Ann Math Artif Intell (2007) 49:77–99
DOI 10.1007/s10472-007-9057-2

Attribute-incremental construction of the canonical implication basis

S. Obiedkov · V. Duquenne

Published online: 5 July 2007

Abstract We propose a new algorithm constructing the canonical implication basis of a formal context. Being incremental, the algorithm processes a single attribute of the context at a single step. Experimental results bear witness to its competitiveness.

Keywords Galois lattice · Formal concept analysis · Implication · Duquenne–Guigues basis · Incremental algorithm

Mathematics Subject Classifications (2000) 03G10 · 06A15 · 68R05 · 68W05 · 68W40

1 Introduction

Recent years have seen an increased mutual interest between the communities of lattice theory [3] researchers, especially those working within the framework of formal concept analysis (FCA) [10], and developers of intelligent data analysis systems. This is attested by the workshops held at Stanford [18] and in Lyon [13], as well as special issues of the *Journal of Experimental and Theoretical Artificial Intelligence* (vol. 14, nos. 2–3, 2002) and *Applied Artificial Intelligence* (vol. 17, no. 3, 2003). Data analysis can be understood as the discovery of plausible dependencies in data that can be used for data recovery, classification, etc. One kind of such

Expanded version of a talk presented at Journées de l'Informatique Messine (Metz, France, September 2003).

S. Obiedkov (✉)
Department of Applied Mathematics, Higher School of Economics, 33/5 u1. Kirpichnaya, 105 679 Moscow, Russia
e-mail: sergei.obj@gmail.com

V. Duquenne
CNRS UMR 7090, ECP6, 175 rue du Chevaleret, 75013 Paris, France
e-mail: v.duquenne@wanadoo.fr

dependencies is given by attribute implications in FCA, which are closely related to functional dependencies in databases.

We discuss the construction of the canonical implication basis [12]; it is a non-redundant implication set of minimal size from which all implications valid in the dataset can be inferred by the Armstrong rules. This kind of construction is by no means exotic or even original. It has appeared in many surrounding areas such as the simplification of Boolean functions, Horn theories, compilation of knowledge in artificial intelligence, up to the discovery or simplification of functional dependencies for bettering the design of databases (see [16] and the references therein, as well as more recent works, e.g., [2]). In these contexts, the main problem consists in compressing a collection of implications (logical expressions, rules, etc.) without loosing information, which can be "internally" done by removing redundancies (and more efficiently by introducing external structures and algorithms based on trees, directed acyclic graphs, etc. [16]). Here, our approach is somehow different, as the input is a "context" (binary matrix) describing objects by binary attributes, and since we adopt an attribute-incremental attitude that can be especially useful when a database is refined by introducing new descriptors.

In what follows, we describe the most popular algorithm for basis construction, **Next Closure**, and propose a new incremental algorithm. Essentially, an algorithm can be incremental in one of two ways depending on whether it processes objects or attributes. Adding a new object may invalidate some attribute implications, but it will never generate new implications. On the other hand, when adding an attribute, old implications remain valid, but new implications (involving the new attribute) may appear. We deal only with the latter (easier) type of incremental update.

2 Formal contexts and implications

In formal concept analysis, data is represented by binary tables regarded as "contexts." A (*formal*) *context* is a triple $\mathbb{K} = (G, M, I)$, where G is an object set, M is an attribute set, and the relation $I \subseteq G \times M$ specifies which objects possess which attributes [10].

Example 1 A formal context is usually visualized by a cross table ($M = \{1, \ldots, 4\}$, for which 1: has exactly four vertices; 2: has exactly three vertices; 3: has a right angle; 4: all sides are equal):

$G \setminus M$	1	2	3	4
a: square	×		×	×
b: rectangle	×		×	
c: right triangle		×	×	
d: isosceles triangle		×		×

For arbitrary $A \subseteq G$ and $B \subseteq M$, the following *derivation operators* are defined:

$$A^I = \{m \in M \mid \forall g \in A(gIm)\};$$
$$B^I = \{g \in G \mid \forall m \in B(gIm)\}.$$

In FCA, it is common to write A' and B' instead of A^I and B^I when this does not result in ambiguity. Sometimes, it is convenient to use specific denotations for derivation operators of different contexts; in these cases, we will provide necessary definitions.

The two mappings $A \mapsto (A')'$ and $B \mapsto (B')'$ given by the composition of the two derivation operators are closure operators (respectively, $\mathfrak{P}(G) \to \mathfrak{P}(G)$ and $\mathfrak{P}(M) \to \mathfrak{P}(M)$), i.e., they are idempotent, extensive, and monotone. We denote $(A')'$ by A'' and $(B')'$ by B''.

A couple of sets (A, B) such that $A \subseteq G$, $B \subseteq M$, $A' = B$, and $B' = A$ is called a (*formal*) *concept* of the context $\mathbb{K}$. The sets A and B are closed; they are called respectively the *extent* and the *intent* of the formal concept (A, B). A concept (A, B) is *more general* than a concept (C, D) if $C \subset A$ (equivalently, $B \subset D$). The set of all concepts of a context $\mathbb{K}$ ordered in this way is a lattice, which we denote by $L(\mathbb{K})$.

An *implication* is an expression $A \to B$ where $A, B \subseteq M$. Set A is called the *premise* and set B is called the *conclusion* or *consequence* of the implication $A \to B$. The implication $A \to B$ holds in the context $\mathbb{K}$ if $A' \subseteq B'$ (equivalently, $B \subseteq A''$), i.e., every object possessing all attributes from A also possesses all attributes from B.

Implications obey the Armstrong rules [1], and, in this sense, we can speak about a *cover of a set of implications*, i.e., a subset of implications from which all other implications can be inferred by the Armstrong rules. Of special interest is the *canonical implication basis of the context* (usually called the *Duquenne–Guigues basis*), which is a minimal cover of the set of implications that hold in the context. The fact that all bases (minimal covers) have the same cardinality was already known in the database literature [16], but the specific role of the canonical basis in that was made explicit in [23] and [24] (see also [10] Section 2.3). The canonical basis is defined using the notion of a pseudo-closed set introduced below.

A set $A \subseteq M$ is called *quasi-closed* (with respect to the closure operator $''$) if $B'' \subset A$ or $B'' = A''$ for any $B \subset A$.

Example 2 In Example 1, the set {1} is quasi-closed, since it has only one proper subset, namely, Ø, and $\emptyset'' = \emptyset \subset \{1\}$. The other quasi-closed sets are Ø, {2}, {3}, {4}, {1, 3}, {2, 3}, {2, 4}, {3, 4}, {1, 2, 3}, {1, 3, 4}, {1, 2, 3, 4}.

A quasi-closed set A is called *pseudo-closed* if $A \neq A''$ and $B'' \subset A$ for any quasi-closed $B \subset A$. A *pseudo-intent* of the context is a pseudo-closed (with respect to $''$) attribute set.

In [10], an equivalent recursive definition of a pseudo-closed set is given: a non-closed set A is pseudo-closed if $B'' \subset A$ for any pseudo-closed $B \subset A$.

Example 3 In Example 1, pseudo-intents are {1}, {3, 4}, and {1, 2, 3}. In this context, all other quasi-closed sets are closed (e.g., consider $\{3\}'' = \{3\}$), though this is not always the case.

The *canonical implication basis* of the context $\mathbb{K}$ (notation: $B(\mathbb{K})$) consists of all implications $P \to P''$ where P is a pseudo-intent of the context $\mathbb{K}$ [12].

Example 4 The canonical basis of the context from Example 1 consists of implications $\{1\} \to \{1, 3\}$, $\{3, 4\} \to \{1, 3, 4\}$, and $\{1, 2, 3\} \to \{1, 2, 3, 4\}$.

Unlike in the case of lattice construction, there are only a few known algorithms constructing the implication basis. The only relatively efficient algorithm that builds the canonical basis directly is that of Ganter [9]. It is a modification of his **Next Closure** algorithm for computing the concept set of a context. There are also algorithms constructing other implication bases: direct basis [17, 24], proper basis [20], etc. The output of these algorithms can then be reduced to the canonical basis [6]. Such approach is justified if the algorithm is fast, the size of its output is not large (with respect to $2^{|M|}$), and the reduction procedure is efficient.

The algorithm proposed below constructs the implication basis processing attributes of the context one by one. Obviously, a new attribute does not cancel old implications, but it may generate new ones. However, a new attribute can change the set of pseudo-intents in an arbitrary way; therefore, addition of an attribute does not always enlarge the canonical basis.

Example 5 Consider context $(M, M, =)$, where $M = \{1, 2, 3, 4, 5\}$:

	1	2	3	4	5
1	×				
2		×			
3			×		
4				×	
5					×

Pseudo-intents of this context are exactly all two-element attribute sets, and the canonical basis consists of ten implications. After adding new attribute 6

	6
1	×
2	
3	
4	
5	×

the basis consists of only nine implications. The new basis contains one implication with 6 for every attribute of the initial context:

$$\{5\} \to \{6\}$$
$$\{4, 6\} \to \{1, 2, 3, 5\}$$
$$\{3, 6\} \to \{1, 2, 4, 5\}$$
$$\{2, 6\} \to \{1, 3, 4, 5\}$$
$$\{1\} \to \{6\}$$

However, the implications $\{1\} \to \{6\}$ and $\{5\} \to \{6\}$ make possible a more compact representation of the set of implications with premises $\{m, n\}$, where $m \in \{1, 5\}$ and $n \in \{2, 3, 4\}$: one implication with attribute 6 replaces two implications with attributes 1 and 5.

3 Ganter's algorithm for computing the canonical basis

One of the advantages of Ganter's **Next Closure** algorithm is its universality. Let M be a set, and $\sigma : \mathfrak{P}(M) \to \mathfrak{P}(M)$ be an arbitrary closure operator. Then, **Next Closure** can be used to produce all σ-closed (i.e., closed with respect to σ) subsets of M.

Assuming a linear order $<$ on the attribute set M, let $min(C)$ be the minimal element and $max(C)$ be the maximal element of $C \subseteq M$ with respect to $<$. Set A is *lectically smaller* than set B if

$$min((A \cup B) \setminus (A \cap B)) \in B.$$

In particular, subsets are lectically smaller than their supersets. The idea of the algorithm is to iterate through subsets of M in lectic order computing the closure of each subset [10]. Several subsets may have the same closure, but only the last generation of every closed set counts. In other words, regarding sets as strings of attributes, the *canonical generator* of a closed set C is its minimal prefix whose closure is equal to C. When a new closure C is generated canonically, the next set to be processed by the algorithm is the set that follows C with respect to the lectic order. On the other hand, if set A is not the canonical generator of its closure, the algorithm skips all sets that differ from A only in that they contain some additional attributes greater than $max(A)$: obviously, such sets are not prefixes (hence, not canonical generators) of their closures, either (due to the monotony of closure operators). This makes it possible to avoid computing closures of a large number of sets.

It is well known that closed and pseudo-closed sets form together a new closure system. Here, we call the closure operator corresponding to this system the *saturation operator*[1] and denote it by $^\bullet$. It can be defined as the transitive closure of the following $^+$ operator:

$$A^+ = A \cup \bigcup \{B'' \mid B \subset A \text{ and } B \text{ is pseudo-closed}\}$$

$$A^\bullet = A^{+ + \cdots +}$$

Note that $\{B'' \mid B \subset A \text{ and } B \text{ is pseudo-closed}\} = \{C \mid B \subset A \text{ and } B \to C \text{ is in the canonical basis}\}$. Thus, $A^\bullet$ can be obtained by applying to A implications from the canonical basis, and even only those implications whose premises are subsets of $A^\bullet$. This makes it possible to use **Next Closure** to compute all sets closed with respect to $^\bullet$, i.e., all intents and pseudo-intents of the context. If the next generated set $A^\bullet$ is not closed with respect to $''$, then it is pseudo-closed and the implication $A^\bullet \to A''$ is added to the basis. When **Next Closure** processes A, it has already generated all implications $B \to B''$ with $B \subset A^\bullet$ from the basis, and it uses them to compute $A^\bullet$.

[1] The saturation operator is defined differently in [12].

Algorithm 1 (Ganter's algorithm)

Input A context $\mathbb{K} = (G, M, I)$
Output The canonical implication basis of $\mathbb{K}$
Begin
  $Basis := \emptyset$, $A := \emptyset$
  $m := max(M)$
  while $A \neq M$
    $B := Saturate(A, Basis)$
    if there is no $n \in B \setminus A$ such that $n < m$ [the "canonicity" test]
      then
        if $B \neq B''$ **then** Add $B \rightarrow B''$ to $Basis$
        $A := B$
        $m := max(\{n \mid n \in M \setminus B\})$
      else $m := max(\{n \mid n \in M \setminus A \text{ and } n < m\})$
    $A := A \cup \{m\} \setminus \{n \mid n \in A \text{ and } m < n\}$
  return $Basis$
End

To make the description of the algorithm self-contained, we describe a "naïve" procedure to calculate the saturation. There are other suitable algorithms for this purpose: LinClosure, which is well known in the database theory [16]; the one proposed in [24], which has nonlinear time complexity but is claimed to be an enhanced version of LinClosure; etc.

Algorithm 2

Saturate($New\,Prem$, $Impl$)
Input Attribute set $New\,Prem \subseteq M$, implication set $Impl$
Output Saturation of $New\,Prem$, i.e., $New\,Prem^{\bullet}$ (with respect to $Impl$)
Begin
  $NewClosure := New\,Prem$
  $UnusedImpl := Impl$
  do
    $OldClosure := NewClosure$
    for each $(Prem \rightarrow Cons)$ in $UnusedImpl$
      if $(Prem \subseteq NewClosure)$
        then
          $NewClosure := NewClosure \cup Cons$
          $UnusedImpl := UnusedImpl \setminus \{Prem \rightarrow Cons\}$
  while $OldClosure \neq NewClosure$
  return $NewClosure$
End

Let us show how Ganter's algorithm works on the context from Example 1.

	$A = \emptyset$,	$B = \emptyset$,	$B'' = \emptyset$
$m = 4$,	$A = \{4\}$,	$B = \{4\}$,	$B'' = \{4\}$
$m = 3$,	$A = \{3\}$,	$B = \{3\}$,	$B'' = \{3\}$
$m = 4$,	$A = \{3, 4\}$,	$B = \{3, 4\}$,	$B'' = \{1, 3, 4\}$ $\boxed{\{3, 4\} \to \{1, 3, 4\}}$
$m = 2$,	$A = \{2\}$,	$B = \{2\}$,	$B'' = \{2\}$
$m = 4$,	$A = \{2, 4\}$,	$B = \{2, 4\}$,	$B'' = \{2, 4\}$
$m = 3$,	$A = \{2, 3\}$,	$B = \{2, 3\}$,	$B'' = \{2, 3\}$
$m = 4$,	$A = \{2, 3, 4\}$,	$B = M$,	the "canonicity" test fails
$m = 1$,	$A = \{1\}$,	$B = \{1\}$,	$B'' = \{1, 3\}$ $\boxed{\{1\} \to \{1, 3\}}$
$m = 4$,	$A = \{1, 4\}$,	$B = \{1, 3, 4\}$,	the "canonicity" test fails
$m = 3$,	$A = \{1, 3\}$,	$B = \{1, 3\}$,	$B'' = \{1, 3\}$
$m = 4$,	$A = \{1, 3, 4\}$,	$B = \{1, 3, 4\}$,	$B'' = \{1, 3, 4\}$
$m = 2$,	$A = \{1, 2\}$,	$B = \{1, 2, 3\}$,	$B'' = M$ $\boxed{\{1, 2, 3\} \to \{1, 2, 3, 4\}}$
$m = 4$,	$A = M$		

Note that Ganter's algorithm always generates the set of all intents as a by-product. As said above, this algorithm is not designed specifically for implications, but for an arbitrary closure operator. If, starting from A, it generates pseudo-intent B (which is closed with respect to the saturation operator), the next set it processes is the set following B according to the lectic order. However, if A is not a prefix of B'', then all supersets of A for which A is a prefix can be excluded from further consideration: they cannot be pseudo-closed or even quasi-closed since they will contain A, but not B'' in spite of $A'' = B''$. In this case, to avoid unnecessary computation, the next set for processing should be chosen as if B were not canonically generated. Then, the main if condition in the algorithm can be restated, for example, as follows:

Algorithm 3 (Ganter+)

```
...
    if there is no n ∈ B \ A such that n < m [the "canonicity" test]
      then
        if B ≠ B'' then Add B → B'' to Basis
        if there is no n ∈ B'' \ B such that n < m
          then
            A := B
            m := max({n | n ∈ M \ B})
          else m := max({n | n ∈ M \ A and n < m})
      else m := max({n | n ∈ M \ A and n < m})
...
```

We have implemented both versions of the algorithm and tested them in our experiments (Section 7); we refer to the second version as *Ganter+*.

4 Types of implications

To define an attribute-incremental procedure for generating the canonical basis, let us start with some required notation and properties that will save the effort of saturating all the candidate subsets as in the classical approach. Without loss of generality, assume that the attribute set consists of natural numbers starting from 1, i.e., $\mathbb{K} = (G, M = \{1, \ldots, |M|\}, I)$. For $x \in M$, the set M_x comprises all natural numbers less than or equal to x: $M_x = \{m \mid 0 < m \leq x\}$. Considering the context $\mathbb{K}_x = (G, M_x, I \cap G \times M_x)$, we denote its derivation operator by ^{x}, while keeping $'$ for the context $\mathbb{K}$. Concepts, extents, intents, pseudo-intents, etc. of the context $\mathbb{K}_x$ will be called x-concepts, x-extents, x-intents, x-pseudo-intents, etc.

Construction of the concept lattice $L(\mathbb{K}_{x+1})$ from the lattice $L(\mathbb{K}_x)$ is well understood [5, 7, 8, 11, 19, 21, 22]; see [15] for a survey. From practice, it turns out that some incremental algorithms (which, at the xth step, process x first attributes or objects) are more efficient in constructing the lattice of the context from scratch than their batch counterparts.

Ganter's algorithm builds the concept lattice $L(\mathbb{K})$ and the implication basis $B(\mathbb{K})$ for the whole context. We propose a new algorithm that, receiving as input $L(\mathbb{K}_x)$, $B(\mathbb{K}_x)$, and $\mathbb{K}_{x+1}$, builds $L(\mathbb{K}_{x+1})$ and $B(\mathbb{K}_{x+1})$.

The main difficulty Ganter's algorithm has to overcome when building the implication basis (as opposed to building the concept lattice) is the need to compute the saturation of attribute sets. This is a time-consuming operation because of the reiteration ("**while** $OldClosure \neq NewClosure$" in the *Saturate* function above), and, if possible, it is better to be avoided by testing the condition of being pseudo-closed *locally*. With this in mind, we identify several types of implications and process each type separately.

We call a set $A \subseteq M_x$ *x-modified* if $x \in (A \setminus \{x\})^{xx}$ and *x-stable* otherwise (cf. modified and old concepts in [11]). Thus, A is x-modified if and only if the implication $A \setminus \{x\} \to x$ holds in $\mathbb{K}$. An implication is *x-modified* if its premise is x-modified and *x-stable* otherwise. The difference between x-stable and x-modified sets is that the closure of an x-stable set does not change when moving from the context K_{x-1} to K_x, whereas the closure of x-modified set includes the new element x.

Example 6 Consider again the context from Example 1. The set $\{1, 2, 3, 4\}$ is 4-modified, since $\{1, 2, 3\} \to \{4\}$. Note also that $\{1, 2, 3\}^{44} = \{1, 2, 3, 4\}$. Another example of a 4-modified set is $\{1, 2\}$, which is the premise of the 4-modified implication $\{1, 2\} \to \{3\}$. On the other hand, $\{2, 3\}^{44} = \{2, 3\}$; therefore, $\{2, 3\}$ and $\{2, 3, 4\}$ are 4-stable.

Separate processing of modified and stable implications is the key principle behind our algorithm. Let us formulate a few important properties of modified and stable sets. From now on, we use y as a shortcut for $x + 1$.

Proposition 7 *Subsets of x-stable sets are x-stable for any* $x \in M$.

Proposition 8 *Supersets of x-modified sets are x-modified for any* $x \in M$.

Proposition 9 *For* $A \subseteq M_x$ *(and* $y = x + 1$*),* $A^{yy} = A^{xx}$ *if and only if A is y-stable.* $A^{yy} = A^{xx} \cup \{y\}$ *if and only if A is y-modified.*

Corollary 10 $A^{xx} = A^{yy} \setminus \{y\}$ *for any* $A \subseteq M_x$.

Proposition 11 *The set* $A \subseteq M_x$ *is y-modified if and only if* A^{xx} *is y-modified.*

Corollary 12 *The set* $A \subseteq M_x$ *is y-stable if and only if* A^{xx} *is y-stable.*

Thus, y-stable subsets of M_x form an order ideal of the Boolean lattice $(\mathfrak{P}(M_x), \subseteq)$ bounded above by x-intents. All other elements of this lattice are y-modified sets. Note that, according to Proposition 11, elements of the same x-closure class are either all y-modified or all y-stable.

Example 13 (Boolean lattice of attribute subsets of M_3*)* See Fig. 1.

We consider four types of implications for $y = x + 1$:

$A \to B$	$x+1 \notin A$	$x+1 \in A$
$(x+1)$**-stable**	old stable	new stable
$(x+1)$**-modified**	old modified	new modified

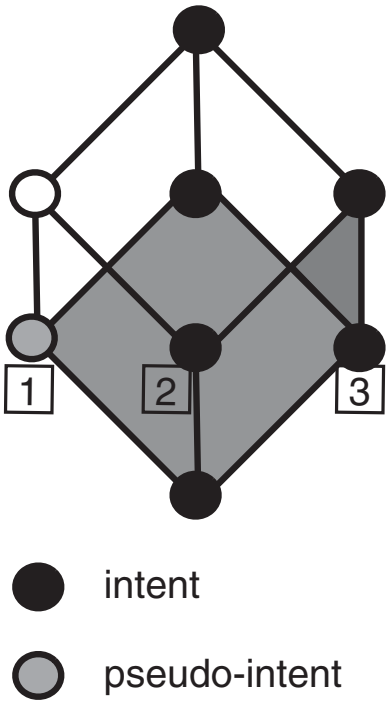

Fig. 1 The attribute subset corresponding to a node consists of attributes that label this node and all nodes to which there is a downward path from this node. *Dark-colored nodes* are intents of the context $\mathbb{K}_3$, while *light-colored nodes* are its pseudo-intents (see Example 1). 4-stable sets form an order ideal of this lattice (*greyed*)

Proposition 14 *For $A, B \subseteq M_x$ (and $y = x + 1$), the implication $A \to B$ holds in the context $\mathbb{K}_x$ if and only if it holds in $\mathbb{K}_y$.*

Proposition 15 *For $A, B \subseteq M_x$, the implication $A \to B \cup \{y\}$ holds in the context $\mathbb{K}_y$ if and only if $A \to B$ holds in the context $\mathbb{K}_x$ and A is a y-modified set.*

Propositions 14 and 15 cover all implications of the context $\mathbb{K}_y$ with premises that are subsets of M_x. Which of these premises are pseudo-closed?

Proposition 16 *A set $A \subseteq M_y$ is y-quasi-closed only if $A \setminus \{y\}$ is x-quasi-closed.*

Proof Take a y-quasi-closed set $A \subseteq M_y$; show that $A \setminus \{y\}$ is x-quasi-closed. For every $B \subseteq M_x$, $B^{xx} \subseteq B^{yy}$. Consider an arbitrary $B \subset A \setminus \{y\}$. Either $B^{yy} \subset A$ or $B^{yy} = A^{yy}$. In the former case, $B^{xx} \subset A$, and $B^{xx} \subseteq A \setminus \{y\}$, as $y \notin B^{xx}$. In the latter case, $(A \setminus \{y\})^{xx} \supseteq B^{xx} = B^{yy} \setminus \{y\} = A^{yy} \setminus \{y\} \supseteq (A \setminus \{y\})^{xx}$; therefore, $B^{xx} = (A \setminus \{y\})^{xx}$. □

Lemma 17 *A y-stable set $A \subseteq M_x$ is y-pseudo-closed if and only if A is x-pseudo-closed.*

Proof As A is y-stable, every of its subsets is also y-stable. Consequently, $\forall B \subseteq A : B^{yy} = B^{xx}$, and, by definition, A is x-pseudo-closed if and only if it is y-pseudo-closed. □

This means that all y-stable y-pseudo-closed subsets of M_x are premises of implications from $B(\mathbb{K}_x)$, which suggests that y-modified implications should be separated from y-stable implications in $B(\mathbb{K}_x)$. As we will see later, this idea makes sense.

Example 18 See Fig. 2.

We know all y-stable implications of $B(\mathbb{K}_y)$ with premise from M_x and now consider their y-modified counterparts.

Lemma 19 *A y-modified set $A \subseteq M_x$ is a y-pseudo-intent if and only if A is minimal among y-modified x-pseudo-intents and x-intents.*

Proof Note that being minimal among y-modified x-pseudo-intents and x-intents is equivalent to being minimal among y-modified x-quasi-closed sets.

Let A be minimal among y-modified x-quasi-closed sets. If A is not a y-pseudo-intent, then there is a y-pseudo-intent $B \subset A$ such that $B^{yy} \not\subset A$. By Proposition 16, B is x-quasi-closed. Since A is minimal, B is y-stable. Then, by Proposition 9, $B^{xx} = B^{yy} \not\subset A$. From A being x-quasi-closed, it follows that $B^{xx} = A^{xx}$. Hence, B^{xx} is y-modified, which is impossible due to B being y-stable and Proposition 11.

Now, if y-modified $A \subseteq M_x$ is a y-pseudo-intent, then A is x-quasi-closed by Proposition 16. Let $B \subseteq A$ be minimal among y-modified x-quasi-closed sets. As

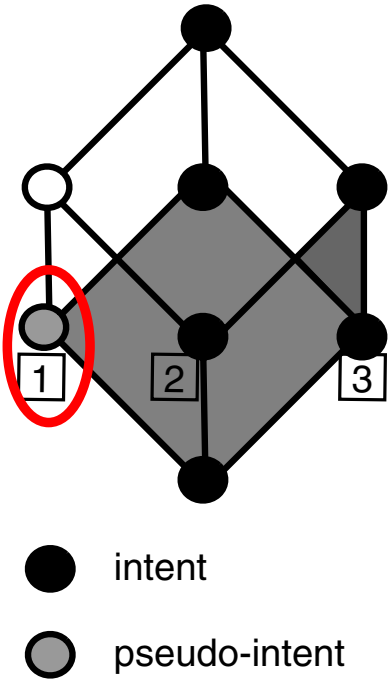

Fig. 2 The only 4-stable pseudo-intent in $\mathbb{K}_3$ (see Example 1) is $\{1\}$. Hence, $\{1\} \to \{3\}$, the only implication in the basis of $\mathbb{K}_3$, is 4-stable, and it keeps its place in the basis of $\mathbb{K}_4$

shown above, B is a y-pseudo-intent. However, $y \in B^{yy} \not\subset A$. Hence, $B = A$, which concludes the proof. □

Example 20 See Fig. 3.

Let us consider y-pseudo-closed sets from M_y that contain y.

Proposition 21 *If y-stable $A \subseteq M_y$ is y-quasi-closed and $y \in A$, then $A \setminus \{y\}$ is x-closed.*

Proof From A being y-stable, it follows that $A \setminus \{y\}$ is y-stable and $(A \setminus \{y\})^{yy} = (A \setminus \{y\})^{xx} \neq A^{yy}$. Taking into account that A is y-quasi-closed, we obtain that $A \setminus \{y\} \subseteq (A \setminus \{y\})^{xx} \subset A$. Consequently, $A \setminus \{y\} = (A \setminus \{y\})^{xx}$, which proves the proposition. □

Lemma 22 *A y-stable set $A \subseteq M_y$ with $y \in A$ is y-pseudo-closed if and only if*

(1) *A is not y-closed;*
(2) *$A \setminus \{y\}$ is x-closed;*
(3) *For each y-stable y-pseudo-intent $B \subset A$: if $y \in B$, then $B^{yy} \subset A$.*

Proof If A is y-pseudo-closed, then (1) and (3) are satisfied by definition and (2) holds by Proposition 21. In the other direction, every subset of A is y-stable. If $B \subset A$ and $y \notin B$, then $B^{yy} = B^{xx} \subseteq (A \setminus \{y\})^{xx} = A \setminus \{y\} \subset A$. Hence, (3) ensures

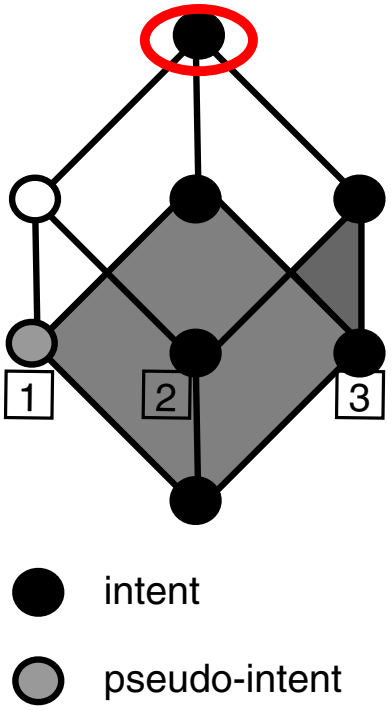

Fig. 3 There is only one 4-modified 3-intent, {1, 2, 3}, and no 4-modified 3-pseudo-intents in Example 1. This adds the 4-modified implication {1, 2, 3} → {4} to the basis of $\mathbb{K}_4$

that A contains the closure of every its y-pseudo-closed subset, which together with (1) means that A is y-pseudo-closed. □

Lemma 22 restricts the search space of pseudo-intents by indicating an efficient method for their construction: simply add the new attribute to an old intent. Even more interesting, Lemma 22 makes it possible to determine if the candidate for being a pseudo-intent is saturated using only y-stable implications whose premise contains y, i.e., those generated by Lemma 22 itself (*new stable implications*, as they are called above). Note that we only have to check whether A is saturated, rather than actually compute $A^\bullet$ (the saturation of A), which is much harder. Thus, separate storage of stable and modified implications will make saturation more efficient.

Example 23 See Fig. 4. Note that the set {1, 2, 3} has double coloring in Fig. 4: it is a 3-intent and a 4-pseudo-intent.

It remains to consider y-modified sets $A \subseteq M_y$ with $y \in A$. Any y-modified y-pseudo-intent can be obtained from a y-modified x-pseudo-intent in this or that way: apparently, all the information conveyed by a y-modified implication $A \to B$ must already be present in $B(\mathbb{K}_x)$ except for the information conveyed by the implication $A \to y$.

Assume that $^\bullet$ is the saturation operator in the context $\mathbb{K}_y$.

Lemma 24 *A y-modified set $A \subseteq M_y$ with $y \in A$ is y-pseudo-closed if and only if $A^{yy} \neq A = B^\bullet$ for some y-modified x-pseudo-closed $B \subseteq A \setminus \{y\}$.*

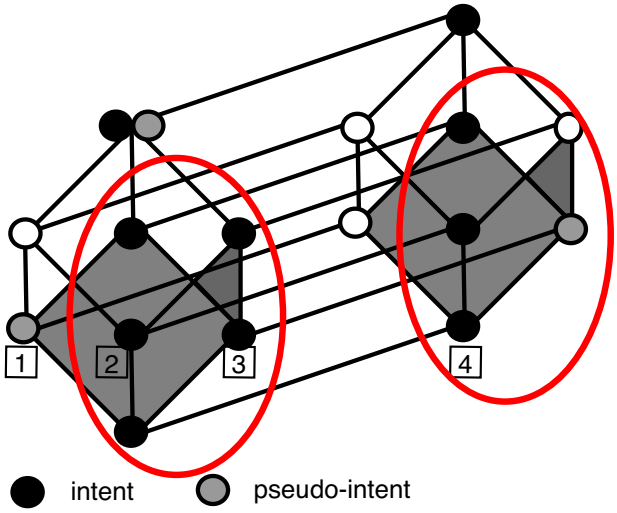

Fig. 4 Consider again Example 1. To detect 4-stable pseudo-intents containing attribute 4, we add 4 to all 4-stable 3-intents of $\mathbb{K}_3$. 3-intents Ø, {2}, and {1, 3} generate 4-intents {4}, {2, 4}, and {1, 3, 4}. The other sets generated by 3-intents are {3, 4} and {2, 3, 4}, which are not 4-intents. Obviously, {3, 4} satisfies condition (3) of Lemma 22 (there are no 4-stable 4-pseudo-closed subsets of {3, 4}), and it is a 4-pseudo-intent. Hence, we add the implication $\{3, 4\} \to \{1, 3, 4\}$ to the basis. Now, {3, 4} is a subset of {2, 3, 4}, but its closure, {1, 3, 4} is not. Consequently, {2, 3, 4} is not 4-pseudo-closed

Proof The sufficiency is immediate from the definition of the saturation operator. Let us prove the necessity. Suppose that a y-modified $A \subseteq M_y$ with $y \in A$ is y-pseudo-closed. Since A is y-modified, $A^{yy} = (A \setminus \{y\})^{xx} \cup \{y\}$. Therefore, as A is not y-closed, $A \setminus \{y\}$ is not x-closed. However, $A \setminus \{y\}$ is x-quasi-closed by Proposition 16. This implies that there is an x-pseudo-intent $B \subseteq A \setminus \{y\}$ such that $B^{yy} = (A \setminus \{y\})^{yy} = A^{yy}$. As $B \subset A$ and A is a y-pseudo-intent, there are no y-pseudo-intents or y-intents, i.e., sets closed under $^\bullet$, between B and A. Therefore, $B^\bullet = A$. □

Lemma 24 makes it possible to find all y-modified y-pseudo-intents $A \subseteq M_y$ with $y \in A$ by saturating y-modified x-pseudo-intents.

Example 25 In Example 1, we are lucky enough to have no 4-modified 3-pseudo-intents. Note that unlike in **Next Closure**, we do not have to consider the set {1, 4} as a candidate, since {1} is a 4-stable 4-pseudo-intent. This suggests that on a big context we can greatly reduce the time spent on saturation, which is a very computationally expensive operation.

The four lemmas lead us to the following theorem:

Theorem 26 *There is an injection from $B(\mathbb{K}_{x+1})$ to $L(\mathbb{K}_x) \cup B(\mathbb{K}_x)$. In other words, every pseudo-intent of $\mathbb{K}_{x+1}$ corresponds to a distinct intent or pseudo-intent in $\mathbb{K}_x$. Therefore, $|B(\mathbb{K}_{x+1})| \leq |L(\mathbb{K}_x)| + |B(\mathbb{K}_x)|$.*

5 Incremental construction of the canonical basis

In this section, we translate the results above into an algorithm constructing the canonical implication basis incrementally. We give a precise and compact description of the algorithm due to some simple though important precautions that have to be taken care of in the way of evaluating the candidate premises, so that the interested reader can adopt and develop this approach in his or her own procedures.

In the pseudo-code below, brackets [] denote lists. Concepts are pairs of sets (*Extent*, *Intent*), while implications are represented by triples of sets (*Extent*, *Premise*, *Consequence*), where *Premise* is a pseudo-intent, *Consequence* is the consequence of the implication that has *Premise* as its premise (i.e., when the algorithm terminates, $Consequence = Premise''$), $Extent = Premise'$ is the set of objects whose intents contain *Premise*. Using dot notation, we write, e.g., *concept.Intent* to designate the intent of *concept*. All function parameters are in/out: a function call may change the value of the variable passed as its argument.

The algorithm is attribute-incremental, i.e., it processes the input context $\mathbb{K}$ attribute after attribute constructing the implication basis (and the concept set) for each context $\mathbb{K}_x$ with x varying from 1 to $|M|$. The algorithm maintains the *Elements* list containing concepts and implications of the basis ordered in a certain way. We use y to denote the attribute being processed assuming that the *Elements* list contains concepts and implications of the context $\mathbb{K}_x$ and $y = x + 1$.

Algorithm 4 (Incremental algorithm)

Input A context $\mathbb{K} = (G, M, I)$.
Output The canonical implication basis of $\mathbb{K}$.
Begin
  Elements := $[(G, \emptyset)]$
  $N := \emptyset$
  for each y in M
    $N := N \cup \{y\}$
    AddAttribute(y, *Elements*, N)
  return the set of all implications in *Elements*
End

The algorithm goes through the *Elements* list processing each element according to its type. As the output, we get the four types of implications described by the lemmas. Our algorithm maintains a separate set for each type of implications:

$A \to B$	$y \notin A$	$y \in A$
y-**stable**	*OldStableImpl* (Lemma 17)	*NewStableImpl* (Lemma 22)
y-**modified**	*MinModImpl* (Lemma 19)	*NonMinModImpl* (Lemma 24)

Lemmas 17, 19, 22, and 24 provide a way to obtain all intents and pseudo-intents of the context $\mathbb{K}_y$ from intents and pseudo-intents of the context $\mathbb{K}_x$. Thus, we have to deal with four types of "generators" of y-pseudo-intents: y-stable and y-modified

x-intents and x-pseudo-intents. The following table shows which lemmas should be consulted when processing different types of generators:

	x-pseudo-intent	x-intent
y-**stable**	Lemma 17	Lemma 22
y-**modified**	Lemma 19 and 24	Lemma 19

Many y-pseudo-intents are generated by immediate application of Lemmas 17, 19, and 22 to x-intents and x-pseudo-intents. However, Lemma 24 suggests that some y-modified x-pseudo-intents rejected by Lemma 19 can still prove useful in obtaining y-pseudo-intents that contain the new attribute. These x-pseudo-intents, which are not minimal in the sense of Lemma 19, are placed into the *NonMinModImpl* set. In the end of the incremental step, when we have a cover of the new basis $B(\mathbb{K}_y)$ available, the x-pseudo-intents are saturated by the *Fuse* procedure (see below) and either accepted as y-pseudo-intents or ultimately discarded.

In the *Elements* list, a concept with a smaller intent or an implication with a smaller premise must precede a concept with a larger intent or an implication with a larger premise. This is the basis of the algorithmic steps involving minimality checks

Algorithm 5

AddAttribute(y, *Elements*, N)
Input New attribute y and its extent y',
list *Elements* consisting of all concepts of $L(\mathbb{K}_x)$ and implications of $B(\mathbb{K}_x)$,
set N of all attributes already processed ($N = M_x$)
Output *Elements* consists of all concepts of $L(\mathbb{K}_y)$ and implications of $B(\mathbb{K}_y)$
Begin
OldStableImpl := ∅
NewStableImpl := ∅
MinModImpl := ∅
NonMinModImpl := ∅
ModConcepts := ∅
for each *element* in *Elements*
if *element*.Extent ⊆ y'
then
if *element* is a concept
then ProcessModifiedConcept(y, *element*, *Elements*, *MinModImpl*, *ModConcepts*);
else ProcessModifiedImplication(y, *element*, *Elements*, *MinModImpl*, *NonMinModImpl*);
else
if *element* is a concept
then ProcessStableConcept(y, *element*, N, *NewStableImpl*);
else ProcessStableImplication(*element*, *OldStableImpl*);
Fuse(*OldStableImpl* ∪ *NewStableImpl* ∪ *MinModImpl*, *Elements*, *NonMinModImpl*, *ModConcepts*);
End

and saturation that occur in relation to Lemmas 19 and 22: e.g., to check if set A is saturated, one should already have all implications from the basis with premises that are subsets of A. Maintaining such an order does not involve a significant computation overhead.

Let us consider how the algorithm processes the four types of generators.

1. According to Lemma 17, y-stable x-pseudo-intents are y-pseudo-intents; thus, no special processing is required.

Algorithm 6

ProcessStableImplication(*implication*, *OldStableImpl*)
Begin
  Add *implication* to *OldStableImpl*
End

2. A y-stable x-intent can serve a basis for a y-pseudo-intent A with $y \in A$ if it satisfies the conditions of Lemma 22, which are not difficult to check.

Algorithm 7

ProcessStableConcept(y, *concept*, *CurrentAttributes*, *NewStableImpl*)
Begin
  $NewExt := concept.\text{Extent} \cap y'$
  $NewPrem := concept.\text{Intent} \cup \{y\}$
  $NewCons := \{n \mid n \in CurrentAttributes \text{ and } NewExt \subseteq n'\}$
  if $NewCons = NewPrem$
    then
      $new\_concept := (NewExt, NewPrem)$
      Add *new_concept* to *Elements*
    else
      if $NewPrem = Saturate(NewPrem, NewStableImpl)$
        $new\_impl := (NewExt, NewPrem, NewCons)$
        Add *new_impl* to *NewStableImpl* and to *Elements*
End

The *CurrentAttributes* set contains all attributes processed at the point of the procedure call, i.e., y and all the preceding attributes. The extent of the attribute y is denoted by y', as, for any k and $l \geq y$, $\{y\}^k = \{y\}^l$. Thus, the *ProcessStableConcept* procedure creates a new attribute set *NewPrem* adding the attribute y to a y-stable intent. If *NewPrem* is y-closed, a new concept is added to *Elements*. Otherwise, *NewPrem* is tested for being pseudo-closed. According to Lemma 22, it suffices to verify that *NewPrem* is saturated by the implications from *NewStableImpl*. If so, the implication $NewPrem \rightarrow NewCons$ is added to *NewStableImpl* and to *Elements*.

There is no need in actual computation of the saturation of *NewPrem*: it is enough to check whether *NewStableImpl* contains an implication whose premise is a subset of *NewPrem* and whose consequence is not. This test can be performed in a number of steps linear in the number of implications in

NewStableImpl, which is generally much more efficient than the computation of the saturation. However, *NewStableImpl* should already contain all relevant implications of the y-basis whose premises are subsets of *NewPrem*. That is why the order of processing is important.

3. $(x+1)$-modified x-pseudo-intents demand special attention, as they give rise to pseudo-intents of two types described by Lemmas 19 and 24.

Algorithm 8

ProcessModifiedImplication(y, *implication*, *Elements*, *MinModImpl*, *NonMinModImpl*)
Begin
Add y to *implication*.Consequence;
if there is *min* $\in$ *MinModImpl* such that *min*.Premise $\subseteq$ *implication*.Premise
then
Add y to *implication*.Premise
Move *implication* from *Elements* to *NonMinModImpl*
else Add *implication* to *MinModImpl*
End

If an implication is y-modified, y should be added to its consequence. First, we check the conditions of Lemma 19. They are satisfied if and only if *MinModImpl* does not contain an implication whose premise is a subset of *implication*.Premise (assuming that smaller x-pseudo-intents and x-intents are processed before larger ones). In this case, *implication*.Premise is a minimal y-modified y-pseudo-intent, and *implication* should be added to *MinModImpl*. Otherwise, it is necessary to compute the saturation of *implication*.Premise, which can be y-pseudo-closed according to Lemma 24. However, we will be in a position to do it only when the set of implications equivalent to $B(\mathbb{K}_y)$ becomes available; therefore, we put implication into *NonMinModImpl* and leave it there for a while.

4. $(x+1)$-modified x-intents are responsible only for implications described by Lemma 19.

Algorithm 9

ProcessModifiedConcept(y, *concept*, *Elements*, *MinModImpl*, *ModConcepts*)
Begin
if there is *min* $\in$ *MinModImpl* such that *min*.Premise $\subseteq$ *concept*.Intent
then Remove *concept* from Elements
else
new_impl := (*concept*.Extent, *concept*.Intent, *concept*.Intent $\cup\{y\}$)
Add *new_impl* to *MinModImpl*
Replace *concept* with *new_impl* in *Elements*
Add y to *concept*.Intent
Add *concept* to *ModConcepts*
End

Similar to the preceding case, we check the minimality. If *concept*.Intent is a minimal y-modified y-pseudo-intent, the new implication with the premise *concept*.Intent is substituted for *concept* in the *Elements* list and is added to *MinModImpl*. As for *concept*, it is placed to the end of *ModConcepts* whether or not the conditions of Lemma 19 are satisfied. This is necessary to maintain the desired order.

Thus, we have four implication sets: *OldStableImpl*, *NewStableImpl*, *MinModImpl*, and *NonMinModImpl*. The first three sets contain all implications of the basis (and only them) except for those described by Lemma 24. Implications from these sets are already in *Elements*. At the last step of the algorithm, implication premises from *NonMinModImpl* are saturated and the implications together with modified concepts from *ModConcepts* are added to *Elements*.

Algorithm 10

```
Fuse(Basis, Elements, ExtraImpl, ExtraElements)
Begin
  for each impl in ExtraImpl
    Remove impl from ExtraImpl
    OtherImpl := Basis ∪ ExtraImpl
    impl.Premise := Saturate(impl.Premise, OtherImpl)
    if impl.Premise ⊂ impl.Consequence
      then add impl to Basis and to ExtraElements
  Sort ExtraElements
  Add ExtraElements to the end of Elements
End
```

To prove that the algorithm is correct it remains to show that, at each point of execution of the algorithm, concepts and implications in the *Elements* list are arranged according to the order $\subseteq$ of their intents and premises, respectively.

Before the *Fuse* procedure is called, the *Elements* list is extended only by new pseudo-intents and intents obtained by processing stable x-intents that are supposed to be in the right order. If A is an x-intent, then adding y-pseudo-intent or y-intent $A \cup \{y\}$ to the end of the list does not violate the order. Indeed, for any A and B, $y \notin A$, $y \notin B$: $B \subseteq A \Leftrightarrow B \cup \{y\} \subseteq A \cup \{y\}$.

At the moment when the *Fuse* procedure is called, the *Elements* list contains all y-stable y-pseudo-intents and y-intents, as well as y-modified pseudo-intents A such that $y \notin A$. The *Fuse* procedure adds to *Elements* y-modified y-intents and y-pseudo-intents A such that $y \in A$. Obviously, such A cannot be a subset of a pseudo-intent or an intent from the *Elements* list, since y-modified sets cannot be subsets of y-stable sets and sets that contain y cannot be subsets of sets that do not contain y. Therefore, sorting *ExtraElements* before adding them to *Elements* is sufficient to preserve the order.

Let us estimate the complexity of the algorithm. Since the concept set and canonical basis are, in general, of exponential size with respect to the number of objects and attributes [14], we use the number of intents and pseudo-intents of the "old" context $\mathbb{K}_x$ to estimate the complexity of the incremental step. We denote $l = |L(\mathbb{K}_x)|$ and $b = |B(\mathbb{K}_x)|$. Since there are at most as many new stable implications

as there are stable concepts in $L(\mathbb{K}_x)$, the complexity of *ProcessStableConcept* is bounded by $O(y|G| + yl)$: as $y = |Current Attributes|$, the complexity of computing the closure *NewCons* is $O(y|G|)$, and $O(yl)$ is the time needed to check whether set *New Prem* is saturated. We estimate the complexity of *ProcessModifiedImplication* and *ProcessModifiedConcept* as $O(y(l + b))$, given that $|MinModImpl| \leq l + b$ (see Theorem 26). Taking into account the if condition in the main loop of *AddAttribute*, one can see that the complexity of the main loop, which runs $l + b$ times, is bounded by $O(y|G|(l + b) + y(l + b)^2)$.

Implications from *ExtraImpl* in the *Fuse* procedure have been generated by *ProcessModifiedImplication*; hence, their number is limited by the number of pseudo-intents in the original context, i.e., by b. Similarly, the number of concepts in *ExtraElements* is limited by l. If properly implemented, e.g., using LinClosure [16] instead of *Saturate* and the QuickSort algorithm for sorting *ExtraElements*, the *Fuse* procedure will not add anything to the complexity of the main loop. Therefore, the complexity of *AddAttribute* (an incremental step) is bounded by $O(y|G|(l + b) + y(l + b)^2)$, where y is the number of attributes processed so far, $l = |L(\mathbb{K}_x)|$, and $b = |B(\mathbb{K}_x)|$.

6 An example

Let us illustrate the work of the algorithm with the context from Example 1. Table rows correspond to iterations of the main loop of the *AddAttribute* procedure. The y column records the new attribute, and the *element* column contains the concept or implication processed at this iteration. Every iteration has an identifier #. The *Elements* cell contains the concept or implication added to the *Elements* list at the iteration; it is empty if nothing has been added. The # identifier in the *Elements* cell

#	y	*element*	*Old Stable Impl*	*New Stable Impl*	*Min Mod Impl*	*Mod Concepts*	*Elements*
0							(abcd, Ø)
1	1	(abcd, Ø)					(ab, 1) #3.2
		stable concept					**(ab, 1, 13)**
2.1	2	(abcd, Ø)					(cd, 2)
		stable concept					
2.2		(ab, 1)					(Ø, 12)
		stable concept					#3.4
3.1	3	(abcd, Ø)					(abc, 3)
		stable concept					
3.2		(ab, 1)			(ab, 1, 13)	(ab, 13)	
		modified concept					
3.3		(cd, 2)					(c, 23)
		stable concept					

continued on next page

continued from previous page

#	y	*element*	*Old Stable Impl*	*New Stable Impl*	*Min Mod Impl*	*Mod Concepts*	*Elements*
3.4		(Ø, 12) modified concept				(Ø, 123)	
							(ab, 13)
			Fuse				(Ø, 123) #4.7 **(Ø, 123, 1234)**
4.1	4	(abcd, Ø) stable concept					(ad, 4)
4.2		(ab, 1, 13) stable implication	(ab, 1, 13)				
4.3		(cd, 2) stable concept					(d, 24)
4.4		(abc, 3) stable concept		(a, 34, 134)			**(a, 34, 134)**
4.5		(c, 23) stable concept					
4.6		(ab, 13) stable concept					(a, 134)
4.7		(Ø, 123) modified concept			(Ø 123, 1234)	(Ø, 1234)	
			Fuse				(Ø, 1234)

indicates that the generated element has been removed from the list at the iteration denoted by the identifier. A table cell is split horizontally if the element has been replaced by the element below. In our example, the *NonMinModIml* list is always empty; hence, there is no such column in the table. When writing sets, we omit commas and braces; i.e., we write 12 instead of {1, 2}, ab instead of {*a*, *b*}, etc.

7 Experiments

We programmed Ganter's algorithm and our incremental algorithm in C# under the .NET platform. Below, we give the time the algorithms spend on various contexts, both real and artificial (if necessary, transformed into binary form using FCA scaling techniques [10]). Even if these results are somewhat preliminary, and the real difference in performance might be smaller after proper optimization, the second numbers in the columns of algorithms suggest that, indeed, there should

be considerable gain in performance.[2] These are the total numbers of implications participating in saturation, that is, the size of all *Impl* sets across all calls to the *Saturate* procedure (see above).

Context	Number of concepts	Size of the basis	Ganter	Ganter+	Incremental algorithm
$(M, M, \neq)$, $\lvert M\rvert = 18$	262144	0	1.7 sec; 0	1.6 sec; 0	4.3 sec; 0
$\lvert G\rvert = 10$, $\lvert M\rvert = 100$, $\lvert g'\rvert = 25$	129	380	7.9 sec; 4 815 975	4.3 sec; 2 391 767	0.7 sec; 101 089
$\lvert G\rvert = 20$, $\lvert M\rvert = 100$, $\lvert g'\rvert = 25$	716	2269	4 min 1 sec; 141 306 696	1 min 39 sec; 55 284 673	14.4 sec; 1 784 027
$\lvert G\rvert = 10$, $\lvert M\rvert = 100$, $\lvert g'\rvert = 50$	559	546	23.2 sec; 16 664 073	18.1 sec; 12 788 166	2.5 sec; 628 517
SPECT: $\lvert G\rvert = 267$, $\lvert M\rvert = 23$, see [4]	21550	2169	2 min 42 sec; 113 690 618	2 min 30 sec; 108 125 700	9.4 sec; 3 890 328
Solar Flare: $\lvert G\rvert = 1389$, $\lvert M\rvert = 49$, see [4]	28742	3382	6 min 26 sec; 108 792 590	5 min 56 sec; 86 593 347	1 min 15 sec; 6 474 343
Congressional Voting: $\lvert G\rvert = 435$, $\lvert M\rvert = 18$, see [4]	10644	849	18.1 sec; 134 374 733	15.9 sec; 80 964 720	2 sec; 1 315 122

8 Conclusion

Since its introduction in the early eighties, one has had at disposal a single efficient algorithm — to our knowledge — for directly extracting the canonical basis of implications holding in a context, namely, **Next Closure**. This algorithm can be used for scanning through any closure operator, being therefore not specifically tailored for the basis extraction, and is somehow slowed down in this situation by the necessity of calculating many a "premise saturation," which is a slow iterative process. Here, we have presented a new approach that specifically addresses the introduction of

[2] We have also implemented the incremental algorithm in Java within Sergey Yevtushenko's Concept Explorer framework and compared its performance with Concept Explorer's implementation of Ganter's algorithm. The results are very similar to those presented in this paper. Concept Explorer is available at http://sourceforge.net/projects/conexp.

a "new" attribute in a context and, thus, avoids recalculating the "new" basis from scratch by undertaking a revision of the "old" basis instead. Therefore, it is attribute-incremental and is useful in refining a database by introducing new attributes to object descriptions. It turns out to be quite competitive due to some "genealogic" properties of implications that decrease the number of premise saturations (but assume storage of all intents in return). An algorithm updating the implication basis with the addition of a new object is a subject of further research.

Since both the concept set and the implication basis can be of exponential size with respect to the context size in the worst case [14], any algorithm for basis construction has necessarily exponential time complexity. However, in general, the exponential sizes of the concept set and implication basis may be two different worst cases. In practice, for an arbitrary closure operator coming from a database, the number of pseudo-closed elements might be small regarding the number of intents (closed elements), which makes it undesirable to carry over the latter when one only needs to generate the former. A challenge would be to devise an efficient algorithm whose complexity depends only on the size of the basis.

Acknowledgements The authors would like to thank Centre National de la Recherche Scientifique (CNRS) and Russian State University for the Humanities for providing the invitations and travel facilities for the collaboration. The first author was also supported by the International Quality Network (IQN), DAAD, and Maison des Sciences de l'Homme, Paris. Part of the work was done while the first author was visiting the Dresden University of Technology (August–October 2002, June 2003–February 2004). Figures of lattices are obtained using the program Concept Explorer by Sergey Yevtushenko.

References

1. Armstrong, W.W.: Dependency structure of data base relationships. In: Proc. IFIP Congress. Geneva, Switzerland, pp. 580–583 (1974)
2. Ausiello, G., D'Atri, A., Saccà, D.: Minimal representation of directed hypergraphs. SIAM J. Comput. **15**(2), 418–431 (1986)
3. Birkhoff, G.: Lattice Theory. Amer. Math. Soc. Coll. Publ., Providence, R.I. (1973)
4. Blake, C.L., Merz, C.J.: UCI Repository of machine learning databases, Irvine, CA, University of California, Department of Information and Computer Science, Web Page http://www.ics.uci.edu/ mlearn/MLRepository.html (1998)
5. Carpineto, C., Romano, G.: A lattice conceptual clustering system and its application to browsing retrieval. Mach. Learn. **24**(2), 95–122 (1996)
6. Day, A.: The lattice theory of functional dependencies and normal decompositions. Int. J. Algebra Comput. **2**(4), 409–431 (1992)
7. Dowling, C.E.: On the irredundant generation of knowledge spaces. J. Math. Psychol. **37**(1), 49–62 (1993)
8. Ferré, S.: Incremental concept formation made more efficient by the use of associative concepts, INRIA Research Report no. 4569 (2002)
9. Ganter, B.: Two Basic Algorithms in Concept Analysis, FB4-Preprint No. 831, TH Darmstadt (1984)
10. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer, Berlin (1999)
11. Godin, R., Missaoui, R., Alaoui, H.: Incremental concept formation algorithms based on Galois lattices. Comput. Intell. **11**(2), 246–267 (1995)
12. Guigues, J.-L., Duquenne, V.: Familles minimales d'implications informatives resultant d'un tableau de données binaires. Math. Sci. Hum. **95**, 5–18 (1986)
13. Liquière, M., Ganter, B., Duquenne, V., Mephu Nguifo, E., Stumme, G.: ECAI 2002 International Workshop on Advances in Formal Concept Analysis for Knowledge Discovery in Databases. Lyon, France (2002)

14. Kuznetsov, S.: On the intractability of computing the Duquenne-Guigues base. J. Univers. Comput. Sci. **10**(8), 927–933 (2004)
15. Kuznetsov, S., Obiedkov, S.: Comparing performance of algorithms for generating concept lattices. J. Exp. Theor. Artif. Intell. **14**(2–3), 189–216 (2002)
16. Maier, D.: The Theory of Relational Databases. Computer Science, Rockville, MD (1983)
17. Mannila, H., Räihä, K.J.: The Design of Relational Databases. Addison Wesley, Boston, MA (1992)
18. Mephu Nguifo, E., Duquenne, V., Liquière, M.: ICCS-2001 International Workshop on Concept Lattices-Based Theory, Methods and Tools for Knowledge Discovery in Databases. Stanford CA, USA (2001)
19. Norris, E.M.: An algorithm for computing the maximal rectangles in a binary relation. Rev. Roum. Math. Pures. Appl. **23**(2), 243–250 (1978)
20. Taouil, R., Bastide, Y.: Computing proper implications. In: Proc. ICCS-2001 International Workshop on Concept Lattices-Based Theory, Methods and Tools for Knowledge Discovery in Databases, pp. 49–61. Stanford CA, USA (2001)
21. Valtchev, P., Missaoui, R.: Building concept (Galois) lattices from parts: Generalizing the incremental methods. In: Proc. ICCS 2001, Lecture Notes in Artificial Intelligence, vol. 2120, pp. 290–303. Springer, Berlin (2001)
22. Van Der Merwe, F.J., Obiedkov, S., Kourie, D.G.: AddIntent: A new incremental lattice construction algorithm. Concept Lattices: Proc. of the 2nd Int. Conf. on Formal Concept Analysis, Lecture Notes in Artificial Intelligence, vol. 2961, pp. 372–385. Springer, Berlin (2004)
23. Wild, M.: Implicational bases for finite closure systems. In: Lex, W. (ed.) Arbeitstagung, Begriffsanalyse und Künstliche Intelligenz, pp. 147–169 Informatik-Bericht 89/3, TU Clausthal (1991)
24. Wild, M.: Computations with finite closure systems and implications, Preprint-Nr: 1708, TH Darmstadt (1994)