextualisée ainsi dérivée que nous projetons d'utiliser pour la navigation dans un système d'information logique. L'intérêt de l'AC dans ce domaine d'applications est de permettre une combinaison étroite de l'interrogation et de la navigation : les concepts sont à la fois des requêtes et des endroits où il est possible à la fois de consulter et d'écrire.

La logique utilisée pour remplacer les attributs est presque arbitraire, mais de nouvelles contraintes portent sur elles lorsqu'on se place dans la perspective de développer un système d'information logique. En effet, les opérations de base (ex. déduction dans $\mathcal{L}$, calcul de l'extension d'une formule) doivent être calculables, et doivent avoir une complexité raisonnable. Par exemple, nous envisageons d'appliquer l'ACL au domaine du génie logiciel. Dans ce cadre, les logiques de descriptions (DL [Nap97]), qui sont des logiques décidables et cependant expressives, peuvent être utilisées pour la gestion de versions et de configurations [Zel98] ; et la logique des isomorphismes de types [Di 95] peut être appliquée à la navigation dans les bibliothèques de composants logiciels [10].

A un niveau plus théorique, deux directions importantes pour de futurs travaux sont la possibilité d'avoir des relations dans le contexte formel pour structurer l'ensemble des objets, et la possibilité de focalisation offerte par les *vues* (comme dans les bases de données, par exemple).

Remerciements

Ces travaux sont soutenus par une bourse du Centre National de la Recherche Scientifique (CNRS) et de la région Bretagne.

Références

[CC92] Cousot (Patrick) et Cousot (Radhia). – Abstract interpretation and application to logic programs. *Journal of Logic Programming*, vol. 13, n2-3, juillet 1992, pp. 103–179.

[CGT90] Ceri (Stefano), Gottlob (Georg) et Tanca (Letizia). – *Logic Programming and Databases.* – Berlin, Heidelberg, New York, Springer-Verlag, 1990, *Surveys in Computer Science.*

[CM92] Chein (M.) et Mugnier (M-L.). – Conceptual graphs: fundamental notions. *Revue d'Intelligence Artificielle*, vol. 6, n4, 1992, pp. 365–406.

[CM98] Chaudron (L.) et Maille (N.). – 1st order logic formal concept analysis: from logic programming to theory. *Computer and Information Science*, vol. 13, n3, 1998.

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

[DP90] Davey (B. A.) et Priestley (H. A.). – *Introduction to Lattices and Order.* – Cambridge University Press, 1990.

[GJSO91] Gifford (David K.), Jouvelot (Pierre), Sheldon (Mark A.) et O'Toole (James W. Jr). – Semantic file systems. *In : Proceedings of 13th ACM Symposium on Operating Systems Principles.* pp. 16–25. – ACM SIGOPS, octobre 1991.

[GM99] Gopal (Burra) et Manber (Udi). – Integrating content-based access mechanisms with hierarchical file systems. *In : Proceedings of third symposium on Operating Systems Design and Implementation.* pp. 265–278. – USENIX Association, 1999.

[GMA93] Godin (R.), Missaoui (R.) et April (A.). – Experimental comparison of navigation in a Galois lattice with conventional information retrieval methods. *International Journal of Man-Machine Studies*, vol. 38, n5, 1993, pp. 747–767.

[GW99a] Ganter (B.) et Wille (R.). – *Formal Concept Analysis — Mathematical Foundations.* – Springer, 1999.

[GW99b] Ganter (Bernhard) et Wille (Rudolf). – Contextual attribute logic. *Lecture Notes in Computer Science*, vol. 1640, juillet 1999, pp. 377–388.

10. Notez que l'isomorphisme de types est une théorie sémantique, et donc qu'une navigation basée sur cette logique diffère fondamentalement des autres navigations proposées qui sont basées sur la documentation ou sur les mots-clés [Lin95].

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

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

[Nap97] Napoli (A.). – *Une introduction aux logiques de descriptions*. – Technical Report nRR-3314, Inria, Institut National de Recherche en Informatique et en Automatique, décembre 1997.

[Plo70] Plotkin (G. D.). – A note on inductive generalization. *Machine Intelligence, Edinburgh Univ. Press*, no5, 1970, pp. 153–163. – Edinburgh Univ. Press, Edinburgh.

[Pre97] Prediger (Susanne). – Logical scaling in formal concept analysis. *Lecture Notes in Computer Science*, vol. 1257, août 1997, pp. 332–341.

[Pre98] Prediger (Susanne). – Simple concept graphs: A logic approach. *Lecture Notes in Computer Science*, vol. 1453, août 1998, pp. 225–239.

[RG95] Rhodes (Ione) et Gelbon (Marie-Claude). – *Le chant du riz pilé — cent recettes vietnamiennes*. – L'Harmattan, 1995.

[Sne98] Snelting (G.). – Concept analysis — A new framework for program understanding. *ACM SIGPLAN Notices*, vol. 33, n7, juillet 1998, pp. 1–10.

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

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

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399