UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

CONCEPTION ET IMPLÉMENTATION DE NOUVELLES FONCTIONNALITÉS DANS UN PROTOTYPE DE FOUILLE DE DONNÉES

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN SCIENCES ET TECHNOLOGIES DE L'INFORMATION

PAR

KÉVIN EMAMIRAD

MAI 2017

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Ce mémoire intitulé :

CONCEPTION ET IMPLÉMENTATION DE NOUVELLES FONCTIONNALITÉS DANS UN PROTOTYPE DE FOUILLE DE DONNÉES

présenté par
Kévin Emamirad

pour l'obtention du grade de maître ès science (M.Sc.)

a été évalué par un jury composé des personnes suivantes :

Prof. Rokia Missaoui ..... Directrice de recherche
Prof. Nadia Baaziz ..... Présidente du jury
Prof. Jean-François Viaud ..... Membre du jury

Mémoire accepté le : 31 mai 2017

Je voudrais exprimer mes remerciements sincères à Prof. Rokia Missaoui qui par son expérience et son enthousiasme m'a énormément aidé dans la réalisation de mon mémoire. Un grand merci aussi à Prof. Nadia Baaziz et Prof. Jean-François Viaud pour le temps qu'ils m'ont consacré en tant que membres de mon jury et pour leurs commentaires pertinents.

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations, sigles et acronymes

AFC Analyse formelle de concepts

AJAX *Asynchronous JavaScript and XML*

ConExp *Concept Explorer*

CSV *Comma Separated Values*

FCA *Formal concept analysis*

Résumé

Partant d'un contexte formel $\mathbb{K} = (G, M, I)$ décrivant un ensemble G d'objets, un ensemble M de propriétés (attributs) et une relation binaire I entre G et M, l'analyse formelle de concepts (AFC) est une technique de regroupement (*clustering*) qui est fondée sur la théorie des treillis et qui permet de générer un ensemble de concepts formels (*clusters*) reliés entre eux par une relation d'ordre partiel et de produire des bases de règles d'association incluant des implications. Le but de ce travail est d'enrichir un prototype de fouille de données, appelé *Lattice Miner*, qui exploite l'AFC et les treillis de concepts (Galois) en vue d'atteindre les objectifs suivants :

1. production d'implications avec négation,
2. enrichissement du module de génération des règles triadiques pour obtenir exhaustivement et précisément les trois formes d'implications triadiques définies par Ganter et Obiedkov,
3. validation intensive de la procédure de production de la base d'implications de Guigues–Duquenne et de la procédure de calcul des relations de flèches. cette dernière est utile dans le processus de décomposition de contextes formels, et
4. amélioration de la convivialité de l'interface usager.

Chapitre 1

Introduction

1.1 Problématique

D'après une étude de la firme EMC^2, la quantité des données numériques double tous les deux ans et passera de 4,4 zettaoctets (plus de 4 000 milliards de Go) en 2013 à 44 zettaoctets en 2020 [1]. Avec cette augmentation exponentielle des données, la fouille de données qui permet d'extraire des connaissances à partir de données prétraitées, suscite de plus en plus l'intérêt aussi bien des chercheurs universitaires que des acteurs industriels du domaine des technologies de l'information. À l'ère des données massives (*Big Data*) et de l'internet des objets (*Internet of Things*), les outils de fouille de données doivent être plus efficaces et offrir une variété de fonctions d'exploration conviviale et interactive des données. Pour cela, vu la très grande quantité de connaissances produites par les outils de fouille de données, il faut que ces derniers évitent de produire des connaissances redondantes ou celles qui ne ciblent pas les besoins des utilisateurs.

L'analyse formelle de concepts (AFC) est un formalisme de représentation des connaissances [13] et de fouille de données qui permet principalement la construction d'un treillis de concepts (Galois) et des règles d'association à partir d'un contexte souvent binaire décrivant un ensemble d'objets par un ensemble de propriétés. Elle a été appliquée avec succès dans divers domaines en informatique, linguistique, sociologie, biologie, etc. L'une des fonctionnalités intéressantes de l'AFC est sa capacité à produire des visualisations graphiques des structures inhérentes aux données [29] sous forme de treillis de concepts. Toutefois, en présence de données très volumineuses, le nombre de nœuds et de liens aug-

1. Étude disponible à l'adresse suivante :
`http://canada.emc.com/leadership/digital-universe/2014iview/executive-summary.htm`

mente très rapidement et le treillis devient complexe, ce qui peut rendre sa visualisation et son exploration très laborieuses.

Dans ce qui suit, nous expliquons le contexte de travail ainsi que les objectifs spécifiques de ce projet de recherche qui utilise l'AFC comme cadre pour la fouille interactive et avancée de données.

1.2 Contexte

La *fouille de données* [15] appelée aussi forage de données ou orpaillage est une étape du processus de découverte de connaissances qui consiste à extraire des connaissances à partir d'une grande quantité de données en faisant appel à diverses méthodes, techniques et outils. Il existe trois principales formes de fouille des données :

- La prédiction. Il s'agit d'un apprentissage supervisé qui consiste à prédire la valeur d'une variable (régression) ou la classe d'appartenance (classification) d'une observation ou d'un individu.
- La découverte. Il s'agit d'un apprentissage non supervisé de type analyse exploratoire où l'on peut découvrir des règles d'association comme l'achat du lait ⇒ l'achat du beurre, des motifs séquentiels (ex. Si un client acquiert un téléviseur, alors il a une probabilité de 80% de souscrire au câble dans un délai de deux mois) ou des grappes (*clusters*) comme la détection de communautés dans un réseau social ou la détermination des divers profils d'internautes.
- La détection de déviation comme des valeurs exceptionnelles ou des tendances au niveau des ventes.

Au cours des deux dernières décennies, on a vu apparaître plusieurs outils d'analyse formelle de concepts tels *ToscanaJ, ConExp, Coron, Java Lattices*, et *Lattice Miner* [4, 26, 16, 30, 22, 5]. Les outils existants permettent de générer des treillis de concepts et des règles d'association, mais plusieurs d'entre eux sont conçus pour des utilisateurs avertis et les fonctionnalités d'exploitation et de visualisation du treillis sont parfois limitées. Un bon outil est celui qui serait conçu sur la base des nouvelles technologies logicielles et matérielles et qui permettrait de (i) gérer efficacement des données massives, (ii) visualiser le treillis de concepts de manière intuitive et interactive en mettant en pratique différentes techniques de visualisation de l'information et de la connaissance, et (iii) générer des connaissances pertinentes et précises selon les besoins des utilisateurs.

1.3 Objectifs

Le but de ce mémoire de maîtrise est d'enrichir un prototype existant de fouille de données spécialement dédié à la production de concepts et de règles d'association en AFC. L'outil en question s'appelle *Lattice Miner* et a été initialement développé par Geneviève Roberge [30] et progressivement enrichi par plusieurs étudiants du laboratoire LARIM sous la supervision de Professeure Rokia Missaoui. Dans son état actuel, cet outil est déjà disponible sur SourceForge, a été téléchargé plus de 5 000 fois et nécessite des extensions au niveau de l'introduction de la négation dans les implications, la production d'implications triadiques à la Ganter et la validation à la fois de la procédure de calcul des relations de flèches et celle de production de la base d'implications Guigues–Duquenne [14].

Le présent document est organisé comme suit. Au chapitre 2, nous rappelons les bases de l'analyse formelle de concepts. Le chapitre 3 décrit différents outils déjà existants de l'AFC et permettant la construction, la visualisation et la manipulation des treillis de concepts. Nous allons également présenter *Lattice Miner* dans son état actuel et faire mention de deux procédures corrigées : le calcul des relations de flèches et la production de la base d'implications de Guigues-Duquenne. Par la suite, nous décrivons les deux nouvelles fonctionnalités introduites dans *Lattice Miner*, à savoir les implications avec négation au chapitre 4 et les implications triadiques au chapitre 5. Le chapitre 6 présente une conclusion du mémoire ainsi que les fonctionnalités que l'on pourrait rajouter dans le futur dans *Lattice Miner*.

Chapitre 2

Rappels

Ce chapitre est une présentation des différentes notions reliées à l'analyse formelle de concepts et utiles pour la suite. Il est constitué de plusieurs définitions et rappels.

2.1 Analyse formelle de concepts

L'analyse formelle de concepts [13] est un formalisme de représentation et de découverte de la connaissance, basé sur la formalisation des concepts et la hiérarchie de concepts. Elle est considérée comme une méthode de regroupement de conceptuel *(conceptual clustering)* puisqu'elle permet de produire des groupes homogènes à la fois chevauchants (*overlapping clusters*) et conceptuels du fait que chaque groupe représenté par un concept formel est décrit non seulement par son extension, mais également par son intention (sa description).

Définition 2.1. Soit $\mathbb{K} = (G, M, I)$ *un contexte formel* où G, M et I sont respectivement un ensemble d'objets, une collection d'attributs et une relation binaire entre G et M. L'expression $(g, m) \in I$ ou encore gIm signifie que l'objet g possède l'attribut m.

Le tableau 2.1 est un contexte simple avec pour objets $\{1, 2, 3, 4, 5, 6\}$ et pour attributs $\{a, b, c, d, e, f\}$.

Tableau 2.1: Exemple de contexte $\mathbb{K}$

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

Comme on le voit dans la tableau 2.1, le contexte formel est représenté sous forme d'un tableau où les objets sont en lignes et les attributs en colonnes. L'intersection des lignes et des colonnes représente la relation binaire I entre les objets et les attributs.

Définition 2.2. Un *concept formel* c est une paire d'ensembles $c := (A, B)$ avec $A \subseteq G$, $B \subseteq M$, $A = B'$ et $B = A'$, où A' est l'ensemble des attributs partagés par les objets dans A et B' est l'ensemble des objets ayant tous leurs attributs dans B. Les sous-ensembles A et B sont appelés respectivement l'extension et l'intention du concept c. Les valeurs de A' et B' sont obtenues comme suit : $A' := \{m \in M \mid gIm \ \forall g \in A\}$ et $B' := \{g \in G \mid gIm \ \forall \ m \in B\}$.

En partant du contexte précédent, la paire $(\{2, 4, 5\}, \{b, f\})$ [1] forme un concept. En effet, $A = \{2, 4, 5\} \subseteq G$, $B = \{b, f\} \subseteq M$. L'ensemble des attributs partagés par les objets dans A, c'est-à-dire par les objets 2, 4 et 5, coïncide avec $\{b, f\}$, donc $B = A'$. De même, l'ensemble des objets possédant les attributs dans B, c'est-à-dire les attributs b et f, coïncide avec $\{2, 4, 5\}$, et donc $A = B'$.

Un sous-ensemble X est fermé si $X'' = X$. Un concept objet pour l'objet g est une paire de la forme $\gamma(g) = (g'', g')$ alors que le concept attribut pour l'attribut m est $\mu(m) = (m', m'')$. Les sous-ensembles fermés de G sont les extensions alors que les sous-ensembles fermés de M sont les intentions de $\mathbb{K}$.

En prenant l'exemple du tableau 2.1, la paire $(\{3\}, \{c, d\})$ forme un concept objet pour 3 car $\gamma(3) = (3'', 3') = (\{3\}, \{c, d\})$. De même, la paire $\mu(e) = (e', e'') = (\{1, 6\}, \{a, e\})$ forme un concept attribut pour e.

1. Dans ce qui suit, nous allons souvent simplifier la notation en écrivant par exemple $(245, bf)$ à la place de la notation $(\{2, 4, 5\}, \{b, f\})$.

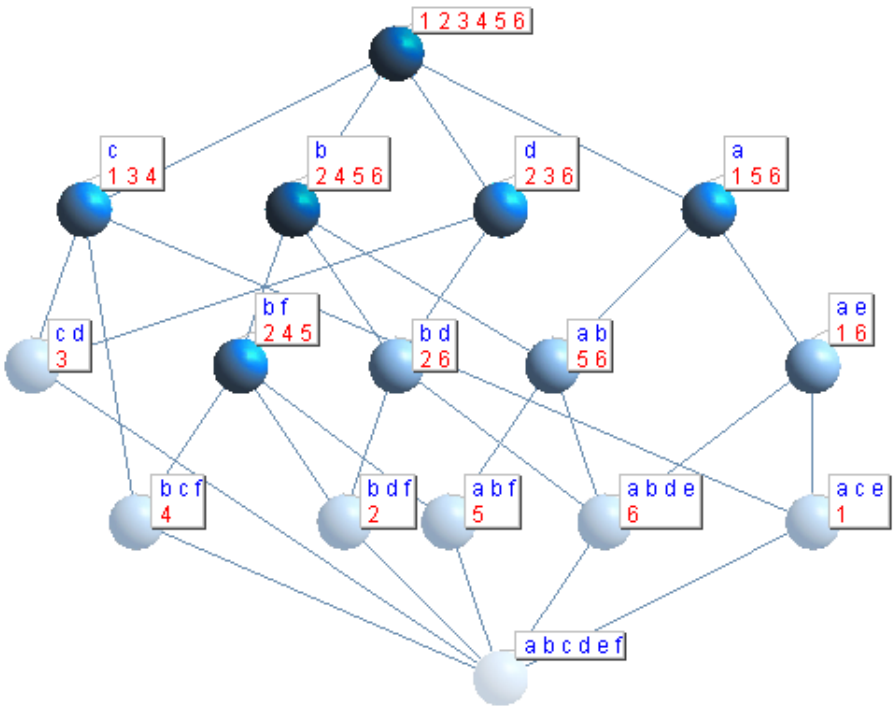
Définition 2.3. Un *treillis de concepts* $\underline{\mathfrak{B}}(G, M, I)$ [13] est un treillis résultant de l'ordre partiel existant entre les concepts du contexte $\mathbb{K} = (G, M, I)$.

$$(A, B) \leq (C, D) \Leftrightarrow A \subseteq C \text{ et } D \subseteq B$$

(A, B) est alors un sous-concept ou prédecesseur de (C, D) alors que ce dernier est un successeur de (A, B). (A, B) est dit prédécesseur immédiat de (C, D) si en plus de l'inégalité précédente, lorsqu'il existe (E, F) tel que $(A, B) \leq (E, F) < (C, D)$, alors $(A, B) = (E, F)$. Autrement dit, il n'existe pas de concept entre (A, B) et (C, D).

Nous pouvons construire le treillis avec différents outils qui seront décrits au chapitre 3.

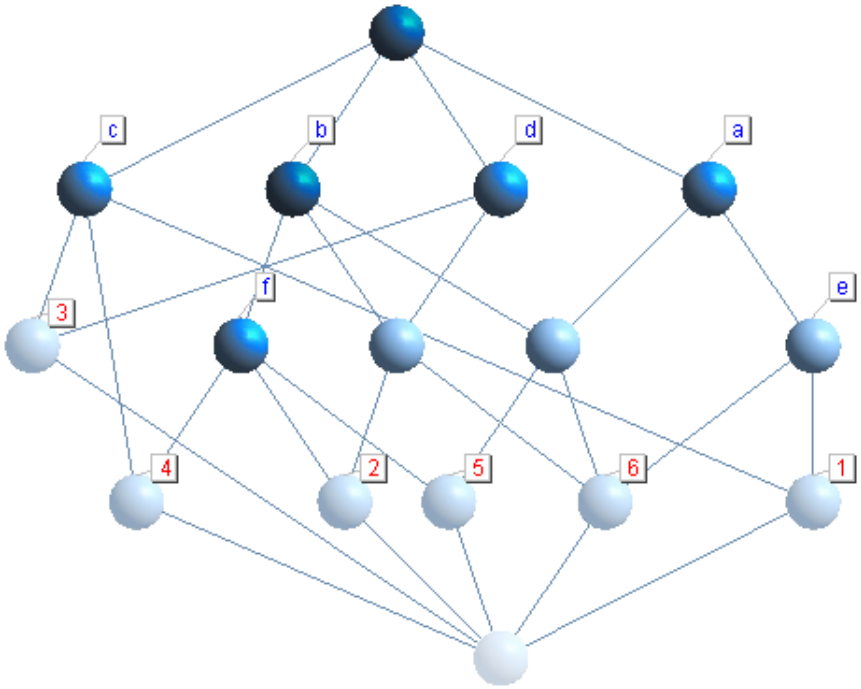
FIGURE 2.1: Exemple de treillis de concepts pour le contexte du tableau 2.1



Dans un treillis de concepts, les nœuds représentent des concepts et les arcs représentent une relation d'ordre. Comme la distance entre les nœuds n'a aucune signification particulière, nous pouvons positionner les nœuds d'une manière qui rend le treillis plus visible. Dans la figure 2.1, on est en présence d'un treillis à étiquetage complet (*full*

labeling) ce qui signifie que l'on affiche systématiquement l'extension et l'intention des divers concepts. Certains outils que nous verrons au chapitre 3 offrent un étiquetage réduit (*reduced labeling*) où un objet est indiqué uniquement la première fois qu'il est rencontré à partir du bas du treillis, et dualement, un attribut est indiqué uniquement la première fois qu'il est rencontré à partir du haut du treillis. (cf. figure 2.2)

FIGURE 2.2: Treillis avec étiquetage réduit pour le contexte du tableau 2.1



Définition 2.4. La borne inférieure $\wedge$ (*meet*) d'un ensemble de concepts (X_i, Y_i) avec $i = 1, \ldots k$ est le plus grand des prédécesseurs communs. De même, la borne supérieure $\vee$ (*join*) d'un ensemble de concepts est le petit des successeurs communs. Elles sont respectivement définies comme suit [13] :

— $\bigwedge_{i=1}^{k}(X_i, Y_i) = (\bigcap_{i=1}^{k} X_i, (\bigcup_{i=1}^{k} Y_i)'')$.
— $\bigvee_{i=1}^{k}(X_i, Y_i) = ((\bigcup_{i=1}^{k} X_i)'', \bigcap_{i=1}^{k} Y_i)$,

Un ensemble partiellement ordonné $(\underline{\mathfrak{B}}(G, M, I), \leq)$ est un treillis complet si chaque sous-ensemble de concepts dispose à la fois d'une plus grande borne inférieure et une plus petite borne supérieure. Le treillis de concepts est complet.

L'*infimum* est le plus petit concept d'un treillis de Galois alors que le *supremum* est son plus grand concept.

Le treillis de concepts décrivant les concepts et l'ordre partiel entre eux est représenté sous forme de diagramme de Hasse (cf. figure 2.1). Dans une telle représentation, lorsqu'un concept c_1 est plus petit qu'un autre concept c_2, alors le nœud représentant c_1 est placé plus bas que son successeur c_2. Lorsqu'une relation d'ordre existe entre les concepts c_1 et c_2, on dit alors que ces concepts sont *comparables* ($\leq$), sinon ils ne le sont pas ($\not\leq$).

Définition 2.5. On rappelle qu'un concept est dit *sup-irréductible* (ou *join-irréductible*) si et seulement si il possède un unique prédécesseur et il est dit *inf-irréductible* (ou *meet-irréductible*) s'il admet un unique successeur.

Par exemple, dans le treillis de la figure 2.1, le nœud $(26, bd)$ est la borne inférieure pour les nœuds $(2456, b)$ et $(236, d)$ et la borne supérieure pour les concepts $(2, dbf)$ et $(6, abde)$.

2.2 Règles d'association et implications

La génération de règles d'association [1] est un aspect important de la fouille de données. Elle a été initialement introduite en analyse du panier du consommateur par Agrawal mais peut être valablement appliquée à divers domaines. Elle consiste à extraire des associations ou relations entre les attributs *(items ou produits)* d'une base de données de transactions. Ainsi, en découvrant les associations qui relient les différents attributs d'une base de données, on peut alors évaluer leur importance principalement avec deux mesures communément connues : le support et la confiance d'une règle d'association.

Définition 2.6. Une règle d'association r a la forme $Y \rightarrow Z$ [sup, conf], où Y et Z sont des sous-ensembles d'attributs appelés *itemsets*, $Y \cap Z = \emptyset$, et *sup* et *conf* représentent respectivement le support et la confiance de la règle. Le *support* de la règle $r : Y \rightarrow Z$ [sup, conf] est $Prob(Y \cup Z)$. Il représente la proportion des objets ayant simultanément les attributs Y et Z. La *confiance* de r représente la probabilité conditionnelle $Prob(Z/Y)$. C'est donc la probabilité d'avoir Z lorsque Y est présent dans le contexte $\mathbb{K}$.

Un *itemset* (sous-ensemble d'attributs) est dit *fréquent* si la proportion d'objets le possédant est au moins égale au support minimum défini par l'utilisateur. Un *itemset* Y est dit *fermé* si $Y = Y''$. Cela signifie que Y est une intention d'un concept formel.

Il y a eu plusieurs travaux en AFC sur le calcul d'une représentation concise de règles comme la base informative (*i.e.*, la représentation qui permet la détermination du support et de la confiance de toute règle dérivable) [2, 17], la base d'implications de Guigues-Duquenne [13, 14], la base générique [27], et la base d'implications partielles de Luxenburger [20]. Les notions de *générateur* d'un itemset fermé [27, 28] et de *pseudo-intent* [13, 14] jouent un rôle clé dans de tels travaux. Nous allons donc définir ces diverses notions.

Définition 2.7. Un ensemble Z est appelé générateur minimal d'une intention Y d'un concept formel si et seulement si son fermé Z'' est égal à Y et s'il existe un autre générateur T tel que $T \subseteq Z$, alors $T = Z$.

Définition 2.8. Soient c un concept formel et $G = \{g_1, \dots g_i, \dots g_n\}$ l'ensemble de ses générateurs. Une *implication* ou *règle exacte* r a la forme $g_i \rightarrow Int(c) \setminus g_i$ avec $support(r) = support(c)$ et $confiance(r) = 100$ % où $Int(c)$ représente l'intention du concept c et $support(c)$ est la proportion d'objets contenus dans l'extension de c. Une base générique [27] d'implications est une représentation relativement concise d'implications de la forme précédente.

Définition 2.9. Soit $P = \{c_1, \dots, c_j, \dots\}$ l'ensemble des prédecesseurs du concept c et soit g_i un générateur de l'intention de c notée $Int(c)$. Une *règle approximative* r générée à partir du concept c est de la forme $g_i \rightarrow Int(c_j) \backslash Int(c)$ avec $support(r) = support(c_j)$ et $confiance(r) = support(c_j)/support(c)$.

Par exemple, en partant du concept $c = (16, ae)$ ayant pour unique générateur $\{e\}$ (cf. figure 2.3), on obtient l'implication $\{e\} \rightarrow \{a\}$ dont le support est 1/3 et deux règles d'association : $\{e\} \rightarrow \{bd\}$ [1/6, 1/2] et $\{e\} \rightarrow \{c\}$ [1/6, 1/2].

Définition 2.10. Un ensemble $P \subseteq M$ est un *pseudo-intent* de (G, M, I) [13, 14] si et seulement si $P \neq P''$ et si $Q'' \subseteq P$ est vraie pour tout pseudo-intent $Q \subseteq P$, $Q \neq P$. L'ensemble des implications de la forme :

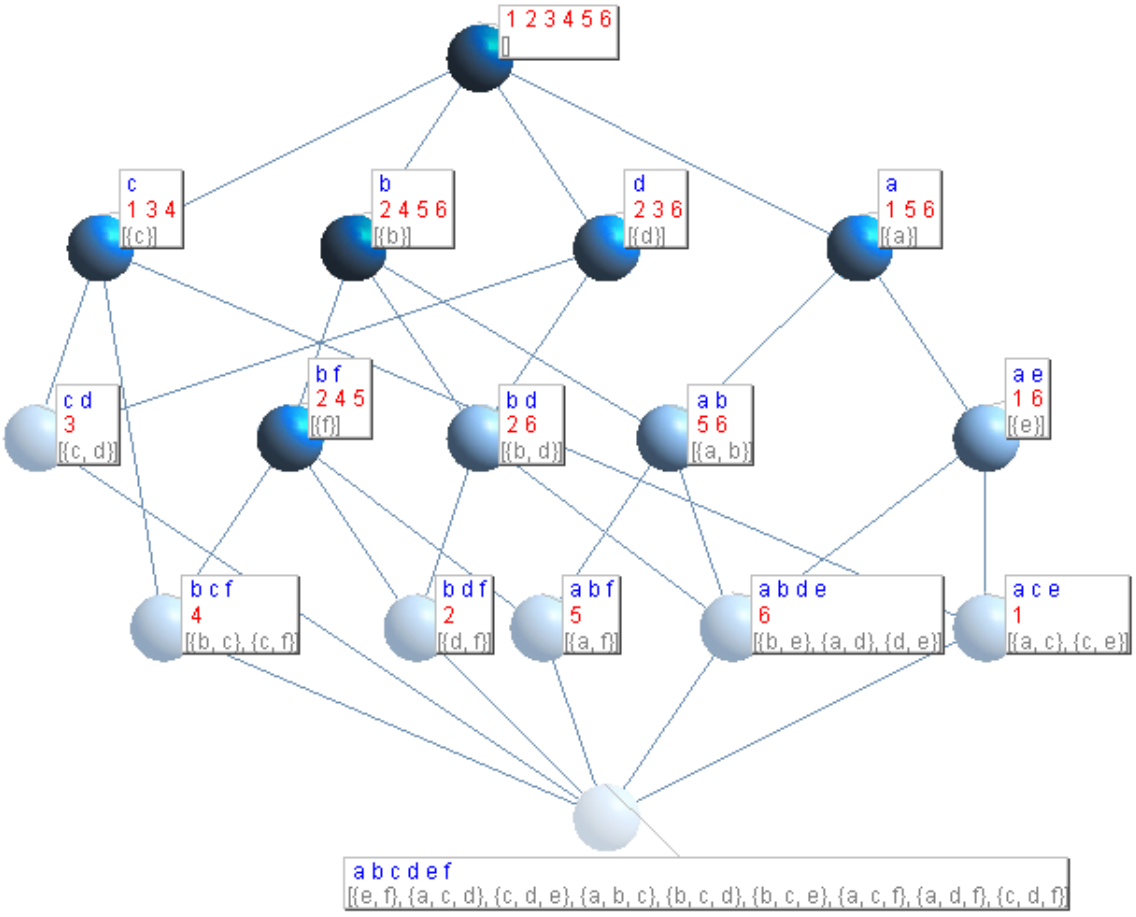
$$\mathcal{L} := \{\mathcal{P} \rightarrow \mathcal{P}'' \setminus \mathcal{P} \mid \mathcal{P} \textit{ pseudo-intent } \}$$

est appelé base de Guigues-Duquenne (*stem base*) et reconnue comme étant minimale.

À titre d'exemple, $\{a, c\}$ est un pseudo-intent dont le fermé est $\{a, c, e\}$, d'où l'implication $\{a, c\} \rightarrow \{e\}$.

Nous avons dû valider intensivement la procédure de calcul de la base de Guigues-Duquenne.

FIGURE 2.3: Treillis de concepts avec indication des générateurs



Comme le chapitre 4 traite de la production des implications avec négation, nous rappelons ci-après la définition d'un contexte complémentaire ainsi que l'opération d'apposition de contextes.

Définition 2.11. Soit un contexte $\mathbb{K} = (G, M, I)$ décrivant un ensemble G d'objets, un ensemble M de propriétés (attributs) et une relation binaire I entre G et M. Le contexte complémentaire de $\mathbb{K}$ est $\tilde{\mathbb{K}} = (G, \tilde{M}, G \times M \backslash I)$ avec $\tilde{M}$ l'ensemble des attributs négatifs.

Définition 2.12. Le contexte $\mathbb{K}|\tilde{\mathbb{K}}$ est l'apposition du contexte $\mathbb{K}$ avec son complémentaire $\tilde{\mathbb{K}}$. Cela signifie que $\mathbb{K}|\tilde{\mathbb{K}} := (G, M \cup \tilde{M}, I \cup G \times M \backslash I)$. La figure 4.1 est un exemple d'apposition d'un contexte avec son complémentaire.

2.3 Relations flèches

Les relations flèches [8, 35] sont utilisées dans un processus de décomposition de contextes en sous-contextes et définies de la manière suivante.

Définition 2.13. La relation entre l'objet g et l'attribut m dans un contexte formel se présente sous l'une des quatre formes suivantes :

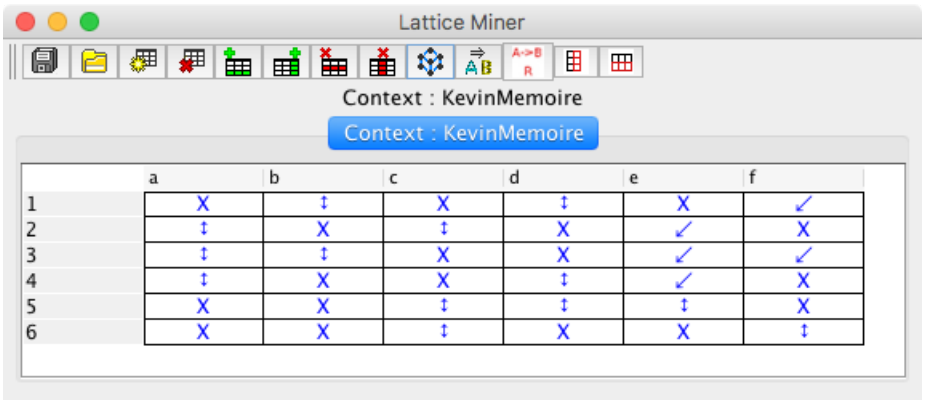
- $g \updownarrow m$ si $\gamma(g) \not\leq \mu(m)$, $\gamma(g) \leq m^+$, et $g^- \leq \mu(m)$
- $g \uparrow m$ si $\gamma(g) \not\leq \mu(m)$, $\gamma(g) \leq m^+$, et $g^- \not\leq \mu(m)$
- $g \downarrow m$ si $\gamma(g) \not\leq \mu(m)$, $\gamma(g) \not\leq m^+$, et $g^- \leq \mu(m)$
- $g \circ m$ si $\gamma(g) \not\leq \mu(m)$, $\gamma(g) \not\leq m^+$, et $g^- \not\leq \mu(m)$

où g^- représente le prédécesseur immédiat du concept objet (sup-irréductible) $\gamma(g)$ et m^+ représente le successeur immédiat du concept attribut (inf-irréductible) $\mu(m)$.

À titre d'exemple (cf. figure 2.4), la relation entre l'objet 3 et l'attribut e est $3 \downarrow e$ du fait que $\gamma(3) = (3, cd)$ n'est pas comparable à $\mu(e) = (16, ae)$, $(3, cd) \not\leq e^+$ et $3^- \leq (16, ae)$ avec $3^- = (\emptyset, abcdef)$ et $e^+ = (156, a)$.

Lattice Miner permet d'afficher les relations flèches entre les éléments d'un contexte directement à partir du contexte comme on peut le voir dans la figure 2.4 :

FIGURE 2.4: Relations flèches dans *Lattice Miner*



	a	b	c	d	e	f
1	X	↕	X	↕	X	↙
2	↕	X	↕	X	↙	X
3	↕	↕	X	X	↙	↙
4	↕	X	X	↕	↙	X
5	X	X	↕	↕	↕	X
6	X	X	↕	X	X	↕

Nous avons dû valider la procédure des relations flèches préalablement implémentée dans deux versions distinctes dont l'une était incorrecte. La validation s'est faite sur plusieurs contextes réels et générés synthétiquement en comparant les résultats obtenus par *Lattice Miner* avec ceux obtenus avec *ConExp*.

Chapitre 3

Les outils de l'analyse formelle de concepts

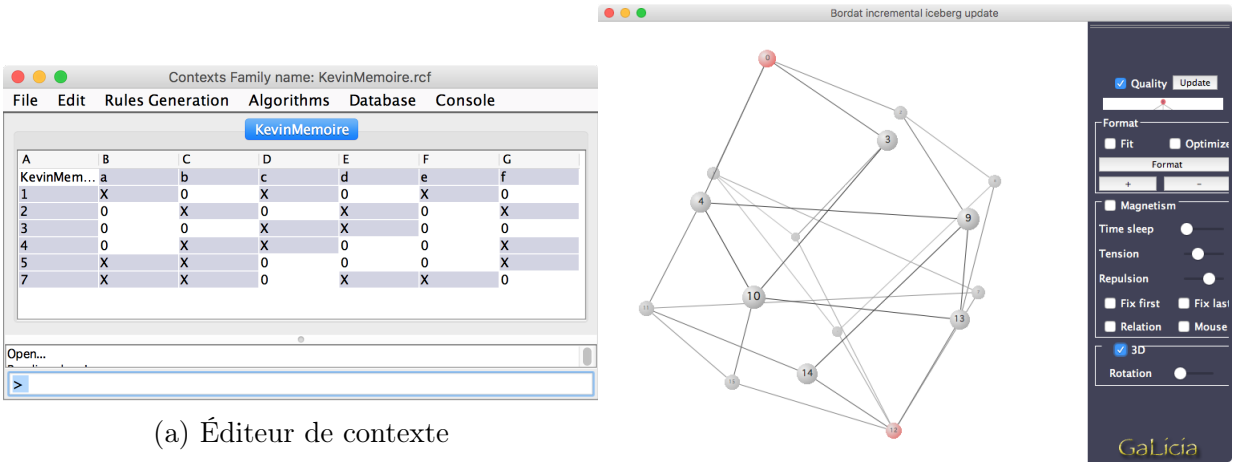
Il existe plusieurs outils logiciels en AFC qui permettent l'analyse, la construction, la visualisation et la manipulation des treillis. Certains outils ont de nombreuses fonctionnalités alors que d'autres sont relativement limités. Cette section fait une revue de quelques outils [1] qui existent dans le domaine de l'analyse formelle de concepts.

3.1 Galicia

Galicia [2] est une plate-forme du domaine public qui offre un support au cycle complet de création des treillis de concepts. Elle a été développée en Java par Petko Valchev et ses collaborateurs dans le but de regrouper dans une même application toutes les opérations de base de manipulation et d'exploitation de contextes et de treillis. Galicia possède un éditeur de contextes comme on peut le voir sur la figure 3.1a et aussi un outil de visualisation du treillis comme sur la figure 3.1b.

1. Une liste exhaustive des outils est disponible à l'adresse suivante : `http://www.upriss.org.uk/fca/fcasoftware.html`

2. La dernière version du logiciel est disponible à l'adresse `http://www.iro.umontreal.ca/~galicia/`



(a) Éditeur de contexte

(b) Affichage du treillis en trois dimensions

FIGURE 3.1: Capture d'écran du logiciel GALICIA

Les algorithmes qui ont été implémentés dans l'outil sont entre autres :

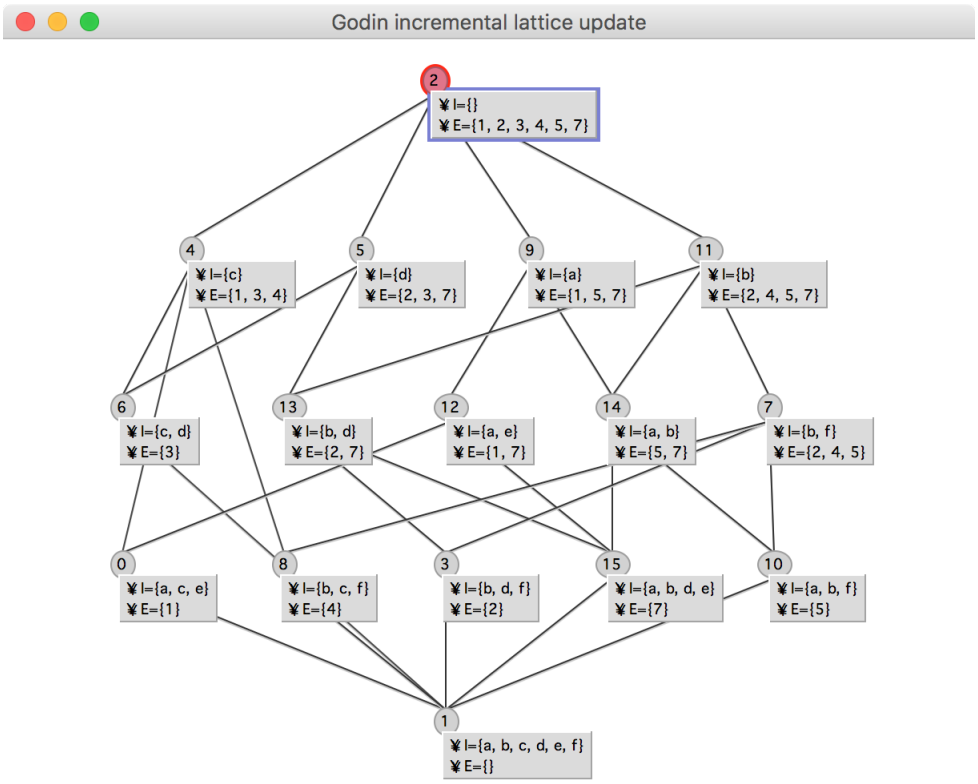
— le calcul de la base des implications de Guigues-Duquenne.

— la génération des règles d'association d'une base générique ou d'une base des règles approximatives.

— les procédures suivantes de construction du treillis : Bordat, Nourine, Godin et *Next-Closure*.

GALICIA est le premier outil à avoir intégré la gestion de contextes relationnels et la production de treillis en analyse relationnelle de concepts [32]. GALICIA offre également la possibilité d'afficher le treillis avec étiquetage complet comme on peut le voir sur la figure 3.2. L'outil a été une source d'inspiration pour les développeurs des systèmes Coron et *Lattice Miner*.

FIGURE 3.2: Treillis GALICIA avec étiquetage complet



Une autre fonctionnalité disponible dans GALICIA est la possibilité de se connecter à des bases de données SQL pour sauvegarder et manipuler les treillis de concepts.

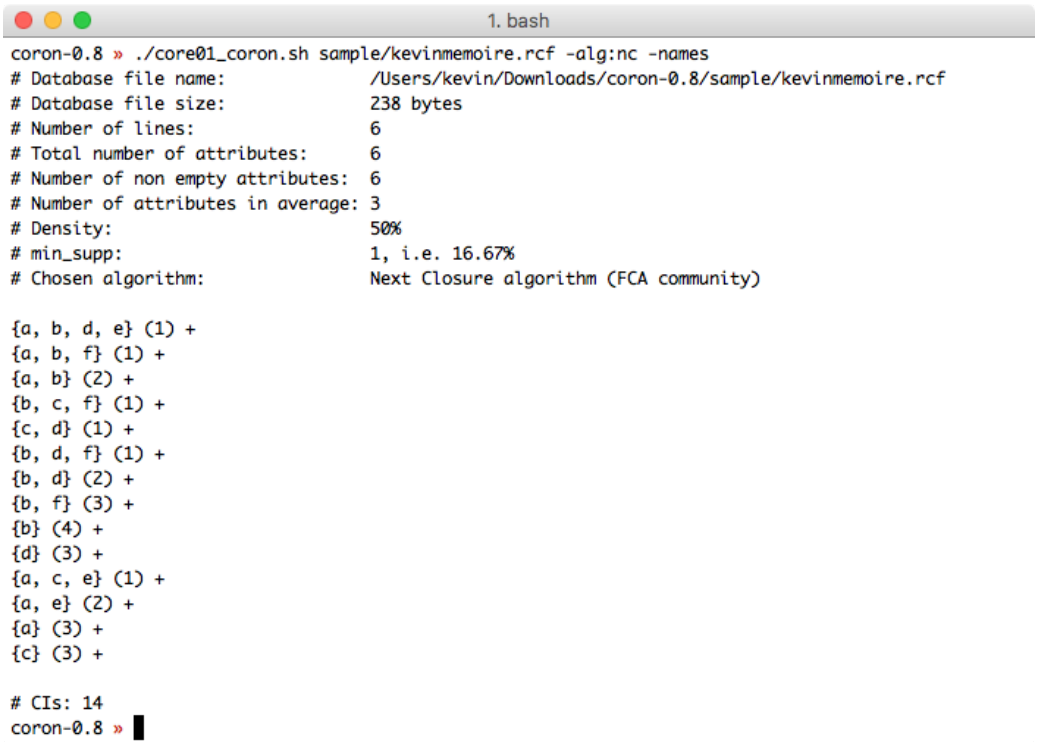
3.2 Le système Coron

Le système Coron [33] [3] est une boîte à outils très performante qui a été développée au laboratoire LORIA [4] et qui implémente plusieurs algorithmes de fouille de données tels qu'Apriori, Close, Aclose, Carpathia, Charm, Next-Closure ou NFI-Simple. Coron a été développé principalement avec le langage de programmation Java et aussi Perl.

3. Disponible à l'adresse suivante : `http://coron.loria.fr/site/index.php`.
4. Laboratoire lorrain de recherche en informatique et ses applications, Nancy, France

L'outil est compatible avec les systèmes d'exploitation Unix, macOS et Windows [5]. La version actuelle de ce système est 0.8 et date de janvier 2010.

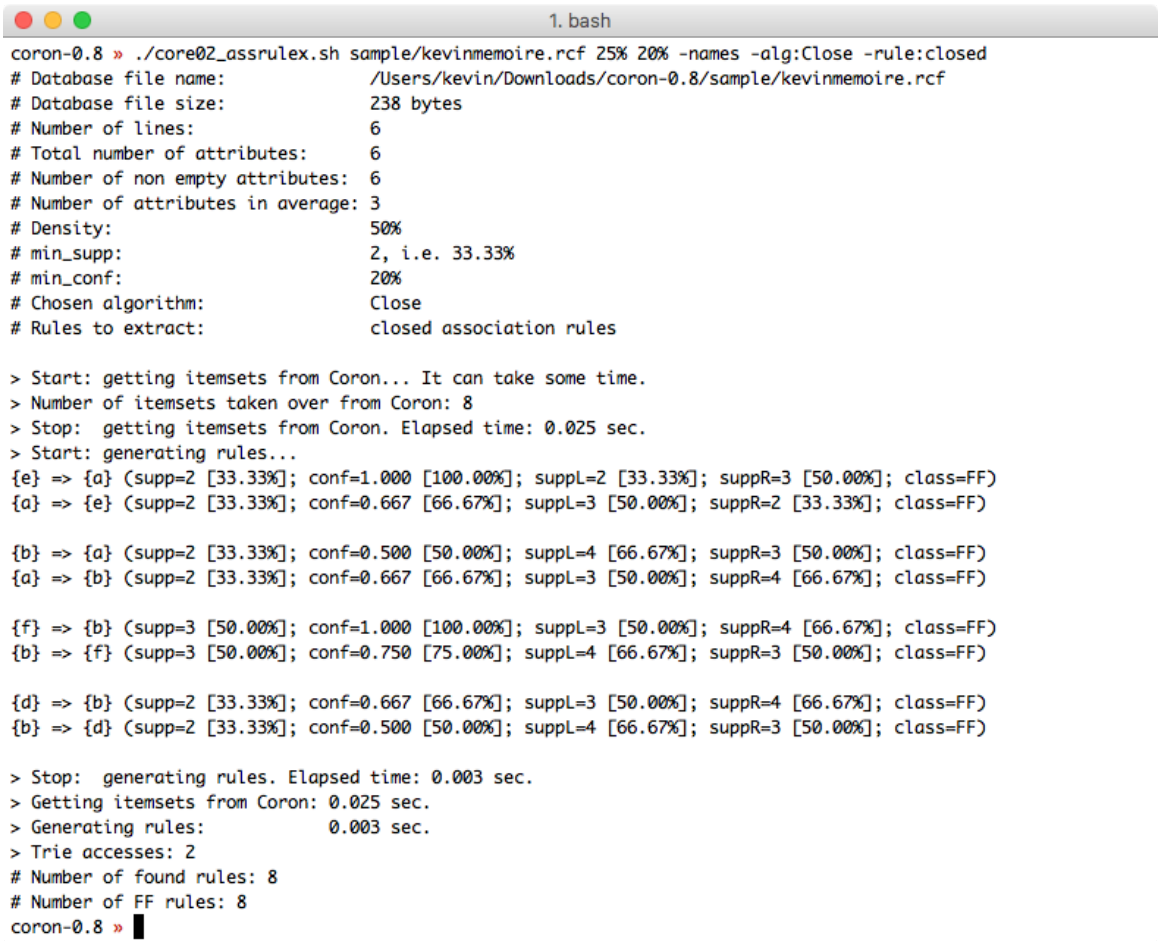
FIGURE 3.3: Calcul des itemsets fréquents et leur support dans le système Coron



Comme nous pouvons le voir sur la figure 3.3, Coron s'exécute en ligne de commandes et ne comporte pas d'interface utilisateur. La capture d'écran indique que nous avons utilisé l'algorithme *Next-Closure* sur l'exemple illustratif pour produire des itemsets fermés fréquents (14 au total) ayant un support absolu minimum de 1. Par exemple, l'itemset $\{a, e\}$ a un support absolu de 2.

5. Pour le support de Windows, on doit au préalable installer *Cygwin* (`http://www.cygwin.com/`).

FIGURE 3.4: Règles d'association avec le système Coron



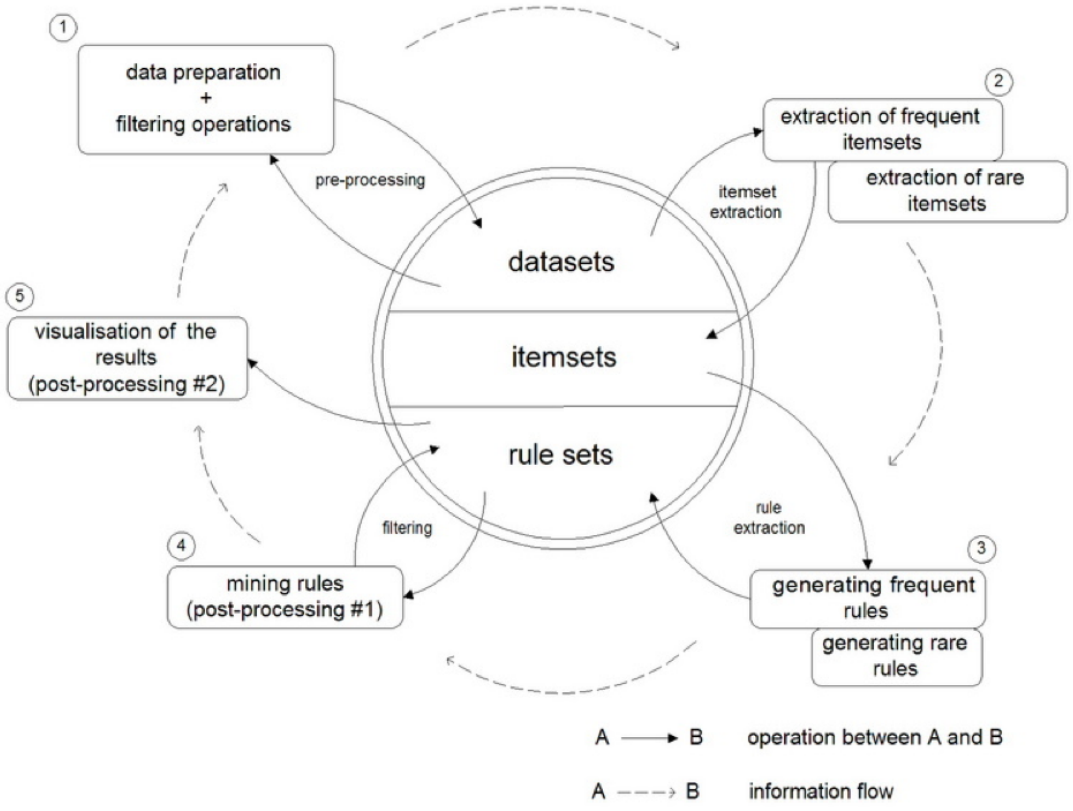
Le système Coron permet également de générer des règles d'association. On peu spécifier alors l'algorithme à exécuter sur le contexte formel ainsi que le support et la confiance minimaux désirés (voir la figure 3.4).

3.2.1 Méthodologie Coron

Les auteurs du système Coron, Kayoute *et al.* [16], ont défini une méthodologie pour l'utilisation du système Coron, avec les étapes suivantes : (1) définition du cadre de l'étude des données (2) étape itérative : nettoyage et préparation des données, étape de prétraitement, étape de traitement, étape de post-traitement, validation des résultats et

génération de nouvelles hypothèses de recherche. Ce cycle de vie de la méthodologie est défini dans la figure 3.5 tirée de [16].

FIGURE 3.5: Architecture du système Coron



3.2.2 Les modules de Coron

Coron est un système modulaire et contient les modules suivants. [6]

Les modules de base (*Core Modules*)

— *Coron-base* est le module le plus important de la plateforme Coron. Ce module permet l'extraction de différents *itemsets* comme entre autres les *itemsets fréquents*, *les itemsets fermés fréquents*, *les générateurs fréquents* et plus encore.

6. Pour plus d'informations techniques, le lecteur peut consulter la documentation disponible à l'adresse suivante : `http://coron.wikidot.com/`.

— *AssRuleX* (*Association Rule eXtractor*). Ce module est responsable de l'extraction des différentes règles d'association.
— *LeCo* (*Lattice Constructor*). Le module permet la construction du treillis sous forme de diagramme de Hasse.

Prétraitement (*Pre-Processing*)

Les modules de prétraitement permettent de formater les données volumineuses. Ces dernières sont décrites sous forme de tableaux binaires dans un format textuel. Les opérations possibles sont : (*i*) la discrétisation des données numériques, (*ii*) la conversion vers un format de fichier différent, (*iii*) la création d'un tableau binaire complémentaire, et (*iv*) d'autres opérations comme la transposition d'une matrice (table).

Post-traitement (*Post-Processing*)

L'étape de fouille de données risque de produire un volume important de résultats. Il est donc nécessaire d'effectuer un post-traitement des résultats. Le module de post-traitement offre donc deux fonctionnalités : (*i*) le filtrage des règles avec la méthode (*Filter-Rules*), et (*ii*) le coloriage des règles (*Rule-Coloring*). Cette dernière méthode prend en entrée un fichier contenant des règles d'association ainsi qu'un fichier définissant la couleur choisie pour certaines règles et il produit un fichier HTML avec les règles d'association, mais cette fois certaines règles ont une couleur spécialisée.

Outils (*Tools*)

Le module d'outils contient plusieurs fonctions qui permettent entre autres de (*i*) visualiser les classes d'équivalence (*VizEq*), (*ii*) générer automatiquement des contextes binaires à des fins de test (*ContextGen*), et (*iii*) générer des statistiques sur les classes d'équivalence (*AnalyseEqClasses*).

3.3 Librairie Java Lattices

La libraire *Java Lattices* [5] a été développée par le laboratoire L3i [7] sous l'initiative de Karell Bertet en 2014. Cette librairie permet de générer des treillis de Galois et des règles d'association. Le code source est entièrement disponible publiquement sur

7. Laboratoire Informatique, Image et Interaction, université de La Rochelle, France

GitHub[8], un service web d'hébergement de code source utilisant le logiciel de gestion de versions décentralisé *git*.

Java Lattices permet aussi de générer des treillis de Galois au format *.png (format d'image), grâce à l'outil *GraphViz*[9]. Nous avons également utilisé *Java Lattices* pour comparer les résultats de nos algorithmes. L'algorithme d'accès limité aux objets (*Limited Object Access (LOA)*) [9] est également implémenté dans cette librairie. Ce dernier algorithme permet de construire un treillis avec un accès limité aux objets, et propose une routine qui permet à partir d'un contexte $\mathbb{K}$ et d'un ensemble fermé B du treillis fermé $(\mathbb{C}_1, \subseteq)$ de $\mathbb{K}$ de trouver les successeurs immédiats de B dans le treillis.

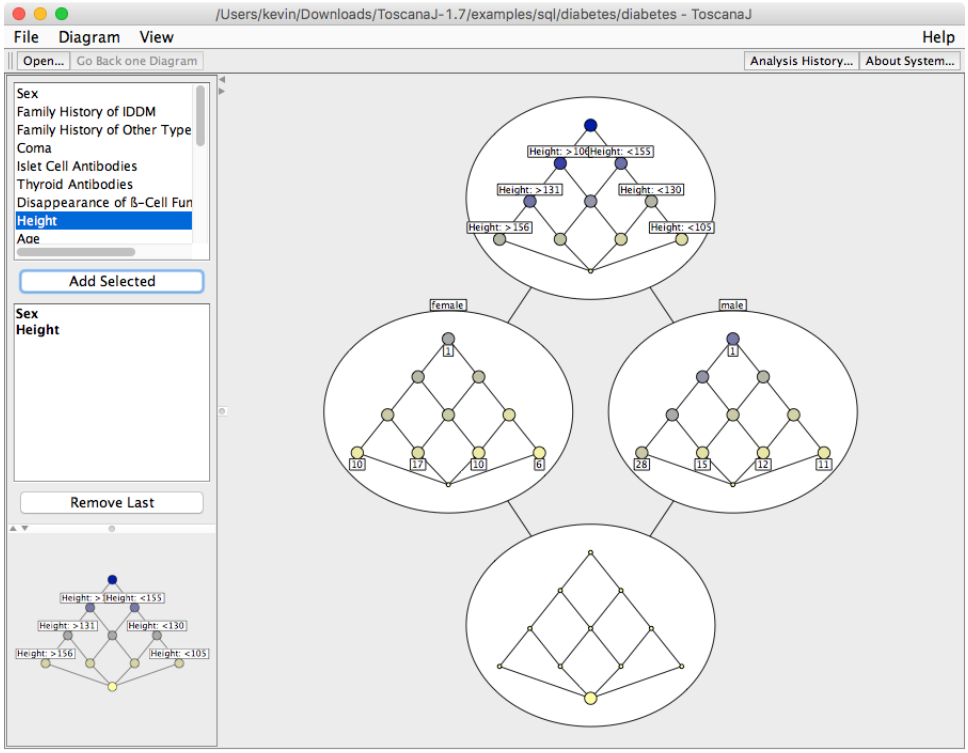
Pour utiliser *Java Lattices*, on peut soit (i) créer son propre client et faire appel à la librairie en utilisant Maven (outil Java pour gérer les dépendances), ou bien (ii) on peut utiliser le projet *java-lattice-bin*[10]. Ce dernier projet est un client en ligne de commandes qui permet d'exécuter quelques méthodes de la librairie *Java Lattices* telles qu'entre autres la génération des *itemsets* fermés avec l'algorithme *Next-Closure* [11] ou Bordat [7], des générateurs minimaux [25] ou les graphes de dépendances.

3.4 ToscanaJ

ToscanaJ [4] est une réimplémentation avec le langage de programmation Java de l'outil d'analyse formelle de concepts connu sous le nom de « *Toscana* ». Cet outil a la particularité d'être le premier outil à offrir l'affichage des diagrammes imbriqués (*nested line diagrams*) [13] comme on peut le voir avec la figure 3.6. Toutefois, *ToscanaJ* n'a pas de module de création de contexte ou de treillis. Pour cela, il faut utiliser *Siena* qui est un outil distribué avec *ToscanaJ*. Ce dernier s'intègre très bien avec les bases de données, par utilisation de pilote JDBC (*Java Database Connectivity*) et l'édition de quelques fichiers de configuration.

8. `https://thegalactic.github.io/java-lattices`
9. Outil disponible à l'adresse suivante : `http://www.GraphViz.org`
10. Voir `https://github.com/thegalactic/java-lattices-bin`

FIGURE 3.6: ToscanaJ : Diagrammes imbriqués

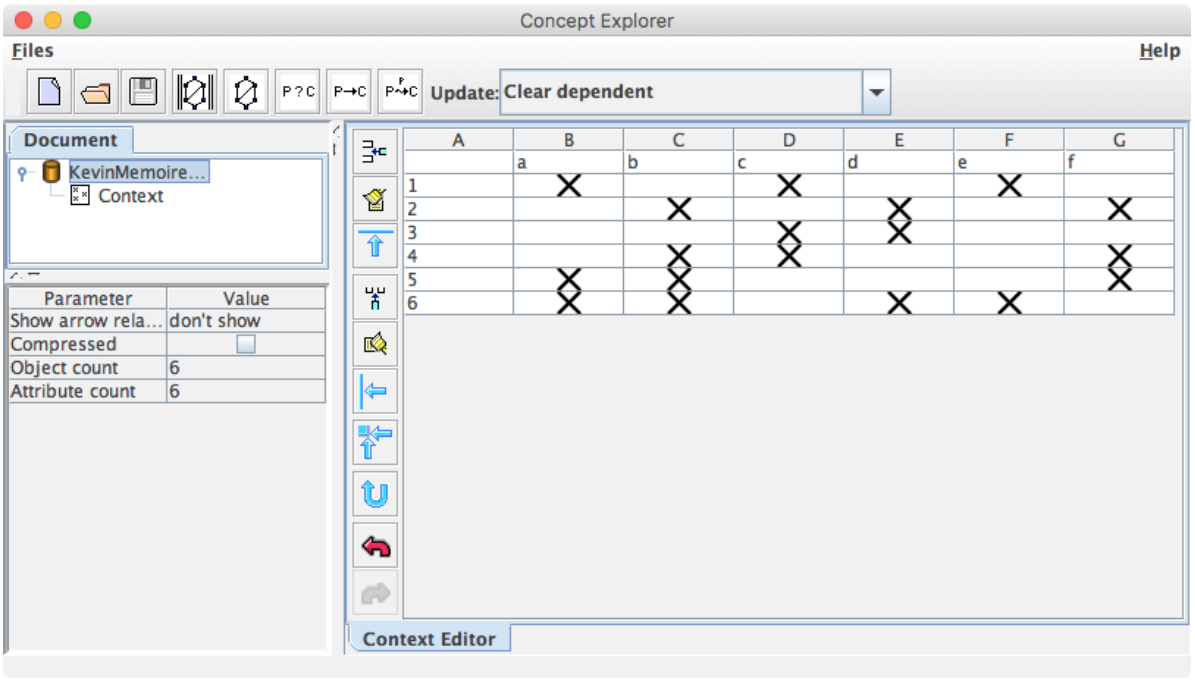


3.5 *Concept Explorer*

Concept Explorer[11] (*ConExp*) est un outil souvent utilisé par les chercheurs en AFC. Il a été initié dans le cadre d'un mémoire de maîtrise sous la supervision du Professeur Tatyana Taran en 2000. Il permet entre autres l'édition et la transformation de contextes, la construction de treillis de concepts, la recherche d'une base d'implications de Guigues-Duquenne [3], la génération des règles d'association et l'exploration des attributs. Il a été développé avec le langage de programmation Java et son code source est en distribution libre.

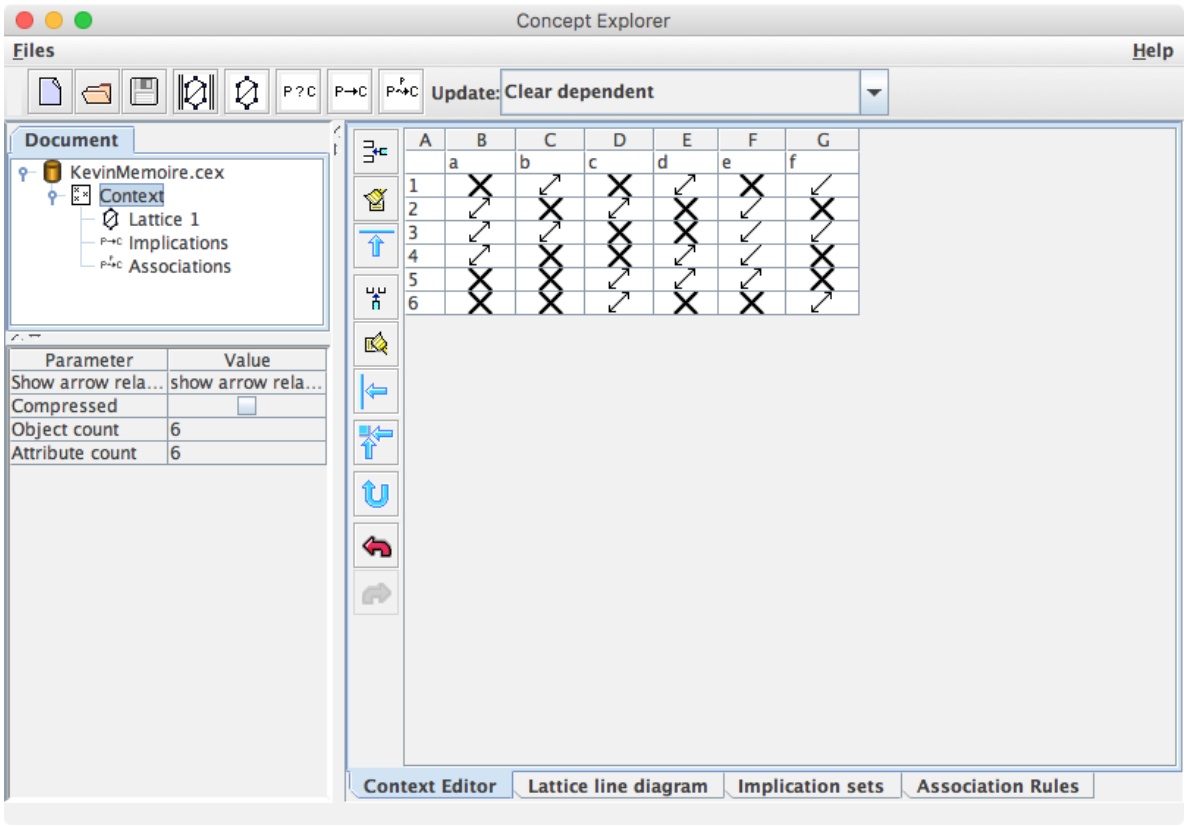
11. Disponible à l'adresse suivante : `http://conexp.sourceforge.net/`

FIGURE 3.7: *ConExp* : Menu du contexte



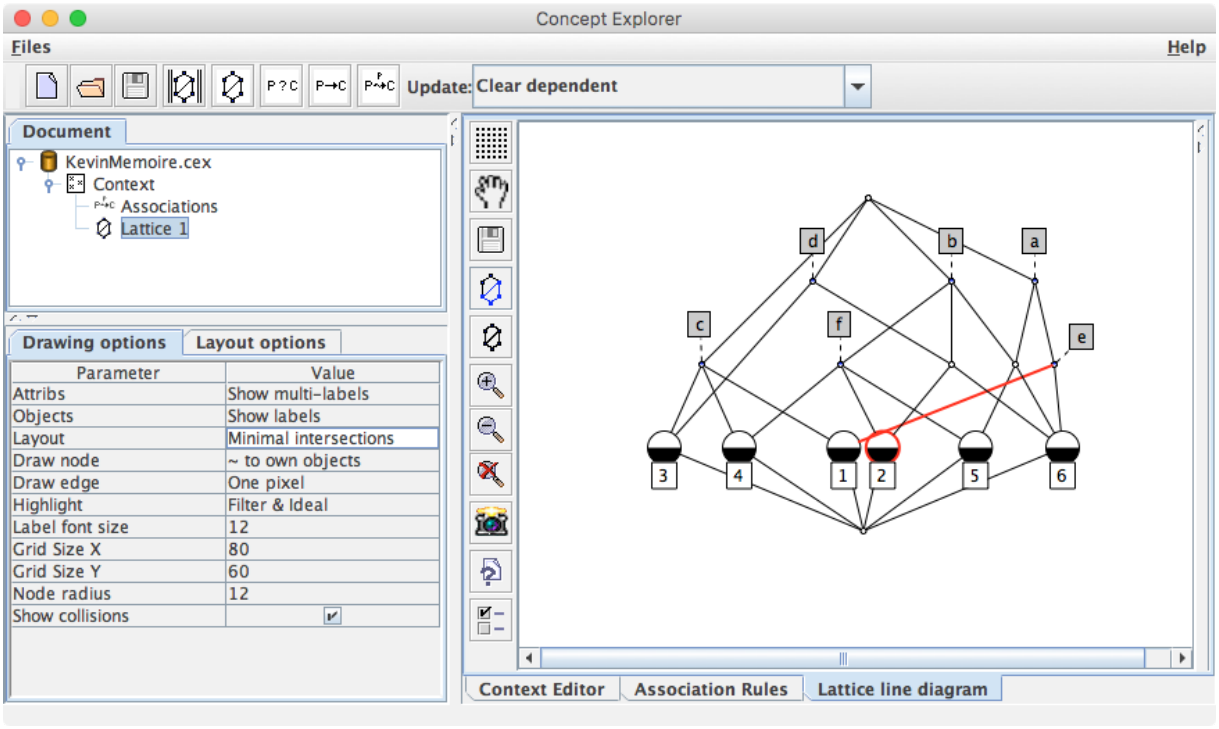
Le menu d'édition du contexte permet la création, la modification et la sauvegarde des contextes au format CEX propre à *ConExp*.

FIGURE 3.8: *ConExp* : Menu du contexte avec les flèches



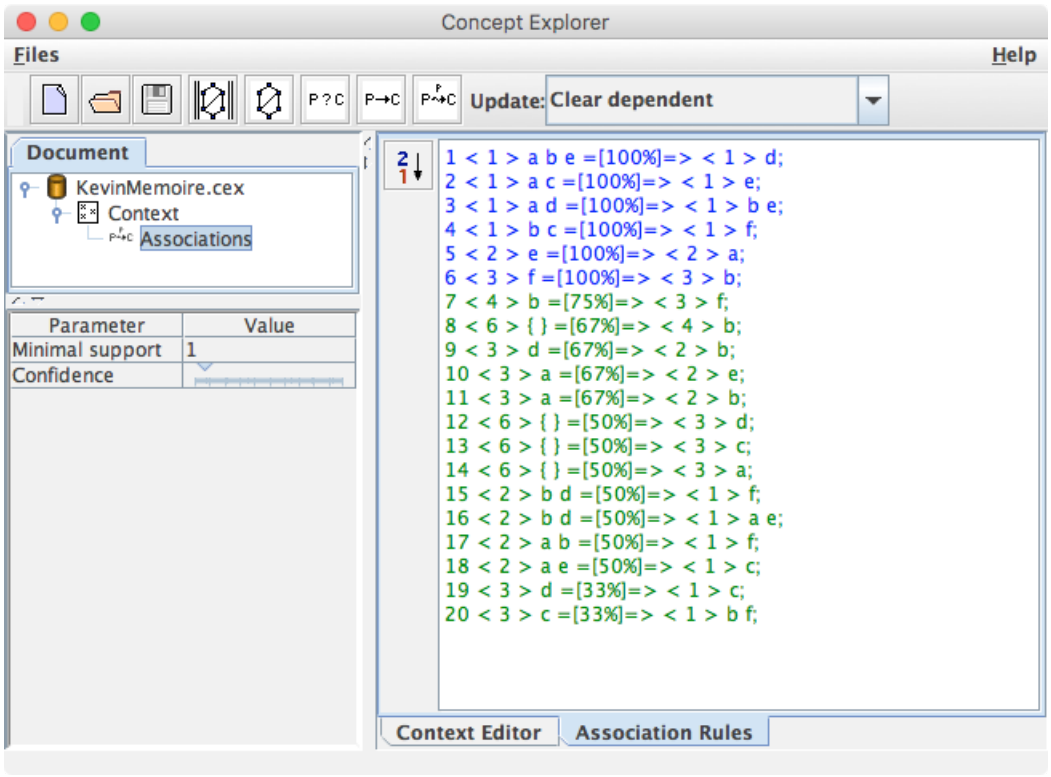
À partir du menu du contexte, nous pouvons également générer les relations flèches (figure 3.8). *ConExp* propose aussi plusieurs fonctionnalités qui sont intéressantes tels que la projection sur les objets et les attributs du contexte.

FIGURE 3.9: *ConExp* : Menu du treillis



Dans la figure 3.9, on remarque que l'affichage du treillis se fait selon un étiquetage réduit seulement (cf. section 2.1). Par défaut, *ConExp* utilise un algorithme de construction du treillis avec le minimum d'intersections et les collisions sont affichées en rouge. Toutefois, l'utilisateur peut aisément manipuler le treillis de concepts et déplacer ses éléments pour rendre l'affichage plus lisible.

FIGURE 3.10: *ConExp* : Règles d'association



ConExp offre également la possibilité de générer la base de Guigues-Duquenne et de calculer les règles d'association. Prenons pour exemple la ligne suivante de la figure 3.10 :

`7 < 4 > b =[75%]=> < 3 > f;`

Il s'agit de la 7-ème règle d'association laquelle indique que b implique f avec un support absolu de 3 parmi les six objets (et donc un support relatif de 50%) et une confiance de 75%.

La sauvegarde du contexte et de son treillis en format XML est également disponible comme pour *Galicia*, *Lattice Miner* et *Coron*. Ce même formalisme est utilisé dans *Lattice Miner* pour sauvegarder les règles d'association, y compris les implications. Dans le cadre de ce mémoire, nous allons utiliser *ConExp* à des fins de validation, notamment pour tester la procédure de production des relations flèches et pour la génération de la base de Guigues-Duquenne.

3.6 *Lattice Miner*

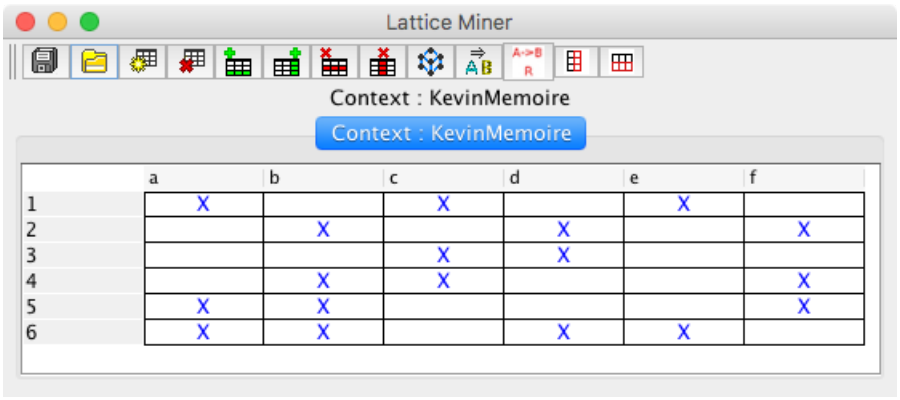
Lattice Miner est un logiciel en Java pour la visualisation et la manipulation de treillis de concepts développé au laboratoire LARIM de l'Université du Québec en Outaouais sous la supervision du Professeure Rokia Missaoui. Il a été initialement conçu par Geneviève Roberge [30] dans le cadre de son mémoire de maîtrise puis continuellement enrichi par les membres de l'équipe et plusieurs stagiaires ingénieurs français. Aussi bien dans sa version initiale que dans sa version révisée, *Lattice Miner* est un logiciel libre disponible sur *SourceForge*[12] et *GitHub*[13].

Comme les treillis de concepts sont souvent volumineux même pour un contexte de taille relativement modeste, le but ultime de *Lattice Miner* est de rendre plus conviviales l'utilisation et la visualisation des treillis de concepts en mettant en relief certains nœuds que l'utilisateur estime utiles pour son analyse et son exploration.

Le logiciel permet d'entrer des données sous forme d'un contexte binaire. Des contextes binaires séparés en niveaux peuvent aussi être créés afin de produire des treillis imbriqués.

Comme *Lattice Miner* est au cœur de notre travail de recherche, nous illustrons son utilisation par un simple exemple.

FIGURE 3.11: Création d'un contexte binaire avec *Lattice Miner*



	a	b	c	d	e	f
1	X		X		X	
2		X		X		X
3			X	X		
4		X	X			X
5	X	X				X
6	X	X		X	X	

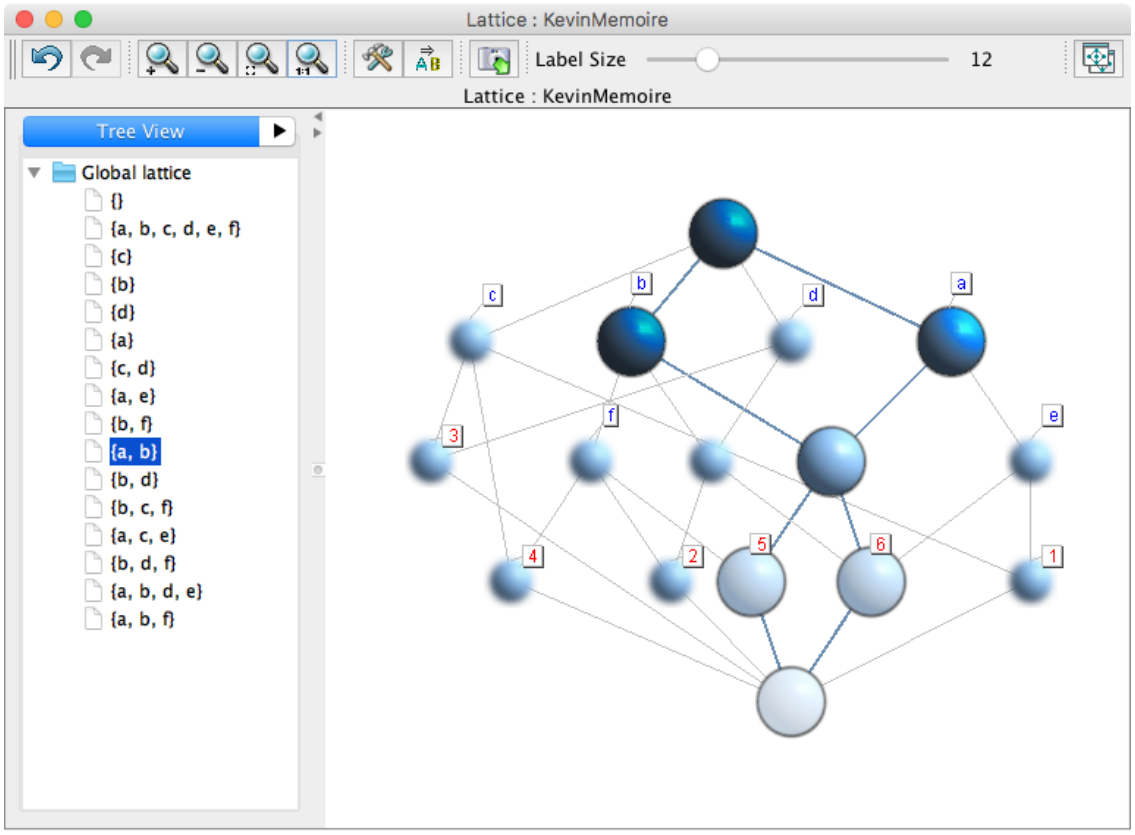
Il est possible non seulement de créer un nouveau contexte, mais également d'importer un contexte existant avec le format de fichier *.lmb de *Lattice Miner*, *.cex de

12. https://sourceforge.net/projects/lattice-miner/
13. https://github.com/LarimUQO/lattice-miner

Concept Explorer [26] ou bien *.slf de GALICIA [34]. Dans l'exemple de la figure 3.11, le contexte est composé de 6 objets et de 6 attributs lesquels sont nommés respectivement : a, b, c, d, e et f. Après avoir entré les informations du contexte, *Lattice Miner* crée un tableau qu'on peut remplir en cliquant sur chaque cellule.

Pour générer le treillis, il suffit de cliquer sur l'icône  . L'outil ouvre alors une nouvelle interface avec le treillis dessiné graphiquement comme dans la figure ci-dessous :

FIGURE 3.12: Génération du treillis de concepts dans *Lattice Miner*



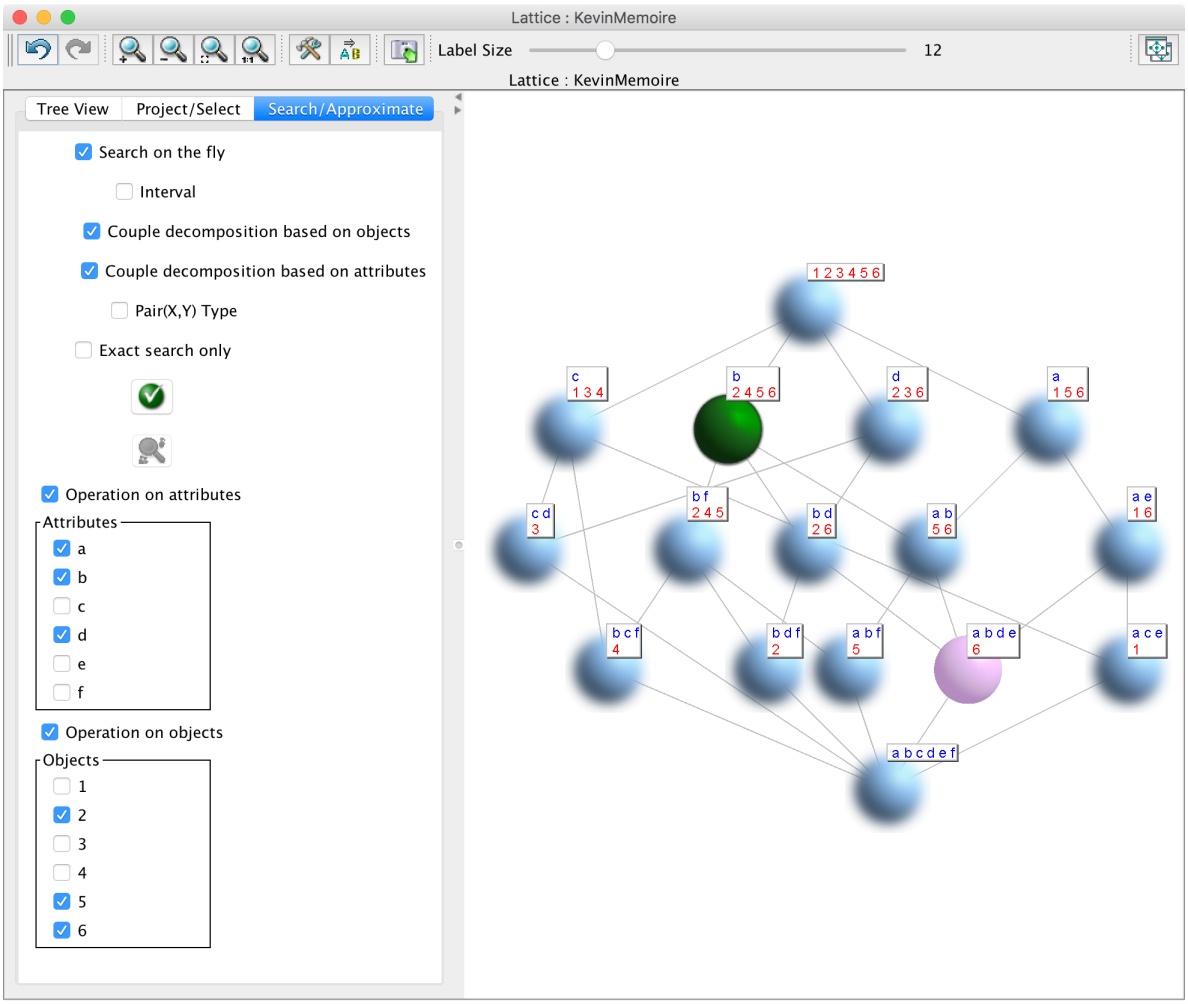
Comme nous l'avons mentionné, l'un des avantages de *Lattice Miner* est son outil de visualisation des treillis qui permet de parcourir le treillis, sélectionner certains nœuds ou certaines sous-structures. Sur la figure 3.12, nous avons sélectionné sur le panneau de gauche l'intention $\{a, b\}$ pour retrouver dans le treillis à droite le nœud (concept) correspondant ainsi que son filtre (partie supérieure) et son idéal (partie inférieure) de sorte que les concepts pertinents restent nets alors que les autres concepts apparaissent

flous. D'autres outils, comme *ConExp*, permettent également de trouver le filtre et l'idéal d'un nœud.

L'utilisateur a la possibilité d'étiqueter les nœuds du treillis sous forme *complète* ou *réduite*. Il peut également afficher les générateurs minimaux associés aux intentions des concepts. Aussi, il est important de noter que le treillis que nous voyons dans la figure 3.12 est dit *réduit* car il montre les objets et les attributs une seule fois dans les noeuds. À titre d'exemple, le nœud dont le label est a représente en fait le concept formel $(156, a)$ indiquant que les objets 1, 5 et 6 ont en commun l'attribut a.

Au niveau de ce module de concepts, nous avons accès à des opérations de base comme la projection sur les attributs, la sélection d'objets, une recherche exacte de concepts ou bien encore l'approximation d'une paire d'objets et d'attributs tels que montrés par la figure 3.13. En effet, la capture d'écran indique que la paire $(256, abd)$ n'est pas un concept, mais se trouve dans le voisinage délimité par les deux concepts formels $(6, abde)$ et $(2456, b)$.

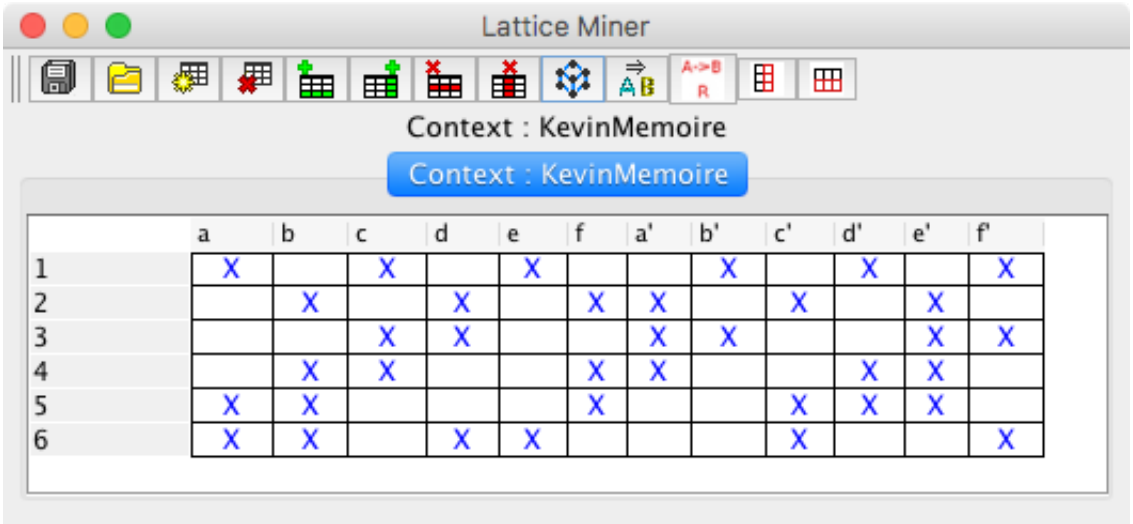
FIGURE 3.13: Approximation dans *Lattice Miner*



Les algorithmes implémentés dans *Lattice Miner* pour la génération du treillis de concepts sont entre autres la procédure de Bordat, l'algorithme de Godin ou bien encore Next-Closure [11]. Les treillis obtenus peuvent être sauvegardés en format XML ou JPG.

À l'aide de *Lattice Miner*, il est également possible de générer le complémentaire d'un contexte et de produire l'apposition d'un contexte avec son complémentaire tel qu'illustré par la figure 3.14 où l'attribut a' est la négation de a, ce qui veut dire que tout objet ayant a n'a pas a' et inversement, tout objet n'ayant pas a possède a'.

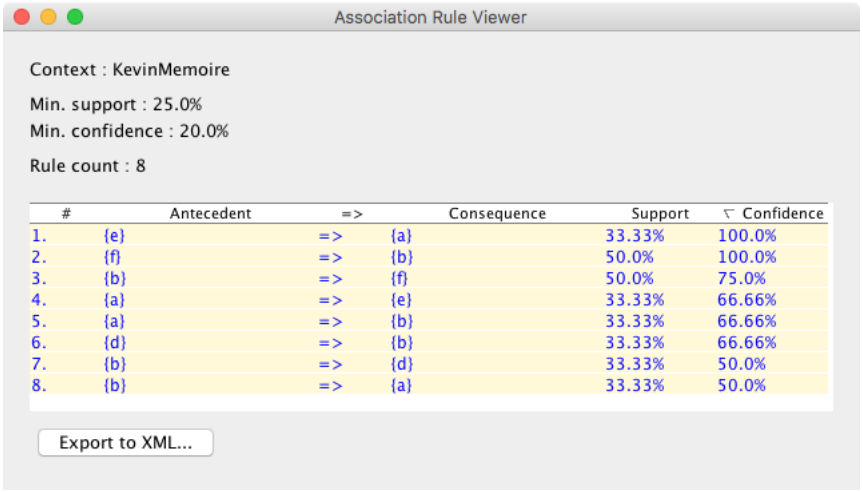
FIGURE 3.14: Affichage de l'apposition d'un contexte avec son complémentaire



	a	b	c	d	e	f	a'	b'	c'	d'	e'	f'
1	X		X		X			X		X		X
2		X		X		X	X		X		X	
3			X	X			X	X			X	X
4		X	X			X	X			X	X	
5	X	X				X			X	X	X	
6	X	X		X	X				X			X

En cliquant sur l'icône  , nous pouvons afficher la base des règles d'association. Pour cela, l'outil demande la valeur minimale du support et celle de la confiance. Il est également possible d'afficher la base générique des implications avec ou sans redondance (cf. figures 3.16 et 3.17). Dans le deuxième cas, il suffit de cliquer sur l'icône  pour produire une couverture minimale de la base d'implications.

FIGURE 3.15: Génération des règles d'association dans *Lattice Miner*



#	Antecedent	=>	Consequence	Support	Confidence
1.	{e}	=>	{a}	33.33%	100.0%
2.	{f}	=>	{b}	50.0%	100.0%
3.	{b}	=>	{f}	50.0%	75.0%
4.	{a}	=>	{e}	33.33%	66.66%
5.	{a}	=>	{b}	33.33%	66.66%
6.	{d}	=>	{b}	33.33%	66.66%
7.	{b}	=>	{d}	33.33%	50.0%
8.	{b}	=>	{a}	33.33%	50.0%

FIGURE 3.16: Base générique d'implications avec possibilité de redondance

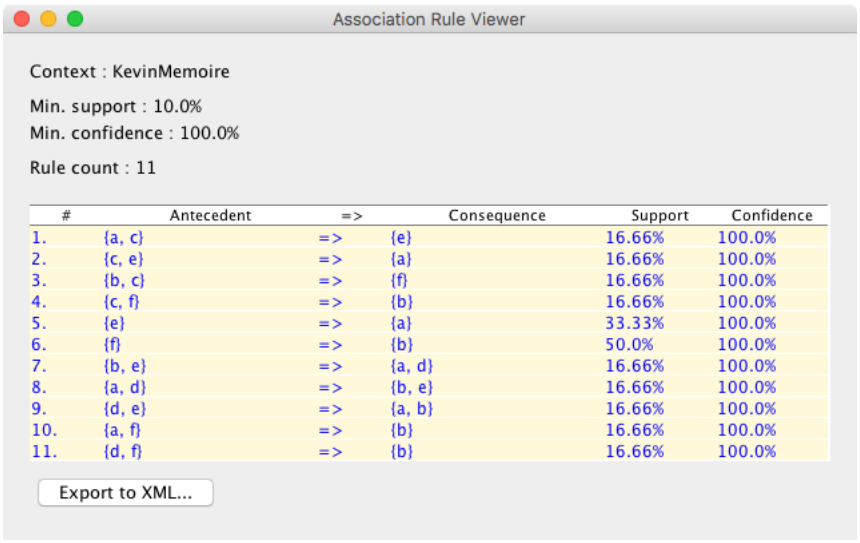
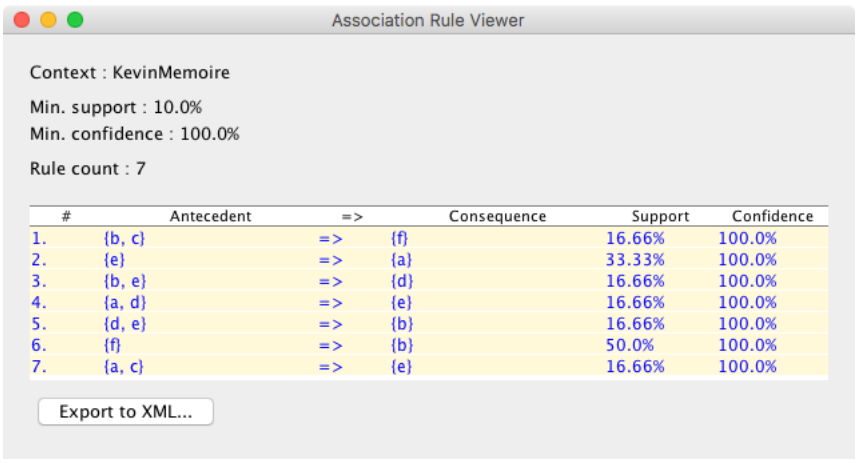


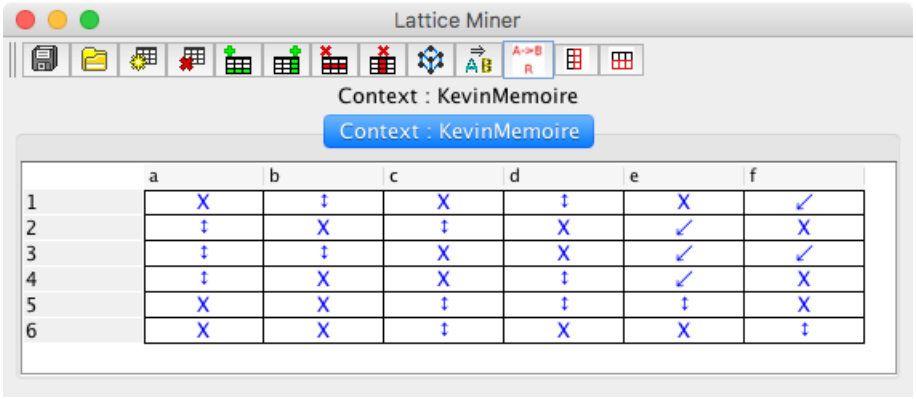
FIGURE 3.17: Base générique d'implications sans redondance



Pour la génération de la base de Guigues–Duquenne, on utilise l'algorithme *Next-Closure* qui permet d'obtenir à la fois les concepts et les *pseudo-intents* [11] associés à un contexte et ainsi générer les implications de cette base.

Les résultats des divers types de bases sont affichés sur une fenêtre et peuvent être exportés dans un document XML d'une manière similaire à la sauvegarde des treillis de concepts.

FIGURE 3.18: Relations flèches dans *Lattice Miner*



	a	b	c	d	e	f
1	X	↕	X	↕	X	↙
2	↕	X	↕	X	↙	X
3	↕	↕	X	X	↙	↙
4	↕	X	X	↕	↙	X
5	X	X	↕	↕	↕	X
6	X	X	↕	X	X	↕

Une autre fonctionnalité à ne pas oublier est la possibilité de produire les relations flèches sur un contexte donné. Par exemple, sur la figure 3.18, nous pouvons voir que nous obtenons exactement le même résultat qu'avec *ConExp* (voir figure 3.8).

Dans le cadre de ce mémoire, nous avons testé intensivement sur *Lattice Miner* les relations flèches ainsi que la base de Guigues–Duquenne sur différents contextes et nous avons comparé les résultats obtenus avec *ConExp*.

Chapitre 4

Production d'implications avec négation

Dans ce chapitre, nous allons discuter de l'introduction de la négation dans les implications et de l'implémentation d'une procédure de calcul de ces implications dans *Lattice Miner* lesquelles permettent de définir des règles conflictuelles du type : *« Si l'on achète du caviar, alors on n'achète pas du thon »*. L'un des intérêts des implications (et d'une manière générale des règles d'association) avec négation est de pouvoir déceler des exceptions (ex. tous les oiseaux volent sauf l'autruche).

4.1 Définitions et notations

Définition 4.1. Soit une implication $Y \rightarrow Z$ [*sup*] dans un contexte $\mathbb{K}|\tilde{\mathbb{K}}$ tel que Y et Z sont des sous-ensembles de $M \cup \tilde{M}$ et $Y \cap Z = \emptyset$. Les implications appartiennent à une des catégories suivantes : implications *purement négatives* avec $M \cap (Y \cup Z) = \emptyset$, implications *purement positives* avec $\tilde{M} \cap (Y \cup Z) = \emptyset$ et implications *mixtes* tel que $M \cap (Y \cup Z) \neq \emptyset$ et $\tilde{M} \cap (Y \cup Z) \neq \emptyset$.

Définition 4.2. On définit les implications basées sur les clés (*key-based implication*) du contexte $\mathbb{K}|\tilde{\mathbb{K}}$ de la manière suivante : $Y \rightarrow M \cup \tilde{M} \backslash Y$ [0] avec Y une clé dans le contexte $\mathbb{K}|\tilde{\mathbb{K}}$, ce qui signifie que Y implique tous les attributs dans $M \cup \tilde{M}$. Le support d'une telle implication est évidemment nul car il n'existe aucun itemset Y impliquant tous les attributs dans $M \cup \tilde{M}$.

Les implications avec négation peuvent être sous l'une des formes suivantes : (i) $B \wedge \tilde{C} \rightarrow D$ indiquant que si un objet possède les propriétés contenues dans l'ensemble B sans avoir celles dans C (c.-à-d. qu'il a les attributs contenus dans la négation de C notée $\tilde{C}$), alors il possède toutes les propriétés contenues dans D, (ii) $B \rightarrow C \wedge \tilde{D}$ signifie que si un objet a toutes les propriétés dans B, alors il possède tous les attributs dans C sans avoir aucun des éléments dans D, et (iii) $\tilde{B} \rightarrow \tilde{C}$ signifie que si un objet n'a aucune propriété dans B, alors il ne possède aucun attribut dans C. Les deux premiers cas correspondent à des implications mixtes alors que le dernier cas représente des implications purement négatives.

Nous allons expliquer et illustrer la méthode naïve puis celle que nous avons retenue à partir d'un exemple concret.

4.2 Méthode naïve

Une méthode naïve mais non efficace de détermination des implications avec négation consiste d'abord à effectuer l'apposition du contexte $\mathbb{K}$avec son complémentaire $\tilde{\mathbb{K}}$ pour obtenir le treillis $\underline{\mathfrak{B}}(\mathbb{K}|\tilde{\mathbb{K}})$ et ensuite d'extraire les implications de ce treillis. Bien qu'assez simple, cette méthode va générer un treillis trop volumineux même si le contexte initial est épars puisque son complémentaire devient dense. Partant d'un contexte initial $\mathbb{K}$, la première étape de la méthode naïve consiste à effectuer l'apposition de $\mathbb{K}$ avec son complémentaire $\tilde{\mathbb{K}}$. On obtient :

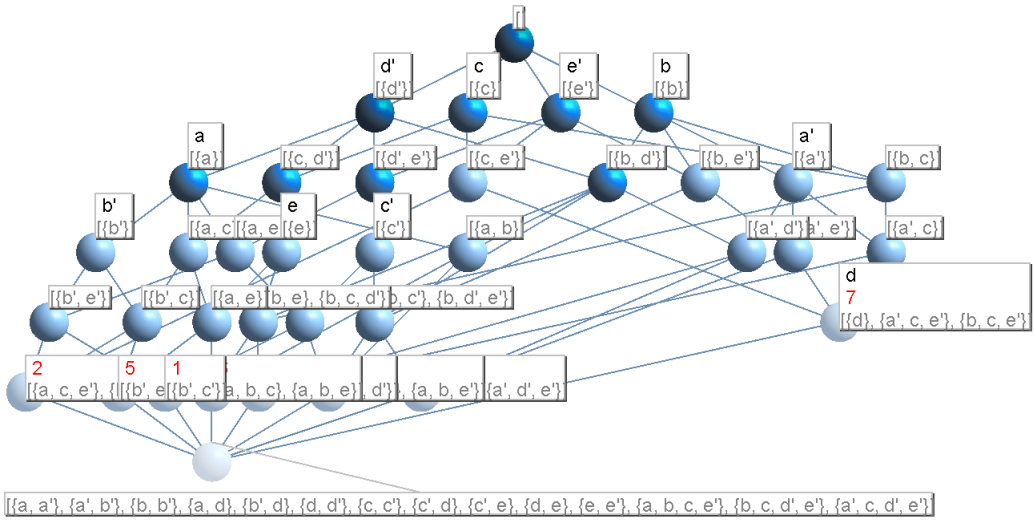
FIGURE 4.1: $\mathbb{K}|\tilde{\mathbb{K}}$: apposition du contexte $\mathbb{K}$ avec son complémentaire $\tilde{\mathbb{K}}$

	a	b	c	d	e	$\tilde{a}$	$\tilde{b}$	$\tilde{c}$	$\tilde{d}$	$\tilde{e}$
1	×						×	×	×	×
2	×		×				×		×	×
3	×	×						×	×	×
4		×				×		×	×	×
5	×		×		×		×		×	
6	×	×	×		×				×	
7		×	×	×		×				×
8		×	×		×	×			×	

Ensuite, on calcule la base d'implications à partir du treillis $\underline{\mathfrak{B}}(\mathbb{K}|\tilde{\mathbb{K}})$ obtenu (cf. 4.2).

Lattice Miner.

FIGURE 4.2: Génération du treillis pour le contexte $\mathbb{K}|\tilde{\mathbb{K}}$



4.3 Solution plus performante

Nous allons implémenter une nouvelle solution plus efficace qui détermine uniquement les co-atomes (successeurs de l'infimum) du treillis $\underline{\mathfrak{B}}(\mathbb{K}|\tilde{\mathbb{K}})$ sans toutefois construire ce dernier ou son contexte pour ensuite calculer les clés (générateurs minimaux de l'infimum) qui servent à déterminer l'ensemble des implications avec négation. Pour cela, nous utilisons un théorème tiré de Missaoui *et al.* [24] qui calcule les clés du contexte $\mathbb{K}|\tilde{\mathbb{K}}$. Nous rappelons que la clé d'une table relationnelle ou d'un contexte formel est un ensemble d'attributs permettant de déterminer le reste des attributs. Le calcul des clés a fait l'objet de plusieurs algorithmes aussi bien dans le domaine des bases de données que dans d'autres domaines en informatique (cf. [10, 21] pour plus de détails).

Théorème 1. *Soit Ax avec x un attribut de $M\tilde{M}$ et $A \subseteq M\tilde{M}$. Alors, $Ax \rightarrow M\tilde{M}\backslash Ax$ [0] $\Leftrightarrow$ $A \rightarrow \tilde{x}$ [sup] avec $sup = |A'|/|G|$ où A' est l'ensemble d'objets possédant les attributs contenus dans A et G est l'ensemble des objets du contexte $\mathbb{K}$.*

Il a été démontré que le théorème 1 représente un système d'inférence complet et valide qui permet de générer toutes les implications avec négation à partir des implications dont la prémisse est une clé. Ces implications avec clé de la forme $Ax \rightarrow M\tilde{M}\backslash Ax$

ont un support nécessairement nul puisque même si un objet possède tous les attributs dans M, il n'en possède aucun dans $\tilde{M}$.

En partant du contexte initial $\mathbb{K}$ sans toutefois produire le contexte $\mathbb{K}|\tilde{\mathbb{K}}$ et son treillis, on se base sur le théorème précédent pour déterminer d'abord les clés du $\mathbb{K}|\tilde{\mathbb{K}}$ pour ensuite exploiter ces clés pour générer les implications avec négation. La détermination des clés se fait par le biais d'un algorithme de calcul des générateurs minimaux au niveau de l'infimum $\perp$ du treillis (non construit) relatif au contexte $\mathbb{K}|\tilde{\mathbb{K}}$. Pour cela, nous avons d'abord besoin de déterminer les co-atomes (successeurs immédiats de l'infimum) du treillis et appliquer une procédure de calcul des générateurs de l'infimum. L'algorithme 1 part du contexte initial $\mathbb{K}$ et détermine les concepts objets qui sont tous les uniques co-atomes du treillis du contexte $\mathbb{K}|\tilde{\mathbb{K}}$ (lignes 6 à 13). L'intention Y de chaque co-atome inclut aussi bien une partie positive dans M (attributs possédés) et une partie négative (attributs non possédés). La procédure JEN [18] ou toute autre algorithme similaire (ligne 14) permet de calculer les générateurs de l'intention d'un nœud du treillis en fonction de ses successeurs.

Algorithme 1 : Calcul des générateurs de l'infimum du treillis du contexte $\mathbb{K}|\tilde{\mathbb{K}}$

1: **Procédure**
2: **Entrée :** Contexte initial $\mathbb{K} := (G, M, I)$
3: **Sortie :** $\mathcal{K}$: Les générateurs de l'infimum du treillis du contexte $\mathbb{K}|\tilde{\mathbb{K}}$
4: $INT \leftarrow \emptyset$ {Ensemble des intentions des co-atomes}
5: $O \leftarrow \emptyset$ {Objets traités}
6: **while** $G \neq O$ **do**
7: **for all** $o \in G$ mais $o \notin O$ **do**
8: $X \leftarrow o''$ {Calcul du fermé de l'objet o}
9: $Y \leftarrow Intent(o) \cup (M \setminus Intent(o))^{\sim}$
10: $O := O \cup X$
11: $INT := INT \cup \{Y\}$
12: **end for**
13: **end while**
14: $\mathcal{K} \leftarrow \text{JEN}(M \cup \tilde{M}, INT)$
15: **return** $\mathcal{K}$

Une fois que l'on obtient l'ensemble des clés Ax dans les implications de la forme : $Ax \rightarrow M\tilde{M} \backslash Ax$ [0]), il suffit d'appliquer le théorème 1 pour déplacer (après négation) un élément à la fois de Ax vers la droite d'une nouvelle implication de la forme : $A \rightarrow \tilde{x}$ [*sup*].

Lorsque l'on utilise l'algorithme de calcul des générateurs (déjà implémenté dans *Lattice Miner*) uniquement au niveau de l'infimum (avec référence aux co-atomes), on obtient les clés suivantes où par exemple a' (notation équivalente à $\tilde{a}$) représente la négation de a :
$\{a, a'\}$, $\{a', b'\}$, $\{b, b'\}$, $\{a, d\}$, $\{b', d\}$, $\{d, d'\}$, $\{c, c'\}$, $\{c', d\}$, $\{c', e\}$, $\{d, e\}$, $\{e, e'\}$, $\{a, b, c, e'\}$, $\{b, c, d', e'\}$, $\{a', c, d', e'\}$.
Après avoir écarté les clés triviales représentant un attribut et sa négation (ex. $\{a, a'\}$), on retient les éléments suivants :
$\{a', b'\}$, $\{a, d\}$, $\{b', d\}$, $\{c', d\}$, $\{c', e\}$, $\{d, e\}$, $\{a, b, c, e'\}$, $\{b, c, d', e'\}$, $\{a', c, d', e'\}$.

On constante que l'on a obtenu le même résultat au niveau des générateurs de l'infimum du treillis de la figure 4.2 sans avoir construit un tel treillis.

4.3.1 Calcul des implications avec négation à partir des clés obtenues

Tel qu'indiqué dans le théorème 1, le calcul d'une implication avec éventuellement des attributs négatifs est fait en déplaçant après négation un attribut de la clé candidate vers la conclusion et en gardant le reste des attributs dans la prémisse. Il y a autant de nouvelles implications avec négation que d'attributs dans une clé. Le tableau 4.1 présente les implications obtenues. L'algorithme 1 contenu dans [24] explique la manière de produire de telles implications à partir des clés.

Tableau 4.1: Création des implications avec négation à partir des clés candidates

Candidat	**Règle(s)**
$\{a', b'\}$	$\{b'\} \rightarrow \{a\}$ $\{a'\} \rightarrow \{b\}$
$\{a, d\}$	$\{d\} \rightarrow \{a'\}$ $\{a\} \rightarrow \{d'\}$
$\{b', d\}$	$\{d\} \rightarrow \{b\}$ $\{b'\} \rightarrow \{d'\}$
$\{c', d\}$	$\{d\} \rightarrow \{c\}$ $\{c'\} \rightarrow \{d'\}$
$\{c', e\}$	$\{e\} \rightarrow \{c\}$ $\{c'\} \rightarrow \{e'\}$
$\{d, e\}$	$\{e\} \rightarrow \{d'\}$ $\{d\} \rightarrow \{e'\}$
$\{a, b, c, e'\}$	$\{b, c, e'\} \rightarrow \{a'\}$ $\{a, c, e'\} \rightarrow \{b'\}$ $\{a, b, e'\} \rightarrow \{c'\}$ $\{a, b, c\} \rightarrow \{e\}$
$\{b, c, d', e'\}$	$\{c, d', e'\} \rightarrow \{b'\}$ $\{b, d', e'\} \rightarrow \{c'\}$ $\{b, c, e'\} \rightarrow \{d\}$ $\{b, c, d'\} \rightarrow \{e\}$
$\{a', c, d', e'\}$	$\{c, d', e'\} \rightarrow \{a\}$ $\{a', d', e'\} \rightarrow \{c'\}$ $\{a', c, e'\} \rightarrow \{d\}$ $\{a', c, d'\} \rightarrow \{e\}$

4.3.2 Calcul du support des implications

Le calcul du support de chaque implication $Y \rightarrow Z$ peut se faire en consultant le contexte $\mathbb{K}|\tilde{\mathbb{K}}$ et en calculant la proportion des objets ayant tous les éléments simulta-

nément de la prémisse Y et de la conclusion Z. Pour éviter le calcul de $\mathbb{K}|\tilde{\mathbb{K}}$, le support peut être également calculé en partant uniquement du contexte $\mathbb{K}$ [24] :

$$Sup(T) = \sum_{PT(T) \subseteq U \subseteq T} (-1)^{n(U)} \times Sup(P(U)) \tag{4.1}$$

où $T = Y \cup Z$ représente l'ensemble de tous les attributs positifs et négatifs de l'implication, $Sup(T)$ est le support de T, $PT(T)$ est l'ensemble des éléments positifs dans T, $n(T)$ est le nombre d'éléments négatifs dans T, et $P(U)$ représente l'ensemble des attributs dans U exprimés en éléments positifs. Voici un exemple de l'application de l'équation 4.1 sur la règle suivante :

$$\{a', d', e'\} \rightarrow \{c'\}$$

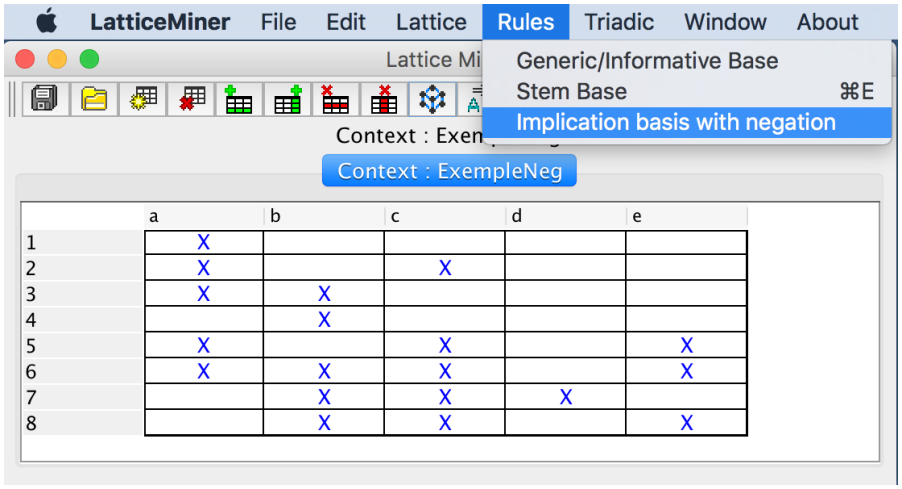
$$\begin{aligned} Sup(\{a', d', e', c'\}) = (-1)^0 \times Sup(\emptyset) \\ + (-1)^1 \times Sup(\{a\}) \\ + (-1)^1 \times Sup(\{d\}) \\ + (-1)^1 \times Sup(\{e\}) \\ + (-1)^1 \times Sup(\{c\}) \\ + (-1)^2 \times Sup(\{a, d\}) \\ + (-1)^2 \times Sup(\{a, e\}) \\ + (-1)^2 \times Sup(\{a, c\}) \\ + (-1)^2 \times Sup(\{d, e\}) \\ + (-1)^2 \times Sup(\{d, c\}) \\ + (-1)^2 \times Sup(\{e, c\}) \\ + (-1)^3 \times Sup(\{a, d, e\}) \\ + (-1)^3 \times Sup(\{a, d, c\}) \\ + (-1)^3 \times Sup(\{a, e, c\}) \\ + (-1)^3 \times Sup(\{d, e, c\}) \\ + (-1)^4 \times Sup(\{a, d, e, c\}) \end{aligned} \tag{4.2}$$

$$Sup(\{a', d', e', c'\}) = \boxed{0.125} \tag{4.3}$$

Ce résultat correspond à ce qu'on avait initialement calculé par exploitation directe du contexte $\mathbb{K}|\tilde{\mathbb{K}}$.

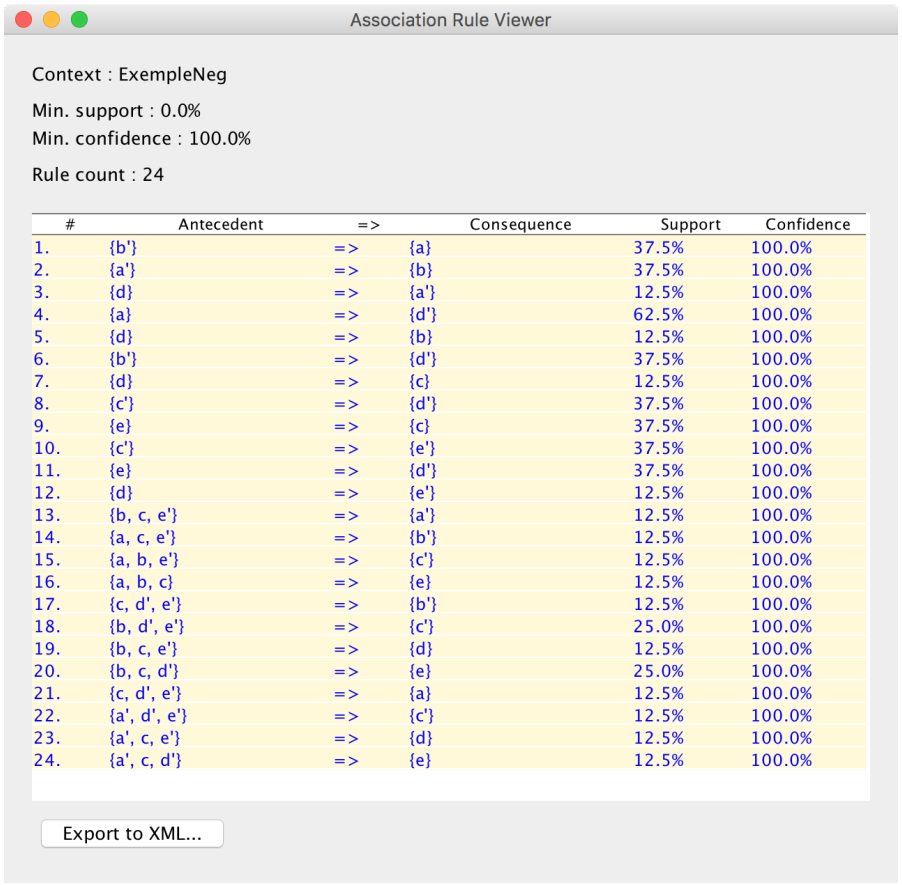
4.3.3 Intégration de la solution dans *Lattice Miner*

Dans le cadre de ce mémoire, nous avons intégré cette méthode de calcul des implications avec négation dans *Lattice Miner*. Pour l'exploiter, l'utilisateur crée un contexte et sélectionne le menu de génération des implications. En partant de l'exemple illustratif contenu dans ce chapitre, on constate que les résultats sont visibles dans la figure 4.3 et correspondent exactement à ce qu'on a obtenu dans le tableau 4.1.



	a	b	c	d	e
1	X				
2	X		X		
3	X	X			
4		X			
5	X		X		X
6	X	X	X		X
7		X	X	X	
8		X	X		X

(a) Éditeur de contexte



#	Antecedent	=>	Consequence	Support	Confidence
1.	{b'}	=>	{a}	37.5%	100.0%
2.	{a'}	=>	{b}	37.5%	100.0%
3.	{d}	=>	{a'}	12.5%	100.0%
4.	{a}	=>	{d'}	62.5%	100.0%
5.	{d}	=>	{b}	12.5%	100.0%
6.	{b'}	=>	{d'}	37.5%	100.0%
7.	{d}	=>	{c}	12.5%	100.0%
8.	{c'}	=>	{d'}	37.5%	100.0%
9.	{e}	=>	{c}	37.5%	100.0%
10.	{c'}	=>	{e'}	37.5%	100.0%
11.	{e}	=>	{d'}	37.5%	100.0%
12.	{d}	=>	{e'}	12.5%	100.0%
13.	{b, c, e'}	=>	{a'}	12.5%	100.0%
14.	{a, c, e'}	=>	{b'}	12.5%	100.0%
15.	{a, b, e'}	=>	{c'}	12.5%	100.0%
16.	{a, b, c}	=>	{e}	12.5%	100.0%
17.	{c, d', e'}	=>	{b'}	12.5%	100.0%
18.	{b, d', e'}	=>	{c'}	25.0%	100.0%
19.	{b, c, e'}	=>	{d}	12.5%	100.0%
20.	{b, c, d'}	=>	{e}	25.0%	100.0%
21.	{c, d', e'}	=>	{a}	12.5%	100.0%
22.	{a', d', e'}	=>	{c'}	12.5%	100.0%
23.	{a', c, e'}	=>	{d}	12.5%	100.0%
24.	{a', c, d'}	=>	{e}	12.5%	100.0%

(b) Affichage des implications avec négation

FIGURE 4.3: Les implications avec négation dans *Lattice Miner*

Chapitre 5

Enrichissement du module de génération des règles triadiques

Les relations ternaires ou plus généralement les relations n-aires mettent en lien trois ou plusieurs dimensions (axes d'analyse) et peuvent cacher des motifs intéressants sous forme de groupes homogènes ou associations entre les éléments des dimensions. Dans ce chapitre, nous allons rappeler l'analyse triadique de concepts en mettant davantage l'accent sur les implications triadiques. Ensuite, nous allons présenter la solution implémentée dans *Lattice Miner* pour la production de divers types d'implications triadiques.

5.1 Analyse triadique de concepts

L'analyse triadique de concepts a été introduite par Lehmann et Wille [19] comme une extension à l'analyse formelle de concepts. On part d'un contexte triadique $\mathbb{K} := (K_1, K_2, K_3, Y)$ avec K_1, K_2 et K_3 représentant respectivement un ensemble d'objets, un ensemble d'attributs et un ensemble de conditions, et $Y \subseteq K_1 \times K_2 \times K_3$ une relation ternaire entre les trois ensembles. Le triplet (a_1, a_2, a_3) dans Y signifie que l'objet a_1 possède l'attribut a_2 sous la condition a_3 (voir le tableau 5.1). Le but d'une telle analyse est d'identifier des concepts et des implications triadiques.

$\mathbb{K}$	P	N	R	K	S
1	abd	abd	ac	ab	a
2	ad	bcd	abd	ad	d
3	abd	d	ab	ab	a
4	abd	bd	ab	ab	d
5	ad	ad	abd	abc	a

Tableau 5.1: Un contexte triadique $\mathbb{K} := (K_1, K_2, K_3, Y)$

L'exemple du tableau 5.1 tiré de [23] représente un cube de données à trois dimensions (Client, Fournisseur, et Produit). Il y a cinq clients du groupe K_1 notés de 1 à 5 qui achètent auprès de cinq fournisseurs (**P**ierre, **N**elson, **R**ichard, **K**evin et **S**imon) du groupe K_2 les produits de l'ensemble K_3. Ces produits sont des **a**ccessoires, livres (***b**ooks*), ordinateurs (***c**omputers*) et caméras (***d**igital cameras*).

Définition 5.1. Un concept triadique d'un contexte $\mathbb{K} := (K_1, K_2, K_3, Y)$ est un triplet (A_1, A_2, A_3) avec $A_1 \subseteq K_1$, $A_2 \subseteq K_2$, $A_3 \subseteq K_3$ et $A_1 \times A_2 \times A_3 \subseteq Y$. Il représente un cuboïde plein de 1. Les sous-ensembles A_1, A_2 et A_3 sont appelés respectivement l'extension, l'intention et le mode (*modus*) du concept.

Du tableau 5.1, on peut extraire plusieurs concepts triadiques comme $(12345, PRK, a)$ et $(14, PN, bd)$. Par contre, le triplet $(135, PN, d)$ n'est pas un concept car il est non maximal puisque son extension peut être augmentée pour aboutir au concept $(12345, PN, d)$ tout en respectant la relation ternaire. En regardant la figure 5.1, on voit qu'il y a deux concepts triadiques associés au nœud n° 13 et ayant la même extension $\{1, 4\}$. Il s'agit de $(14, PNK, b)$ et $(14, PN, bd)$.

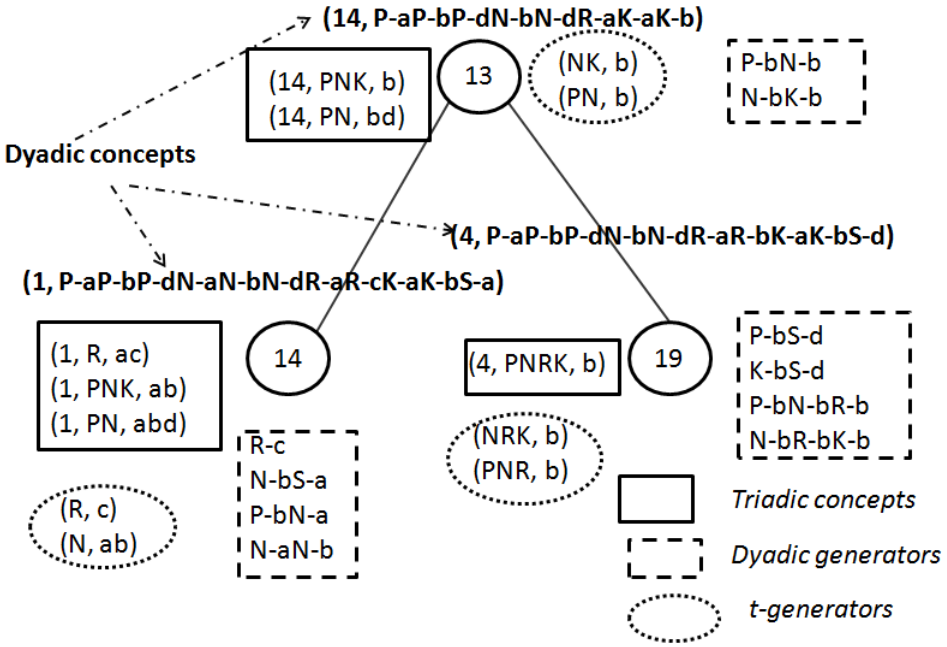


FIGURE 5.1: Concepts triadiques et dyadiques

$\mathbb{K}^{(1)}$	P				N				R				K				S			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
1	1	1		1	1	1		1	1		1		1	1			1			
2	1			1		1	1	1	1	1		1	1			1				1
3	1	1		1				1	1	1			1	1			1			
4	1	1		1		1		1	1	1			1	1						1
5	1			1	1			1	1	1		1	1	1	1		1			

Tableau 5.2: Contexte dyadique $\mathbb{K}^{(1)} := (K_1, K_2 \times K_3, Y^{(1)})$ extrait de $\mathbb{K}$

Pour les implications triadiques, c'est d'abord Biedermann [6] qui a proposé la définition suivante.

Définition 5.2. Une implication triadique de la forme $(A \to D)_C$ est vraie si chaque fois que A se produit pour l'ensemble des conditions dans C, alors D se produit également dans les mêmes conditions.

L'implication $(N \to P)_{abd}$ est vraie puisque chaque fois que le fournisseur N vend l'ensemble des produits $\{a, b, d\}$, le fournisseur P fait autant.

Plus tard, Ganter et Obiedkov [12] ont proposé trois nouveaux types d'implications : les implications de type *attribut*×*condition*, les implications d'*attributs conditionnels* (*conditional attribute implications*), et les implications de type *conditions attributionnelles* (*attributional condition implications*).

Définition 5.3. Une implication *attribut*×*condition* (AxCI) a la forme $A \to D$, avec A, D des sous-ensembles de $K_2 \times K_3$. De telles implications sont extraites du contexte binaire $\mathbb{K}^{(1)} := (K_1, K_2 \times K_3, Y^{(1)})$ où $(a_i, (a_j, a_k)) \in Y^{(1)} :\Leftrightarrow (a_i, a_j, a_k) \in Y$.

De telles implications sout plutôt dyadiques puisqu'elles sont produites à partir du contexte $\mathbb{K}^{(1)}$ issu de l'aplatissement du contexte triadique (voir le tableau 5.2). La figure 5.2 montre un sous-ensemble des règles d'association, y compris des implications triadiques AxCI. À titre d'exemple, l'implication $(N-b \to K-a, N-d, P-a, P-d, R-a)$ $[60\%, 100\%]$ signifie que tous les clients ayant acheté le produit b auprès du fournisseur N ont également acheté le produit a auprès des fournisseurs K, P et R et le produit d auprès des fournisseurs N et P. Le support de cette implication vaut 60%.

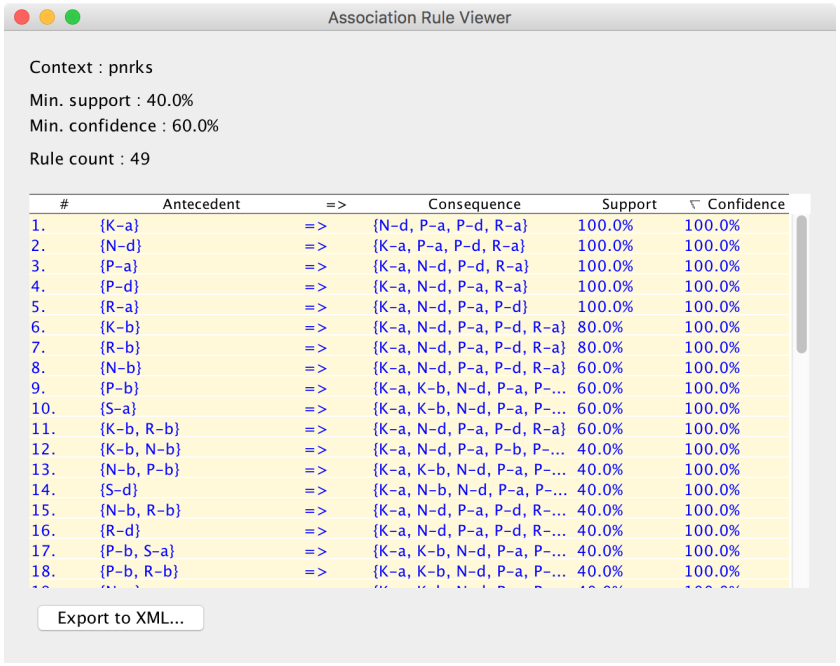


FIGURE 5.2: Implications AxCI

Définition 5.4. Une implication d'*attributs conditionnels* (CAI) est de la forme $A \xrightarrow{C} D$, avec A et D des sous-ensembles de K_2, et C un sous-ensemble de K_3. L'implication

signifie que A *implique* D *pour la totalité ou tout sous-ensemble des conditions de* C. Une telle implication est alors liée à la définition de l'implication triadique de Biedermann comme suit [12] : $A \xrightarrow{C} D \iff (A \to D)_{C_1}$ pour tout $C_1 \subseteq C$, y compris pour tout élément atomique de C.

Par exemple, $N \xrightarrow{ad} P$ veut dire que quand **N**elson fournit des **a**ccessoires et des caméras (***d****igital cameras*), alors **P**ierre fait autant.

Bien que l'implication à la Biedermann $(N \to P)_{abd}$ soit vraie, l'implication CAI $N \xrightarrow{abd} P$ ne l'est pas car $(N \to P)_{C_1}$ n'est pas vérifiée pour tous les $C_1 \subseteq \{a, b, d\}$ et en particulier pour $C_1 \in \{b, bd\}$.

Définition 5.5. Une implication de type conditions attributionnels (ACI) est de la forme $A \xrightarrow{C} D$, où A et D sont des sous-ensembles de K_3, et C est un sous-ensemble de K_2.

La relation $b \xrightarrow{PN} d$ signifie que chaque fois que les livres (***b****ooks*) sont fournis par **P**ierre et **N**elson (ou l'un d'eux), alors les caméras (***d****igital cameras*) sont aussi fournies par ces deux fournisseurs.

En étendant la notion d'implications d'*attributs conditionnels* aux règles d'association, on obtient la définition suivante :

Définition 5.6. Une règle d'association d'attributs conditionnels à la Biedermann (BCAAR) $(A \to D)_C$ $[s, c]$ est vraie si chaque fois que $A \subseteq K_2$ est vraie sous l'ensemble de conditions (et non nécessairement un sous-ensemble de) $C \subseteq K_3$, alors $D \subseteq K_2$ est vraie pour C avec un support s et une confiance c.

L'avantage des règles d'association triadiques et principalement des implications comme CAI et ACI repose sur le fait qu'elles représentent des motifs (connaissances) de manière plus compacte et significative que les AxCI et plus généralement des règles d'association qui peuvent être extraites du contexte formel (dyadique) $\mathbb{K}^{(1)} := (K_1, K_2 \times K_3, Y^{(1)})$ (cf. figure 5.2).

5.2 Algorithme de calcul des implications triadiques CAI et ACI

Comme on l'a vu dans l'introduction, l'un des objectifs de ce mémoire est d'implémenter dans *Lattice Miner* les trois formes d'implications triadiques à la Ganter.

Dans cette section nous allons présenter le pseudo-code de la procédure de calcul des implications CAI et puis nous allons détailler cette dernière à travers un exemple. Un raisonnement dual peut être effectué pour les implications ACI. Pour les implications AxCI, il suffit simplement, comme indiqué précédemment, d'aplatir le contexte tradique et ensuite calculer les implications.

En partant de la définition même d'une implication CAI qui indique que $A \xrightarrow{C} D$ est vraie ssi $(A \rightarrow D)_k$ pour tout $k \in C$, nous allons procéder comme suit :

1. créer autant de contextes dyadiques qu'il y a de conditions dans K_3
2. calculer les implications à la Biedermann au niveau des chacune des conditions
3. placer les implications avec un même antécédent dans un même baquet (*bucket*) puis décomposer chaque implication avec p attributs dans la conclusion en p implications avec un seul attribut dans la conséquence
4. cumuler les conditions pour toutes les implications ayant le même antécédent et la même conséquence
5. regrouper les conclusions des implications ayant le même antécédent
6. calculer le support des implications obtenues.

Algorithme 2 : Génération des implications triadiques CAI à partir de $\mathbb{K}$

Procédure
Entrée : Contexte triadique $\mathbb{K} = (K_1, K_2, K_3, I)$
Sortie : Σ_{CAI}, l'ensemble des implications CAI
$\mathbb{H} \leftarrow$ SLICE($\mathbb{K}$) {production d'autant de contextes dyadiques que de conditions dans K_3.}
$\Sigma_I \leftarrow \emptyset$ {Ensemble des implications à une seule condition}
for all $\mathbb{K}(k) := (G, M, I_k) \in \mathbb{H}$ **do**
  $\Sigma_k \leftarrow$ BaseGen($\mathbb{K}(k), 0, 1$) {On génère les implications du contexte $\mathbb{K}(k)$ avec un support minimum nul}
  $\Sigma_I := \Sigma_I \cup \Sigma_k$
end for
$Labels \leftarrow \emptyset$ {On crée un ensemble d'étiquettes de baquets}
for all $r : A \rightarrow B \in \Sigma_I$ **do**
  if $A \notin Labels$ **then**
    $Labels := Labels \cup A$ {Si l'étiquette du baquet n'existe pas, on la crée}
  $Buck(A) := Buck(A) \cup r$ {$Buck(A)$ est le baquet contenant les implications avec l'antécédent A}
end for
$\Sigma_{CAI} \leftarrow \emptyset$
for all $l \in Labels$ **do**
  $\Sigma_{CAI} := \Sigma_{CAI} \cup Traite(Buck(l))$ {cette fonction consiste à cumuler les conditions d'une même implication d'un même baquet}
end for
return Σ_{CAI}

Passage d'un contexte triadique à un ensemble de contextes dyadiques

Pour commencer, nous partons du contexte triadique $\mathbb{K} := (K_1, K_2, K_3, I)$ du tableau 5.1, avec K_1 l'ensemble des objets, K_2 l'ensemble des attributs, K_3 l'ensemble des conditions et I la relation ternaire entre ces trois derniers ensembles. La première étape consiste à produire $|K_3|$ contextes formels (dyadiques) de la forme $\mathbb{K}(k) = (K_1, K_2, I_k)$ avec $k = 1, \ldots |B|$ et I_k la relation binaire entre K_1 et K_2 pour la condition k. En suivant toujours le même exemple de contexte triadique, nous obtenons quatre contextes $\mathbb{K}(a)$, $\mathbb{K}(b)$, $\mathbb{K}(c)$, et $\mathbb{K}(d)$.

$\mathbb{K}(a)$	P	N	R	K	S
1	X	X	X	X	X
2	X		X	X	
3	X		X	X	X
4	X		X	X	
5	X	X	X	X	X

$\mathbb{K}(b)$	P	N	R	K	S
1	X	X		X	
2		X	X		
3	X		X	X	
4	X	X	X	X	
5			X	X	

$\mathbb{K}(c)$	P	N	R	K	S
1			X		
2		X			
3					
4					
5				X	

$\mathbb{K}(d)$	P	N	R	K	S
1	X	X			
2	X	X	X	X	X
3	X	X			
4	X	X			X
5	X	X	X		

FIGURE 5.3: Les contextes dyadiques extraits de $\mathbb{K}$ pour chaque condition

Calcul des implications

Dans l'étape qui suit, nous devons générer les implications Σ_k pour chaque contexte $\mathbb{K}(k)$. Pour cela, nous réutilisons l'algorithme qui permet générer la base générique d'implications qui est déjà disponible dans *Lattice Miner*. Cette méthode est implémentée en utilisant plusieurs fils d'exécution en parallèle. Nous obtenons les règles suivantes :

Tableau 5.3: Implications non redondantes à la Biedermann pour les contextes $\mathbb{H}_k$.

Contexte $\mathbb{K}(a)$	Contexte $\mathbb{K}(b)$	Contexte $\mathbb{K}(c)$	Contexte $\mathbb{K}(d)$
$(\{K\} \to \{P, R\})_a$	$(\{K, N\} \to \{P\})_b$	$(\{P\} \to \{K, N, R, S\})_c$	$(\{N\} \to \{P\})_d$
$(\{P\} \to \{K, R\})_a$	$(\{N, P\} \to \{K\})_b$	$(\{S\} \to \{K, N, P, R\})_c$	$(\{P\} \to \{N\})_d$
$(\{R\} \to \{K, P\})_a$	$(\{P\} \to \{K\})_b$	$(\{N, R\} \to \{K, P, S\})_c$	$(\{S\} \to \{N, P\})_d$
$(\{S\} \to \{K, P, R\})_a$	$(\{K, N, R\} \to \{P\})_b$	$(\{K, N\} \to \{P, R, S\})_c$	$(\{R\} \to \{N, P\})_d$
$(\{N\} \to \{K, P, R, S\})_a$	$(\{N, P, R\} \to \{K\})_b$	$(\{K, R\} \to \{N, P, S\})_c$	$(\{K\} \to \{N, P, R, S\})_d$
	$(\{P, R\} \to \{K\})_b$		$(\{R, S\} \to \{K, N, P\})_d$
	$(\{S\} \to \{K, N, P, R\})_b$		

Placement des implications dans des baquets

Le but de cette étape est de placer toutes les implications ayant le même antécédent (cf. tableau 5.3) dans un même baquet. Par exemple, les implications $(\{K, N\} \to \{P\})_b$ et $(\{K, N\} \to \{P, R, S\})_c$ vont se retrouver dans le même baquet relatif à l'antécédent $\{K, N\}$.

Création des règles CAI

À l'intérieur de chaque baquet, on décompose chaque implication en autant d'implications qu'il y a d'attributs dans la conclusion afin de faciliter les opérations subséquentes visant le cumul des conditions. Par exemple $(\{K\} \rightarrow \{P, R\})_a$ donne lieu à $(\{K\} \rightarrow \{P\})_a$ et $(\{K\} \rightarrow \{R\})_a$. Puis, nous rassemblons les implications ayant le même attribut en conclusion pour ensuite cumuler les conditions.

L'avantage de cette méthode est qu'elle peut encore une fois être traitée en parallèle sur plusieurs fils d'exécution. Par exemple, prenons le baquet $\{P\}$ dans le tableau 5.4.

Tableau 5.4: Exemple du baquet $\{P\}$

Baquet $\{P\}$
$(\{P\} \rightarrow \{K, R\})_a$
$(\{P\} \rightarrow \{K\})_b$
$(\{P\} \rightarrow \{K, N, R, S\})_c$
$(\{P\} \rightarrow \{N\})_d$

On obtient les implications CAI suivantes pour le baquet $\{P\}$ (cf. tableau 5.5).

Tableau 5.5: Implications CAI du bac $\{P\}$

Implications CAI du bac $\{P\}$
$P \xrightarrow{c,d} N$
$P \xrightarrow{a,c} R$
$P \xrightarrow{c} S$
$P \xrightarrow{a,b,c} K$

Lorsqu'une implication CAI provient d'une seule et même implication à une seule condition, le support est déjà calculé. Cependant, quand elle est le résultat de plusieurs implications, nous ne pouvons pas déterminer systématiquement le support de cette nouvelle implication CAI et devons le calculer par consultation du contexte triadique initial. Toutefois, lorsque le support de l'une des implications initiales est nul, alors le support de l'implication créée est également nul.

Avantages de cette nouvelle méthode

L'avantage de cette nouvelle méthode de création des implications triadiques CAI et ACI repose sur le fait qu'elle est plus performante que celle initialement implantée

en 2015 dans *Lattice Miner*. L'ancienne méthode consiste à aplatir le contexte initial et donc à produire le contexte formel $\mathbb{K}^{(1)}$ pour générer en premier lieu un ensemble souvent volumineux d'implications A×C. Ensuite, on calcule les implications à la Biedermann pour finalement calculer les implications triadiques à la Ganter. En ce qui concerne les performances, cette nouvelle méthode a l'avantage de pouvoir s'exécuter en parallèle par traitement simultané des baquets.

Résultats de la procédure

Voici les implications obtenues à partir de l'exemple initial du contexte du tableau 5.1.

Tableau 5.6: Résultats de la procédure CAI sur le contexte $\mathbb{K}$.

$K \xrightarrow{a,d} \{P,R\}$	$R \xrightarrow{d} N$	$\{N,P\} \xrightarrow{b} K$
$K \xrightarrow{d} \{N,S\}$	$S \xrightarrow{b,c,d} N$	$\{K,N,R\} \xrightarrow{b} P$
$P \xrightarrow{c,d} N$	$S \xrightarrow{a,b,c} \{K,R\}$	$\{N,P,R\} \xrightarrow{b} K$
$P \xrightarrow{a,c} R$	$S \xrightarrow{a,b,c,d} P$	$\{P,R\} \xrightarrow{b} K$
$P \xrightarrow{c} S$	$N \xrightarrow{a} \{K,R,S\}$	$\{N,R\} \xrightarrow{c} \{K,P,S\}$
$P \xrightarrow{a,b,c} K$	$N \xrightarrow{a,d} P$	$\{K,R\} \xrightarrow{c} \{N,P,S\}$
$R \xrightarrow{a} K$	$\{K,N\} \xrightarrow{b,c} P$	$\{R,S\} \xrightarrow{d} \{K,N,P\}$
$R \xrightarrow{a,d} P$	$\{K,N\} \xrightarrow{c} \{R,S\}$	

5.3 Les implications triadiques dans *Lattice Miner*

Toutes ces variantes d'implications triadiques et de règles d'association sont produites par notre outil en utilisant nos procédures définies précédemment [23, 31], comme l'illustre la figure 5.4.

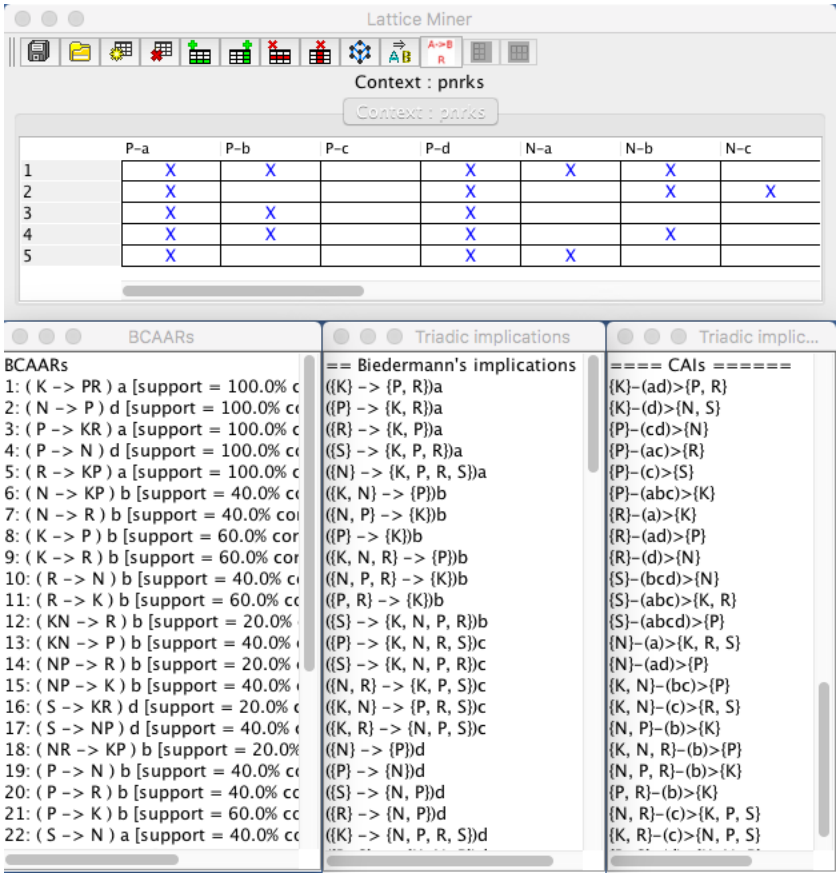


FIGURE 5.4: Implications triadiques dans *Lattice Miner*

En effet, la figure 5.4 montre le résultat d'exécution des procédures de calcul des implications triadiques. En haut de la figure, on affiche le contexte dyadique $\mathbb{K}^{(1)}$ extrait du contexte triadique présenté dans le tableau 5.1. À gauche, ce sont les règles d'association triadiques à la Biedermann (BCAAR). On trouve les implications à la Biedermann au milieu et les implications d'attributs conditionnels (CAI) à droite.

Pour l'intégration de toutes ces méthodes, nous avons également ajouté le support des fichiers JSON (*JavaScript Object Notation*) pour la lecture des contextes triadiques. Ce format a l'avantage d'être simple et est très utilisé dans le développement d'applications modernes. D'ailleurs, beaucoup de librairies existent pour manipuler les fichers JSON pour différents langages de programmation. Ce format est également utilisé par défaut avec certaines bases de données NoSQL tel que MongoDB [1].

1. `https://www.mongodb.com/`

Chapitre 6

Conclusion

Le but de ce mémoire est d'enrichir le prototype *Lattice Miner* qui utilise l'analyse formelle de concepts (AFC) comme cadre théorique de référence pour la fouille de données à travers un treillis de concepts. Dans le cadre de ce mémoire, nous avons d'abord rappelé les notions de base de l'AFC puis présenté plusieurs outils existants dans ce domaine. Certains de ces outils, comme *ConExp*, nous ont aidé à vérifier quelques fonctionnalités de *Lattice Miner* à savoir, les procédures de production de la base d'implications de Guigues–Duquenne et le calcul des relations flèches. Ensuite, nous avons présenté une implémentation d'une procédure de calcul des implications avec négation ajoutée à la nouvelle version de *Lattice Miner*. Enfin, après une présentation de l'analyse triadique de concepts, nous avons enrichi le module de génération des règles triadiques pour obtenir les trois types d'implications identifiés par Ganter [12].

La nouvelle version de *Lattice Miner* est maintenant disponible sur les plate-formes SourceForge [1] et GitHub [2]. Le code source est disponible pour téléchargement et est mis à jour avec la dernière version de Java 8. Une brève documentation technique est produite pour aider les prochains développeurs du prototype. Une présentation de cette nouvelle version de *Lattice Miner* sera également faite à la conférence ICFCA 2017 [22].

Parmi les travaux futurs, nous envisageons d'associer au module de calcul des implications (y compris celles avec négation et les implications triadiques) une autre procédure qui permet de calculer le fermé d'un groupe d'attributs Y avec ou sans un ensemble C de conditions. Finalement, une version Web *Lattice Miner* est également envisagée et aura l'avantage d'être encore plus rapide, car elle sera exécutée sur des serveurs plus

1. `https://sourceforge.net/projects/lattice-miner/`
2. `https://github.com/LarimUQO/lattice-miner`

performants et pourra exploiter le parallélisme pour l'exécution de quelques procédures comme la nouvelle procédure de production des implications triadiques à la Ganter.

Bibliographie

[1] Agrawal, R., Imieliński, T., and Swami, A. Mining Association Rules Between Sets of Items in Large Databases. *SIGMOD Rec. 22*, 2 (June 1993), 207–216.

[2] Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., and Lakhal, L. Mining frequent patterns with counting inference. *SIGKDD Explorations 2*, 2 (December 2000).

[3] Bazhanov, K., and Obiedkov, S. Optimizations in computing the Duquenne-Guigues basis of implications. *Annals of Mathematics and Artificial Intelligence 70*, 1 (2013), 5–24.

[4] Becker, P., Hereth, J., and Stumme, G. ToscanaJ : An Open Source Tool for Qualitative Data Analysis. In *Advances in Formal Concept Analysis for Knowledge Discovery in Databases (FCAKDD'2002)* (July 2002), pp. 1–2.

[5] Bertet, K., Demko, C., Viaud, J.-F., and Guérin, C. java-lattices : a Java library for lattices computation. `http://thegalactic.org`, 2014.

[6] Biedermann, K. How Triadic Diagrams Represent Conceptual Structures. In *ICCS* (1997), pp. 304–317.

[7] Bordat, J. P. Calcul pratique du treillis de Galois d'une correspondance. *Mathématiques et Sciences Humaines 96* (1986), 31–47.

[8] Caspard, N., Leclerc, B., and Monjardet, B. *Ensembles ordonnés finis : concepts, résultats et usages*. Springer-Verlag Berlin Heidelberg, 2007.

[9] Demko, C., and Bertet, K. Generation Algorithm of a Concept Lattice with Limited Object Access. In *Proceedings of The Eighth International Conference on Concept Lattices and Their Applications* (2011), pp. 239–250.

[10] Fredman, M. L., and Khachiyan, L. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *J. Algorithms 21*, 3 (1996), 618–628.

[11] Ganter, B. *Formal Concept Analysis : 8th International Conference, ICFCA 2010, Agadir, Morocco, March 15-18, 2010. Proceedings*. Springer Berlin Heidelberg, 2010, ch. Two Basic Algorithms in Concept Analysis, pp. 312–340.

[12] Ganter, B., and Obiedkov, S. A. Implications in Triadic Formal Contexts. In *ICCS* (2004), pp. 186–195.

[13] Ganter, B., and Wille, R. *Formal Concept Analysis : Mathematical Foundations*. Springer-Verlag New York, Inc., 1999.

[14] Guigues, J. L., and V., D. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines 95* (1986), 5–18.

[15] Han, J., and Kamber, M. *Data Mining : Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[16] Kaytoue, M., Marcuola, F., Napoli, A., Szathmary, L., and Villerd, J. The Coron System. In *8th International Conference on Formal Concept Analsis - ICFCA 2010* (Agadir, Morocco, 2010), pp. 55–58.

[17] Kryszkiewicz, M., and Gajek, M. Concise Representation of Frequent Patterns Based on Generalized Disjunction-Free Generators. In *6th Pacific-Asia Conference, PAKDD* (2002), Springer-Verlag, pp. 159–171.

[18] Le Floc'h, A., Fisette, C., Missaoui, R., Valtchev, P., and Godin, R. JEN : un algorithme efficace de construction de générateurs pour l'identification des règles d'association. *Revue ECA X*, Y (2003), 1–Z.

[19] Lehmann, F., and Wille, R. A Triadic Approach to Formal Concept Analysis. In *Proceedings of the Third International Conference on Conceptual Structures : Applications, Implementation and Theory* (1995), pp. 32–43.

[20] Luxenburger, M. Implications partielles dans un contexte. *Mathématiques, informatique et sciences humaines 29*, 113 (1991), 35–55.

[21] Maier, D. *The Theory of Relational Databases*. Computer Science Press, 1983.

[22] Missaoui, R., and Emamirad, K. Lattice Miner 2.0 : A Formal Concept Analysis Tool. In *Supplementary Proc. of ICFCA, Rennes, France, 2017* (2017), pp. 91–94.

[23] Missaoui, R., and Kwuida, L. Mining Triadic Association Rules from Ternary Relations. In *9th International Conference ICFCA* (May 2011), pp. 204–218.

[24] Missaoui, R., Nourine, L., and Renaud, Y. Computing Implications with Negation from a Formal Context. *Fundam. Inf. 115*, 4 (December 2012), 357–375.

[25] Nehmé, K., Valtchev, P., Rouane, M. H., and Godin, R. *On Computing the Minimal Generator Family for Concept Lattices and Icebergs*. Springer Berlin Heidelberg, 2005, pp. 192–207.

[26] Objedkov, S. The Concept Explorer. `http://conexp.sourceforge.net/`, 2006.

[27] Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. Efficient Mining of Association Rules Using Closed Itemset Lattices. *Information Systems 24*, 1 (1999), 25–46.

[28] Pfaltz, J., and Taylor, C. Scientific discovery through iterativet ransformations of concept lattices. In *Proceedings of the 1st Internationla Workshop on Discrete Mathematics and Data Mining* (April 2002), pp. 65–74.

[29] Priss, U. Formal Concept Analysis Homepage. `http://www.fcahome.org.uk/fca.html`, 2007.

[30] Roberge, G. Visualisation des résultats d'une fouille de données dans les treillis de concepts. Master's thesis, Université du Québec en Outaouais, 2007.

[31] Rodriguez-Lorenzo, E., Cordero, P., Enciso, M., Missaoui, R., and Mora, A. An axiomatic System for Conditional Attribute Implications in Triadic concept analysis. *International Journal of Intelligent Systems* (February 2017).

[32] Rouane Hacene, M., Huchard, M., Napoli, A., and Valtchev, P. Relational concept analysis : mining concept lattices from multi-relational data. *Ann. Math. Artif. Intell. 67*, 1 (2013), 81–108.

[33] Szathmary, L. *Symbolic Data Mining Methods with the Coron Platform*. Thèse de doctorat, Univ. Henri Poincaré - Nancy 1, France, November 2006.

[34] Valtchev, P., Grosser, D., Roume, C., and Hacene, M. R. Galicia : An Open Platform for Lattices. In *Using Conceptual Structures : Contributions to the 11th Intl. Conference on Conceptual Structures* (2003), Shaker Verlag, pp. 241–254.

[35] Viaud, J.-F., Bertet, K., Demko, C., and Missaoui, R. Décomposition sous-directe d'un treillis en facteurs irréductibles. In *Journées francophones d'Ingénierie des Connaissances IC 2015* (Rennes, France, June 2015), collection AFIA.