Incremental Construction of the Canonical Implication Basis

S. Obiedkov* V. Duquenne†

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

1 Introduction

Recent years have seen an increased mutual interest between the communities of lattice theory researchers [2], especially those working within the framework of formal concept analysis (FCA) [8], and developers of intelligent data analysis systems. This is attested by the workshops held at Stanford [15] and in Lyon [12], as well as special issues of JETAI [16] and AAI [17]. 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 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 [10]; it is a minimal non-redundant implication set from which all implications valid in the dataset can be inferred by the Armstrong rules.

2 Formal Contexts and Implications

Recall that a (formal) context is a triple $K = (G, M, I)$, 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 [8].

Example 1 A formal context is usually visualized by a cross table:

<table>
<thead>
<tr>
<th>G \ M</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

$A' = \{ m \in M \mid \forall g \in A \ (g \ I \ m) \}$;

$B' = \{ g \in G \mid \forall m \in B \ (g \ I \ m) \}$.

The operator "'" is a closure operator (to be precise, "" is a homonymous denotation of two closure operators: $2^G \rightarrow 2^G$ and $2^M \rightarrow 2^M$). 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 $K$. The sets $A$ and $B$ are closed; they are called respectively the extent and the intent of the formal concept $(A, B)$.

An implication is an expression $A \rightarrow B$ where $A, B \subseteq M$. The implication $A \rightarrow B$ holds in the context $K$ 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. It is based on the notion of a pseudo-closed set.

A set $A \subseteq M$ is called quasi-closed (with respect to the closure operator ")" if $B'' \subseteq A$ or $B'' = A''$ for any $B \subseteq A$. 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 [8], 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 $B \subset A$.

The canonical implication basis of the context $K$ (notation: $B(K)$) consists of all implications $P \rightarrow P''$ where $P$ is a pseudo-intent of the context $K$ [10].

---

* Russian State University for the Humanities, Miusskaya pl. 6, 125267 Moscow (Russia). Mail: s-obj@yandex.ru
† CNRS UMR 7090, ECP6, 175 rue du Chevaleret, 75013 Paris (France). Mail: v.duquenne@wanadoo.fr
As opposed to 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 [7]. It is a modification of his NextClosure algorithm for computing the concept set of a context. There are also algorithms constructing other implication bases: direct basis [14, 22], proper basis [19], etc. The canonical basis could be obtained from the output of these algorithms using a certain procedure. Such approach is justified if the algorithm is fast, the size of its output is not large (with respect to \(2^{|M|}\)), and the “canonization” procedure is efficient.

The algorithm proposed below allows a step-by-step construction of the basis, processing a single attribute at a single step. Although a new attribute does not cancel old implications, it can arbitrarily change the set of pseudo-intents; hence, one cannot assume that the canonical basis will contain more implications after adding an attribute.

It is well known that closed and pseudo-closed sets form together a new closure system. We call the closure operator corresponding to this system the saturation operator and denote it by \(\bar{\cdot}\). It can be defined as follows:

\[
A^+ = A \cup \{B^+ \mid B \subset A \land B \text{ is pseudo-closed}\}; \\
A^*= A^{++*}.
\]

3 Types of Implications

Without loss of generality, assume that the context is of the form \(K = (G, M = \{1, \ldots, |M|\}, I\). For \(i \in [0, |M|], K_i = (G, M_i = \{m \mid 0 < m \leq i\}, I \cap (G \times M_i))\) is the context with the derivation operator \(\psi^i\). Concepts, extents, intents, pseudo-intents, etc. of the context \(K\), will be called \(i\)-concepts, \(i\)-extents, \(i\)-intents, \(i\)-pseudo-intents, etc.

Construction of the concept lattice \(L(K_{i+1})\) by the lattice \(L(K)\) is well understood [4–6, 9, 18, 20, 21]; see [11] for a survey. From practice, it turns out that some incremental algorithms (which, at the \(i^{th}\) step, process \(i\) first attributes or objects) are more efficient in constructing the lattice of the context from scratch than their batch counterparts.

The algorithm of Ganter builds the concept lattice \(L(K)\) and the implication basis \(B(K)\) for the whole context. We propose a new algorithm that, receiving as input \(L(K), B(K)\), and \(K_{i+1}\), builds \(L(K_{i+1})\) and \(B(K_{i+1})\).

The main difficulty the algorithm of Ganter 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, 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 them differently.

We call a set \(A \subseteq M_i \ i\text{-modified if } i \in (A \setminus \{i\})^0\) and \(i\text{-stable otherwise (cf. modified and old concepts in [9]). Thus, } A \text{ is } i\text{-modified if and only if the implication } A \setminus \{i\} \Rightarrow i \text{ holds in } K.\) An implication is \(i\text{-modifed if its premise is } i\text{-modified and } i\text{-stable otherwise.}

Separate processing of modified and stable implications is the key principle behind our algorithm.

**Proposition 1** Subsets of \(i\text{-stable sets are } i\text{-stable for any } i \in M.\)

**Proposition 2** Supersets of \(i\text{-modified sets are } i\text{-modified for any } i \in M.\)

**Proposition 3** For \(A \subseteq M_i\), \(A^{i+1} = A^{i+1}_i\) if and only if \(A\) is \((i + 1)\text{-stable}. A^{i+1}_i = A^{i} \cup \{i + 1\}\) if and only if \(A\) is \((i + 1)\text{-modified}.

**Corollary.** For \(A \subseteq M_i\), \(A^{i+1}_i = A^{i+1}_i \setminus \{i + 1\}\) for any \(A \subseteq M_i.\)

**Proposition 4** The set \(A \subseteq M_i\) is \((i + 1)\text{-modified if and only if } A^{i+1}_i\) is \((i + 1)\text{-modified.}

**Corollary.** The set \(A \subseteq M_i\) is \((i + 1)\text{-stable if and only if } A^{i+1}_i\) is \((i + 1)\text{-stable.}

Thus, \((i + 1)\text{-stable subsets of } M_i\) form an order ideal of the lattice \(2^M, \subseteq\) bounded above by \(i\text{-intents. All other elements of this lattice are } (i + 1)\text{-modified sets. Note that, according to Proposition 4, all elements of the same } i\text{-closure class are either } (i + 1)\text{-modified or } (i + 1)\text{-stable.}\)

**Proposition 5** For \(A, B \subseteq M_i\), the implication \(A \Rightarrow B\) holds in the context \(K_i\) if and only if it holds in \(K_{i+1}\).

**Proposition 6** For \(A, B \subseteq M_i\), the implication \(A \Rightarrow B \cup \{i + 1\}\) holds in the context \(K_{i+1}\) if and only if \(A \Rightarrow B\) holds in the context \(K_i\) and \(A\) is an \((i + 1)\text{-modified set.}\)

Propositions 5 and 6 cover all implications of the context \(K_{i+1}\) with premises that are subsets of \(M_i\). Let us find out which of these premises are pseudo-closed.
Proposition 7 A set \( A \subseteq M_{i+1} \) is \((i + 1)\)-quasi-closed only if \( A \setminus \{i + 1\} \) is i-quasi-closed.

Proof: Take an \((i + 1)\)-quasi-closed set \( A \subseteq M_{i+1} \); show that \( A \setminus \{i + 1\} \) is i-quasi-closed. For every \( B \subseteq M_i \), \( B^i \subseteq B^{i+1}_i \). Consider an arbitrary \( B \subseteq A \setminus \{i + 1\} \). Either \( B^{i+1}_i \subseteq A \) or \( B^{i+1}_i \not\subset A \). In the former case, \( B^i \subseteq A \), and \( B^{i+1}_i \subseteq A \setminus \{i + 1\} \), as \( i + 1 \notin B^i \). In the latter case, \( (A \setminus \{i + 1\})^i \supseteq B^{i+1}_i \subseteq A \setminus \{i + 1\} \). Therefore, \( B^i = (A \setminus \{i + 1\})^i \).

Lemma L1 An \((i + 1)\)-stable set \( A \subseteq M_i \) is \((i + 1)\)-pseudo-closed if and only if \( A \) is i-pseudo-closed.

Proof: As \( A \) is \((i + 1)\)-stable, every its subset is also \((i + 1)\)-stable. Consequently, \( \forall B \subseteq A : B^{i+1}(i+1) = B^i \), and, by definition, \( A \) is i-pseudo-closed if and only if it is \((i + 1)\)-pseudo-closed.

This means that all \((i + 1)\)-stable \((i + 1)\)-pseudo-closed subsets of \( M_i \) are premises of implications from \( B(K) \), which suggests that \((i + 1)\)-modified implications should be separated from \((i + 1)\)-stable ones in \( B(K) \). As we will see later, this idea is quite fruitful.

We know all \((i + 1)\)-stable implications of \( B(K_{i+1}) \) with the premise from \( M_i \) and now turn to \((i + 1)\)-modified ones.

Lemma L2 An \((i + 1)\)-modified set \( A \subseteq M_i \) is an \((i + 1)\)-pseudo-intent if and only if \( A \) is minimal among \((i + 1)\)-modified i-pseudo-intents and i-intents.

Proof: Note that being minimal among \((i + 1)\)-modified i-pseudo-intents and i-intents is equivalent to being minimal among \((i + 1)\)-modified i-quasi-closed sets.

Let \( A \) be minimal among \((i + 1)\)-modified i-quasi-closed sets. If \( A \) is not an \((i + 1)\)-pseudo-intent, then there is an \((i + 1)\)-pseudo-intent \( B \subseteq A \) such that \( B^{i+1}(i+1) \not\subset A \). By Proposition 7, \( B \) is \((i + 1)\)-quasi-closed. Since \( A \) is minimal, \( B \) is \((i + 1)\)-stable. Then, by Proposition \( 3 \), \( B^i = B^{i+1}(i+1) \not\subset A \). From \( A \) being \( i \)-quasi-closed, it follows that \( B^i = A^i \). Hence, \( B^i \) is \((i + 1)\)-modified, which is impossible due to the fact that \( B \) is \((i + 1)\)-stable and Proposition 4.

Now, if \((i + 1)\)-modified \( A \subseteq M_i \) is an \((i + 1)\)-pseudo-intent, then \( A \) is \((i + 1)\)-quasi-closed by Proposition 7. Let \( B \subseteq A \) be minimal among \((i + 1)\)-modified i-quasi-closed sets. As shown above, \( B \) is an \((i + 1)\)-pseudo-intent. However, \( (i + 1) \in B^{i+1}(i+1) \not\subset A \). Hence, \( B = A \), which completes the proof.

Let us consider \((i + 1)\)-pseudo-closed sets from \( M_{i+1} \) that contain \( i + 1 \).

Proposition 8 If \((i + 1)\)-stable \( A \subseteq M_{i+1} \) is \((i + 1)\)-pseudo-closed and \( i + 1 \in A \), then \( A \setminus \{i + 1\} = (A \setminus \{i + 1\})^i \).

Proof: From \( A \) being \((i + 1)\)-stable, it follows that \( A \setminus \{i + 1\} \) is \((i + 1)\)-stable and \( (A \setminus \{i + 1\})^{(i+1)(i+1)} = (A \setminus \{i + 1\})^i \). Taking into account that \( A \) is \((i + 1)\)-quasi-closed, \( A \setminus \{i + 1\} \subseteq (A \setminus \{i + 1\})^i \subseteq A \). Consequently, \( A \setminus \{i + 1\} = (A \setminus \{i + 1\})^i \), which proves the proposition.

Lemma L3 An \((i + 1)\)-stable set \( A \subseteq M_{i+1} \) with \( i + 1 \in A \) is \((i + 1)\)-pseudo-closed if and only if

1. \( A \neq A^{(i+1)(i+1)} \);
2. \( A \setminus \{i + 1\} = (A \setminus \{i + 1\})^i \);
3. for each \((i + 1)\)-stable \((i + 1)\)-pseudo-intent \( B \subseteq A \); if \( i + 1 \in B \), then \( B^{(i+1)(i+1)} \subseteq A \).

Proof: If \( A \) is \((i + 1)\)-pseudo-closed, then (1) and (3) are satisfied by definition and (2) holds by Proposition 8. On the other hand, every subset of \( A \) is \((i + 1)\)-stable. If \( B \subseteq A \) and \( i + 1 \notin B \), then \( B^{i+1}(i+1) = B^i \subseteq (A \setminus \{i + 1\})^i \subseteq A \setminus \{i + 1\} \subseteq A \). Hence, (3) ensures that \( A \) contains the closure of every its \((i + 1)\)-pseudo-closed subset, which together with (1) means that \( A \) is \((i + 1)\)-pseudo-closed.

Lemma L3 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 that Lemma L3 makes it possible to determine if the candidate for pseudo-intent is saturated using only \((i + 1)\)-stable implications. Thus, separate storage of stable and modified implications will make saturation more efficient.

It remains to consider \((i + 1)\)-modified sets \( A \subseteq M_{i+1} \) with \( i + 1 \in A \). Any \((i + 1)\)-modified \((i + 1)\)-pseudo-intent can be obtained from an \((i + 1)\)-modified \((i + 1)\)-pseudo-intent in this or that way: apparently, all the information rendered by an \((i + 1)\)-modified implication \( A \to B \) must already be present in \( B(K) \) except for the information that \( A \to i + 1 \).

Assume that \( \ast \) is the saturation operator in the context \( K_{i+1} \).
Lemma L4 An \((i + 1)\)-modified set \(A \subseteq M_{i+1}\) with \(i + 1 \in A\) is \((i + 1)\)-pseudo-closed if and only if \(A^{(i+1)(i+1)} \neq A = B^*\) for some \((i + 1)\)-modified i-pseudo-closed \(B \subseteq A \setminus \{i + 1\}\).

Proof: The sufficiency is immediate from the definition of the saturation operator. Let us prove the necessity. Suppose that an \((i + 1)\)-modified \(A \subseteq M_{i+1}\) with \(i + 1 \in A\) is \((i + 1)\)-pseudo-closed. Let also \(B \subseteq A \setminus \{i + 1\}\) be an \(i\)-pseudo-intent such that \(B^{(i+1)(i+1)} = A^{(i+1)(i+1)}\). Such \(B\) exists, since \((A \setminus \{i + 1\})^{(i+1)(i+1)} = A^{(i+1)(i+1)}\), while \(A \setminus \{i + 1\}\) is \(i\)-quasi-closed by Proposition 7 and cannot be \(i\)-closed, as \((i + 1)\)-modified \(A\) is not \((i + 1)\)-closed. As \(B \subseteq A\) and \(A\) is an \((i + 1)\)-pseudo-intent, there are no \((i + 1)\)-pseudo-intents or \((i + 1)\)-intents, i.e., sets closed under \(^*\), between \(B\) and \(A\). Therefore, \(B^* = A\).

Lemma L4 promises to find all \((i + 1)\)-modified \((i + 1)\)-pseudo-intents \(A \subseteq M_{i+1}\) with \(i + 1 \in A\) by saturation of \((i + 1)\)-modified \(i\)-pseudo-intents.

4 Incremental Construction of the Canonical Basis

In this section, we translate the preceding results into an algorithm constructing the canonical implication basis. In the pseudo-code below, brackets \([\ ]\) denote lists. Concepts are pairs \((\text{Extent}, \text{Intent})\), while implications are represented by triples \((\text{Extent}, \text{Premise}, \text{Consequence})\), 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., \text{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.

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

Incremental Algorithm

\textbf{Input:} a context \(K = (G, M, I)\).

\textbf{Output:} the canonical implication basis of \(K\).

Begin

\begin{itemize}
  \item \textbf{Begin}
  \item \text{Elements} := [(\text{G, } \emptyset)];
  \item \text{N} := \emptyset;
  \item \textbf{for each} \(j\) \textbf{in} \(M\)
  \item \text{N} := \text{N} \cup \{j\};
  \item \text{AddAttribute}(j, \text{Elements}, \text{N});
  \item \textbf{return} the set of all implications in \text{Elements};
  \item \textbf{End}
\end{itemize}

End

The algorithm goes through the \text{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:

\begin{equation}
\begin{array}{|c|c|c|}
\hline
A \rightarrow B & i + 1 \notin A & i + 1 \in A \\
\hline
(i + 1)\text{-stable} & \text{OldStableImpl (Lemma L1)} & \text{NewStableImpl (Lemma L3)} \\
(i + 1)\text{-modified} & \text{MinModImpl (Lemma L2)} & \text{NonMinModElem (Lemma L4)} \\
\hline
\end{array}
\end{equation}

Lemmas L1–L4 provide a way to obtain all intents and pseudo-intents of the context \(K_{i+1}\) from intents and pseudo-intents of the context \(K_i\). Thus, we have to deal with four types of “generators” of \((i + 1)\)-pseudo-intents: \((i + 1)\)-stable and \((i + 1)\)-modified \(i\)-intents and \(i\)-pseudo-intents. The following table shows which lemmas should be consulted when processing different types of generators:

\begin{equation}
\begin{array}{|c|c|}
\hline
& \text{i\text{-pseudo-intent}} & \text{i\text{-intent}} \\
\hline
(i + 1)\text{-stable} & \text{Lemma L1} & \text{Lemma L3} \\
(i + 1)\text{-modified} & \text{Lemmas L2 and L4} & \text{Lemma L2} \\
\hline
\end{array}
\end{equation}

Most \((i + 1)\)-pseudo-intents are generated by immediate application of Lemmas L1–L3 to \(i\)-intents and \(i\)-pseudo-intents. However, Lemma L4 suggests that some \((i + 1)\)-modified \(i\)-pseudo-intents rejected by Lemma L2 can still prove useful in obtaining \((i + 1)\)-pseudo-intents that contain the new attribute. These \(i\)-pseudo-intents (and \(i\)-intents), which are not minimal in the sense of Lemma L2, are placed into the \text{NonMinModElem}
In the end of the incremental step, when we have a cover of the new basis $B(K_i)$ available, the $i$-pseudo-intents are saturated by the Fuse procedure (see below) and either accepted as $(i+1)$-pseudo-intents or ultimately discarded.

**AddAttribute** ($j$, **Elements**, $N$)

**Input**: a new attribute $j$ and its extent $j'$,

- a list **Elements** with all concepts of $L(K_i)$ and implications of $B(K_i)$,
- the set $N$ of all attributes already processed ($N = M_i$)

**Output**: **Elements** contains all concepts of $L(K_i)$ and implications of $B(K_i)$

**Begin**

OldStableImpl := $\emptyset$;
NewStableImpl := $\emptyset$;
MinModImpl := $\emptyset$;
NonMinModElem := $\emptyset$

for each element in **Elements**

if element.Extent $\subseteq j'$

then

if element is a concept

then ProcessModifiedConcept($j$, element, **Elements**, MinModImpl, NonMinModElem);

else ProcessModifiedImplication($j$, element, **Elements**, MinModImpl, NonMinModElem);

else

if element is a concept

then ProcessStableConcept($j$, element, $N$, NewStableImpl);

else ProcessStableImplication(element, OldStableImpl);

Fuse (OldStableImpl $\cup$ NewStableImpl $\cup$ MinModImpl, **Elements**, NonMinModElem);

**End**

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 and saturation that occur in relation to Lemmas L2 and L3: e.g., to compute the saturation of a set $A$, 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 L1, $(i+1)$-stable $i$-pseudo-intents are $(i+1)$-pseudo-intents; thus, no special processing is required.

**ProcessStableImplication** ($implication$, **OldStableImpl**)

**Begin**

Add implication to **OldStableImpl**;

**End**

2. An $(i+1)$-stable $i$-intent can serve a basis for an $(i+1)$-pseudo-intent $A$ with $i+1 \in A$ if it satisfies the conditions of Lemma L3, which are not difficult to check.

**ProcessStableConcept** ($j$, **concept**, **CurrentAttributes**, **NewStableImpl**)

**Begin**

NewExt := concept.Extent $\cap j'$;
NewPrem := concept.Intent $\cup \{j\}$;
NewCons := $\{n \mid n \in \text{CurrentAttributes} \& \text{NewExt} \subseteq n'\}$;

if NewCons = NewPrem

then

new_concept := (NewExt, NewPrem);
Add new_concept to **Elements**;

else

if $\text{NewPrem} = \text{Saturate(NewPrem, NewStableImpl)}$

new_impl := (NewExt, NewPrem, NewCons);
Add new_impl to **NewStableImpl** and to **Elements**;

**End**
The \textit{CurrentAttributes} set contains all attributes processed (being processed) at the point of the procedure call, i.e., \( j \) and all the preceding attributes. The extent of the attribute \( j \) is denoted by \( j' \), as, for any \( k \) and \( l \geq j, j' = j \). Thus, the \textit{ProcessStableConcept} procedure creates a new attribute set \textit{NewPrem} adding the attribute \( j \) to a \( j \)-stable intent. If \textit{NewPrem} is \( j \)-closed, a new concept is added to \textit{Elements}. Otherwise, \textit{NewPrem} is tested for being pseudo-closed. According to Lemma L3, it suffices to verify that \textit{NewPrem} is saturated by the implications from \textit{NewStableImpl}. If so, the implication \textit{NewPrem} \( \rightarrow \textit{NewCons} \) is added to \textit{NewStableImpl} and to \textit{Elements}. There is no need in actual computation of the saturation of \textit{NewPrem}: it is enough to check whether \textit{NewStableImpl} contains an implication whose premise is a subset of \textit{NewPrem} and whose consequence is not. This check can be performed in a number of steps linear in the number of implications in \textit{NewStableImpl}, which is much more efficient than the computation of the saturation. However, \textit{NewStableImpl} should already contain all relevant implications of the \( j \)-basis whose premises are subsets of \textit{NewPrem}. That is why the order of processing is important.

3. \((i + 1)\)-modified \( i \)-pseudo-intents demand special attention, as they give rise to pseudo-intents of two types described by Lemmas L2 and L4.

\begin{verbatim}
ProcessModifiedImplication (j, implication, Elements, MinModImpl, NonMinModElem)
Begin
  Add j to implication.Consequence;
  if \( \neg\exists \text{min} \in \text{MinModImpl} (\text{min}.\text{Premise} \subseteq \text{implication}.\text{Premise}) \)
     then Add implication to MinModImpl;
     else
       Add j to implication.Premise;
       Move implication from Elements to NonMinModElem;
End

If an implication is \( j \)-modified, \( j \) should be added to its consequence. First, we check the conditions of Lemma L2. They are satisfied if and only if \textit{MinModImpl} does not contain an implication whose premise is a subset of \textit{implication}.Premise (assuming that \( i \)-pseudo-intents and \( i \)-intents are processed from smaller to larger ones.) If the check is successful, then \textit{implication}.Premise is a minimal \( j \)-modified \( i \)-pseudo-intent, and \textit{implication} should be added to \textit{MinModImpl}. If not, it is necessary to compute the saturation of \textit{implication}.Premise, which can be \( j \)-pseudo-closed according to Lemma L4. However, we will be in a position to do it only when the set of implications equivalent to \( B(K_{i+1}) \) becomes available; therefore, we put \textit{implication} into \textit{NonMinModElem} and leave it for a while.

4. \((i + 1)\)-modified \( i \)-intents are responsible only for implications described by Lemma L2.

\begin{verbatim}
ProcessModifiedConcept (j, concept, Elements, MinModImpl, NonMinModElem)
Begin
  if \( \neg\exists \text{min} \in \text{MinModImpl} (\text{min}.\text{Premise} \subseteq \text{concept}.\text{Intent}) \)
     then new_impl := (concept.Extent, concept.Intent, concept.Intent \( \cup \) \( \{ j \} \));
       Add new_impl to MinModImpl;
       Replace concept with new_impl in Elements;
     else
       Remove concept from Elements;
       Add j to concept.Intent;
       Add concept to NonMinModElem;
End

Similar to the preceding case, we check the minimality. If this test is successful, \textit{concept}.Intent is a minimal \( j \)-modified \( i \)-pseudo-intent. The new implication with the premise \textit{concept}.Intent is substituted for \textit{concept} in the \textit{Elements} list and is added to \textit{MinModImpl}. As for \textit{concept}, it is placed to the end of \textit{Elements} whether or not the conditions of Lemma L2 are satisfied. This is necessary to maintain the desired order.

Thus, we have four sets: \textit{OldStableImpl}, \textit{NewStableImpl}, \textit{MinModImpl}, and \textit{NonMinModElem}. The first three sets contain all implications of the basis (and only them) except for those described by Lemma L4. At the last step of the algorithm, implication premises from \textit{NonMinModElem} are saturated and the implications together with concepts from this set are added to \textit{Elements}.

\begin{verbatim}
Fuse(Basis, Elements, ExtraElements)
Begin
  ExtraImpl is the list of all implications from ExtraElements;
  for each element in ExtraElements
End
\end{verbatim}
if element is a concept then add element to Elements;  
else
  Remove element from ExtraImpl;
  OtherImpl := Basis ∪ ExtraImpl;
  impl.Premise := Saturate (impl.Premise, OtherImpl);
  if impl.Premise ⊆ impl.Consequence then add impl to Basis and to Elements;
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 [13]; the one proposed in [22], which has nonlinear time complexity but is claimed to be an enhanced version of LinClosure; etc.

Saturate(New_Prem, Impl)
Begin
  New_Closure := New_Prem;
  Unused_Impl := Impl;
  do
    Old_Closure := New_Closure;
    for each (Ext, Prem, Cons) in Unused_Impl
      if (Prem ⊆ New_Closure) then
        New_Closure := New_Closure ∪ Cons;
        Unused_Impl := Unused_Impl \ {(Ext, Prem, Cons)};
      end do
    while Old_Closure ≠ New_Closure;
  return (New_Closure);
End

The correctness of the proposed algorithm can be considered well founded if we are able to show that, at each point of execution of the algorithm, concepts and implications in the Elements list are arranged according to the order ⊆ of premises and intents starting from the smallest ones.

Before the Fuse procedure is called, the Elements list is extended only by new pseudo-intents and intents obtained by processing i-intents. If A is an i-intent, then a j-pseudo-intent or j-intent A ∪ {j} may be added to the end of the list. This does not violate the order. Indeed, for any A and B, j ∉ A, j ∉ B: B ⊆ A ⇔ B ∪ {j} ⊆ A ∪ {j}.

At the moment when the Fuse procedure is called, the Elements list contains all j-stable j-pseudo-intents and j-intents, as well as j-modified pseudo-intents A such that j ∉ A. The Fuse procedure adds to Elements j-modified j-intents and j-pseudo-intents A such that j ∈ A. Obviously, such A cannot be a subset of a pseudo-intent or an intent from the Elements list. Therefore, it suffices to show that the elements of the NonMinModElem list are in the required order. And this is so, indeed: NonMinModElem was formed by elements of the form A ∪ {j} with A from the properly ordered Elements list. It can be easily seen that the saturation will not destroy the order: recall that a pseudo-intent cannot be a subset of a pseudo-intent within the same closure class (i.e., with the same closure).

Let us illustrate the work of the algorithm with the context from Example 1. A table row corresponds to the pair “new attribute”—“old concept”; it has an identifier #. Such identifiers occur beside some elements in the column Elements; it means that the given element was removed from the list at the step denoted by the identifier. A table cell is split into several ones if this element was replaced in the list by the next element.

<table>
<thead>
<tr>
<th>#</th>
<th>element</th>
<th>OldStableImpl</th>
<th>NewStableImpl</th>
<th>MinModImpl</th>
<th>NonMinModElem</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(1234, ∅)</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>(1234, ∅)</td>
<td></td>
<td></td>
<td></td>
<td>(12, a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(12, a, ac)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(12, a, ac)</td>
</tr>
<tr>
<td>b1</td>
<td>b</td>
<td>(1234, ∅)</td>
<td></td>
<td></td>
<td></td>
<td>(34, b)</td>
</tr>
<tr>
<td>b2</td>
<td></td>
<td>(12, a)</td>
<td></td>
<td></td>
<td></td>
<td>(34, b)</td>
</tr>
<tr>
<td>c1</td>
<td>c</td>
<td>(1234, ∅)</td>
<td></td>
<td></td>
<td></td>
<td>(123, c)</td>
</tr>
<tr>
<td>c2</td>
<td></td>
<td>(12, a)</td>
<td></td>
<td></td>
<td></td>
<td>(12, a, ac)</td>
</tr>
<tr>
<td>c3</td>
<td>(34, b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(3, bc)</td>
</tr>
<tr>
<td>Context</td>
<td>Number of concepts</td>
<td>Size of the basis</td>
<td>Algorithm of Ganter</td>
<td>Incremental algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>--------------------</td>
<td>-----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>((M, M, #),</td>
<td>M</td>
<td>= 18)</td>
<td>262144</td>
<td>0</td>
<td>1.9 sec; 0</td>
<td>2.4 sec; 0</td>
</tr>
<tr>
<td>(</td>
<td>G</td>
<td>= 10,</td>
<td>M</td>
<td>= 100,</td>
<td>g©</td>
<td>= 25)</td>
</tr>
<tr>
<td>(</td>
<td>G</td>
<td>= 20,</td>
<td>M</td>
<td>= 100,</td>
<td>g©</td>
<td>= 25)</td>
</tr>
<tr>
<td>(</td>
<td>G</td>
<td>= 50,</td>
<td>M</td>
<td>= 100,</td>
<td>g©</td>
<td>= 10)</td>
</tr>
<tr>
<td>SPECT: (</td>
<td>G</td>
<td>= 267,</td>
<td>M</td>
<td>= 23), see [3]</td>
<td>21550</td>
<td>2169</td>
</tr>
</tbody>
</table>

5 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, NextClosure. 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 proposed a new approach that specifically addresses the introduction of 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 suppose storing all intents in return). An algorithm updating the implication basis with the addition of a new object is a subject of the further research.

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 a pity to carry over the latter when you just want to care the former. Since our algorithm generates the concept set in addition to the canonical basis, it has necessarily exponential time complexity in terms of the input context size. However, there is no evidence that the size of the implication basis must be estimated as exponential; therefore, a challenge would be to devise an algorithm with a more attractive theoretical complexity.
Acknowledgements

The authors would like to thank Centre National de la Recherche Scientifique (CNRS) and Russian State University for the Humanities (RSUH) for providing the invitations and travel facilities for the collaboration. The first author was also supported by the International Quality Network (IQN), DAAD.

References


