

## TWO BASIC ALGORITHMS IN CONCEPT ANALYSIS\*

Bernhard Ganter

Preprint-Nr. 831

Juni 1984

Abstract: We describe two algorithms for closure systems. The purpose of the first is to produce all closed sets of a given closure operator. The second constructs a minimal family of implications for the "logic" of a closure system. These algorithms then are applied to problems in concept analysis: Determining all concepts of a given context and describing the dependencies between attributes. The problem of finding all concepts is equivalent, e.g., to finding all maximal complete bipartite subgraphs of a bipartite graph.

---

\* A version of this preprint was published in German as part of the book "Beiträge zur Begriffsanalyse" (Ganter, Wille and Wolff, eds., BI-Wissenschaftsverlag 1987). Computer programs for FCA were already in use in 1984. The new algorithm was not only more efficient, it could also be used to compute the "canonical base" of implications in a finite formal context that had recently been discovered by J.L. Guigues and V. Duquenne.

§1 CLOSURE SYSTEMS

Let  $2^M$  denote the power set of a set  $M$ . By a CLOSURE OPERATOR we mean a mapping

$$\bar{\phantom{x}} : 2^M \rightarrow 2^M$$

which is extensive, monotone, and idempotent, i.e. which satisfies for all  $A, B \subseteq M$

- a)  $A \subseteq \bar{A}$
- b)  $A \subseteq B \Rightarrow \bar{A} \subseteq \bar{B}$
- c)  $\bar{\bar{A}} = \bar{A}$ .

A set  $X$  is CLOSED iff  $X = \bar{X}$ . The collection of all closed sets of some closure operator is called a CLOSURE SYSTEM. It is easy to see that a family  $\mathcal{F} \subseteq 2^M$  is a closure system if and only if  $M \in \mathcal{F}$  and  $\mathcal{F}$  is closed under arbitrary intersections. The corresponding closure operator then is given by

$$A \mapsto \bar{A} := \bigcap \{C \in \mathcal{F} \mid A \subseteq C\}.$$

There may be as many as  $2^{|M|}$  closed sets. Thus, any algorithm computing all closed sets will require a computing time exponential in  $|M|$ , even if the time for computing  $\bar{X}$  from  $X$  is neglected. Nevertheless, some algorithms are slower than others, and the slow ones seem to be more popular.

In this paragraph we describe a simple algorithm which requires per closed set

- at most  $|M|$  steps, each of which consists of
- a single application of the closure operator, plus
- some elementary set manipulations (of complexity  $O(|M|)$ ),

Moreover, the total number of steps never exceeds  $2^{|M|}$ .

We shall adapt a method which has proven successful for many similar problems. Think of the subsets of  $M$  as being lexicographically ordered, and suppose that we have an algorithm that produces from a given closed set the lexicographically next closed set. We can scan through all closed sets by repeated applications of this algorithm: We start with the lexicographically first set (which is the closure of the empty set) and apply the algorithm again and again until we reach the lexicographically last set (which is  $M$ ).

There is, however, some ambiguity to the term "lexicographic order", when applied to sets. To be more precise, we describe formally a linear ordering of the subsets of  $M$ , to which we shall refer as the LECTIC ORDER. We assume that  $M$  itself is linearly ordered, w.l.o.g.  $M = \{1, 2, \dots, m\}$ .

1.1. DEFINITION: For  $A, B \subseteq \{1, 2, \dots, m\}$ , define

$$A < B : \Leftrightarrow \exists i \in B \setminus A \quad A \cap \{1, 2, \dots, i-1\} = B \cap \{1, 2, \dots, i-1\}.$$

We read  $A < B$  as "A is lectically smaller than B".

Verbally,  $A \leq B$  iff the smallest element in which  $A$  and  $B$  differ belongs to  $B$ . It is quite obvious that this defines a linear order. Note that  $A \subseteq B$  implies  $A \leq B$ . We should mention that the lectic order, as it is defined above, is the same as the lexicographic order of the characteristic vectors. (As usual, the characteristic vector of a set  $A \subseteq M$  is the binary tuple  $a_1 a_2 \dots a_m$ , where  $a_i = 1$  iff  $i \in A$ .) It is immediate from 1.1 that  $A < B \Leftrightarrow$  the characteristic vector of  $A$  is lexicographically smaller than the characteristic vector of  $B$ .

Our next aim is to characterize the lectic successor of a given set  $A \subseteq M$ ,  $M$  finite. We prepare by introducing two notations and by listing, without proof, some easy facts:

1.2 DEFINITIONS: For  $A, B \subseteq M$ ,  $i \in M$ , define

- 1)  $A <_i B \Leftrightarrow i \in B \setminus A$  and  $A \cap \{1, 2, \dots, i-1\} = B \cap \{1, 2, \dots, i-1\}$
- 2)  $A @ i := \overline{A \cap \{1, 2, \dots, i-1\} \cup \{i\}}$

1.3 PROPOSITION:

- 1)  $A < B$  iff  $A <_i B$  for some  $i \in \{1, 2, \dots, m\}$ .
- 2) If  $A <_i B$ ,  $A <_j C$ , and  $i < j$ , then  $C <_i B$ .
- 3)  $i \notin A \Rightarrow A < A @ i$ .
- 4) If  $A <_i B$  for some closed set  $B$ , then  $A @ i \subseteq B$ , thus  $A @ i \leq B$ .
- 5) If  $A <_i B$  for some closed set  $B$ , then  $A <_i A @ i$ .

1.4 LEMMA:

With respect to the lectic order, the smallest closed set greater than a given set  $A \subseteq M$  is

$$A @ i,$$

where  $i$  is the largest element of  $M$  satisfying

$$A <_i A @ i$$

Proof: Let  $A^+$  be the lectic successor of  $A$ . We have to show that  $A^+$  is of the form described in 1.4.

Since  $A < A^+$  we have  $A <_i A^+$  for some  $i \in M$ . By 1.3.5 we conclude that  $A <_i A @ i$ , and from 1.3.4 we get that  $A @ i \leq A^+$ , thus  $A^+ = A @ i$ . It remains to prove that this  $i$  is as large as possible. Suppose  $A <_j A @ j$  for some  $j \neq i$ . Since  $A @ i = A^+ < A @ j$  we can use 1.3.2 to deduce  $j < i$ .

It is not difficult to translate 1.4 into an algorithm:

1.5 ALGORITHM to compute the successor  $A_{\text{next}}$  of a given set  $A$  with respect to the lexic order.

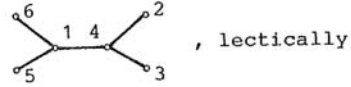
- (A) For  $i = m, m-1, \dots, 2, 1$  step -1
- (B) If  $A <_i A @ i$  then return with  $A_{\text{next}} \leftarrow A @ i$
- (C) Next  $i$
- (D) Stop: No successor (input was  $A = M$ ) .

A more detailed version of step (B) is:

- (B1) If  $i \notin A$  then goto (C)
- (B2)  $A_{\text{next}} \leftarrow \overline{A \cap \{1, 2, \dots, i-1\} \cup \{i\}}$
- (B3) If there is an element  $j \in A_{\text{next}}$  such that  $j < i$  and  $j \notin A$  then goto (C)
- (B4) Return

1.6 EXAMPLE:

The connected subgraphs of the tree  
ordered:



Vertex set	characteristic vector
$\{\emptyset\}$	0 0 0 0 0 0
$\{6\}$	0 0 0 0 0 1
$\{5\}$	0 0 0 0 1 0
$\{4\}$	0 0 0 1 0 0
$\{3\}$	0 0 1 0 0 0
$\{3, 4\}$	0 0 1 1 0 0
$\{2\}$	0 1 0 0 0 0
$\{2, 4\}$	0 1 0 1 0 0
$\{2, 3, 4\}$	0 1 1 1 0 0
$\{1\}$	1 0 0 0 0 0
$\{1, 6\}$	1 0 0 0 0 1
$\{1, 5\}$	1 0 0 0 1 0
$\{1, 5, 6\}$	1 0 0 0 1 1
$\{1, 4\}$	1 0 0 1 0 0
$\{1, 4, 6\}$	1 0 0 1 0 1
$\{1, 4, 5\}$	1 0 0 1 1 0
$\{1, 4, 5, 6\}$	1 0 0 1 1 1
$\{1, 3, 4\}$	1 0 1 1 0 0
$\{1, 3, 4, 6\}$	1 0 1 1 0 1
$\{1, 3, 4, 5\}$	1 0 1 1 1 0
$\{1, 3, 4, 5, 6\}$	1 0 1 1 1 1
$\{1, 2, 4\}$	1 1 0 1 0 0
$\{1, 2, 4, 6\}$	1 1 0 1 0 1
$\{1, 2, 4, 5\}$	1 1 0 1 1 0
$\{1, 2, 4, 5, 6\}$	1 1 0 1 1 1
$\{1, 2, 3, 4\}$	1 1 1 1 0 0
$\{1, 2, 3, 4, 6\}$	1 1 1 1 0 1
$\{1, 2, 3, 4, 5\}$	1 1 1 1 1 0
$\{1, 2, 3, 4, 5, 6\}$	1 1 1 1 1 1

The rest of this paragraph is devoted to improving this algorithm.

The reader who is not interested in actually programming this procedure should skip the following pages (and continue with §2!)

The notation of the closed sets by means of their characteristic vectors, as in the above example, makes a useful property of the lexic order quite conspicuous: Those characteristic vectors which start with some specified tuple come one after another. The algorithm therefore can easily be modified to generated only such closed sets which contain certain specified elements and avoid certain others.

A closer inspection of 1.5 shows that the closure operator itself was used only in a special instance, namely to compute  $A @ i$ . In some cases this can simplify the computation, in particular if the closure of  $A$  is already known and can be used. Taking this into consideration we formulate a refined version of the algorithm.

### 1.7 ALGORITHM (modified version of 1.5) :

#### Input/Output:

A number  $r \geq 0$  ,  
 a list  $i_0 < i_1 < \dots < i_r$  of elements of  $M$ ,  
 a list  $A_0 \subset A_1 \subset \dots \subset A_r (=A)$  of closed sets,  
 satisfying the following conditions:

$$i_0 = 0$$

$$A_0 = \emptyset$$

$$A_j = A_{j-1} @ i_j \quad \text{for } j = 1, \dots, r ,$$

$$A_{j-1} < i_j \subset A_j \quad \text{for } j = 1, \dots, r .$$

#### Algorithm:

- (A) For  $i = m, m-1, \dots, 1$  step -1
- (B) If  $i \in A_r$  then goto (G)
- (C) If  $i < i_r$  then  $r \leftarrow r-1$ ; goto (C)
- (D)  $i_{r+1} \leftarrow i$  ;  $A_{r+1} \leftarrow A_r \cup \{i\}$
- (E) If  $A_{r+1} \setminus A_r$  contains an element  $< i$  then goto (G)
- (F)  $r \leftarrow r+1$  ; return
- (G) Next  $i$
- (H) Stop: No successor (input was  $A_r = M$ ).

Initialization (when all closed sets are to be produced): The algorithm is called first with input  $r = 0$ ,  $i_0 = 0$ ,  $A_0 = \emptyset$ . Later, the output of the previous run is used as the new input. If this is done then  $A_r$  runs through all closed sets (Note that  $\emptyset$ , the first input, never occurs as an output.).

No doubt, the rather intricate data list used as input/output-list requires some explanation. Perhaps it is helpful to see an example first:

1.8 EXAMPLE: Again we consider the subtrees of the tree in the figure. We use 1.7 to determine the lectic successor of the subtree  $\{2,4\}$ .

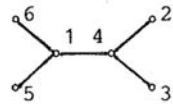


figure 1

If we choose  $r = 2$

$$i_0 = 0, i_1 = 2, i_2 = 4$$

$$A_0 = \emptyset, A_1 = \{2\}, A_2 = \{2,4\}$$

then, since  $A_1 = A_0 \oplus i_1$

$$A_2 = A_1 \oplus i_2$$

$$A_0 <_2 \{2\}$$

$$A_1 <_4 \{2,4\},$$

this list is suitable as an input for 1.7. Here is what the algorithm does:



- (A)  $i \leftarrow 6$   
 (B) —  
 (C) —  
 (D)  $A_3 \leftarrow \overline{\{2,4\} \cup \{6\}} = \{1,2,4,6\}$   
 (E) goto (G)  
 (G)  $i \leftarrow 5$   
 (B) —  
 (C) —  
 (D)  $A_3 \leftarrow \overline{\{2,4\} \cup \{5\}} = \{1,2,4,5\}$   
 (E) goto (G)  
 (G)  $i \leftarrow 4$   
 (B) goto (G)  
 (G)  $i \leftarrow 3$   
 (B) —  
 (C)  $r \leftarrow 1$ ; goto (C)  
 (C) —  
 (D)  $i_2 \leftarrow 3$ ;  $A_2 \leftarrow \overline{\{2\} \cup \{3\}} = \{2,3,4\}$   
 (E) —  
 (F)  $r \leftarrow 2$ ; return

The actual values are now

$$r = 2$$

$$i_0 = 0, i_1 = 2, i_2 = 3$$

$$A_0 = \emptyset, A_1 = \{2\}, A_2 = \{2,3,4\}.$$

The lexic successor of the subtree  $\{2,4\}$  thus is the subtree

$$A_r = \{2,3,4\}. \text{ Because of } A_2 = A_1 \oplus 3$$

$$A_1 <_3 A_2,$$

the output list also satisfies the conditions of 1.7 and can be used for another call of the algorithm.

In fact, understanding the structure of the I/O - list in 1.7 is the crucial step in proving the correctness of the algorithm. Let  $A$  be the closed set whose successor is to be computed, i.e.  $A_r = A$  at the start. The conditions in 1.7 imply that, for each  $j \leq r$

$$A_j = (\dots((\overline{\emptyset} \oplus i_1) \oplus i_2) \oplus \dots) \oplus i_j$$

and therefore

$$A_j = \overline{A \cap \{1, 2, \dots, i_j\}}.$$

Steps (B) and (C) cause that  $i_r < i$ , therefore in Step (D)

$$\overline{A_r \cup \{i\}}$$

is equal to  $A_r \oplus i$ , and steps (E) and (F) can be jointly reformulated to

$$\text{If } A <_i A_{r+1} \text{ then } r \leftarrow r+1 ; \text{return.}$$

Since  $A_{r+1} = A \oplus i$ , this is exactly the same as step (B) of 1.5.

We also find that

$$A_r \oplus i = A \oplus i.$$

This guarantees that the output list is of the appropriate form.

One might wish to start the algorithm with some set different from  $\overline{\emptyset}$ . For this it is necessary to build up an input list as required by 1.7.

**1.9 PROPOSITION:** For  $A \in M$  define numbers  $r, i_0, i_1, \dots, i_r$  and sets  $A_0, A_1, \dots, A_r$  by the following rules:

- 1)  $i_0 := 0, A_0 := \overline{\emptyset}$
- 2) If  $i_j$  and  $A_j$  are already defined then either  $A \setminus A_j = \emptyset$ , in which case we define  $r := j$  or else let
 
$$i_{j+1} := \text{smallest element of } A \setminus A_j$$

$$A_{j+1} := \overline{A_j \cup \{i_{j+1}\}}.$$

Then  $A_r = \overline{A}$ , and the so defined data list satisfies the conditions of 1.7.

Proof: Since  $A_{j+1} = \overline{A_j \cup \{i_{j+1}\}}$  for  $0 \leq j < r$ , and since  $i_{j+1}$  is the smallest element of  $A \setminus A_j$ , we conclude that

$$A_0 \subset A_1 \dots \subset A_r$$

$$i_0 < i_1 < \dots < i_r.$$

Since  $A_j = \overline{\{i_1, i_2, \dots, i_j\}}$ ,  $0 \leq j \leq r$

we conclude that

$$A_{j+1} = \overline{A_j \cap \{1, 2, \dots, i_j\} \cup \{i_{j+1}\}}$$

and

$$A_j <_{i_{j+1}} A_{j+1}.$$

This shows that the conditions of 1.7 are satisfied. Clearly

$\{i_1, i_2, \dots, i_r\} \subseteq A$ , thus  $A_r = \overline{\{i_1, i_2, \dots, i_r\}} \subseteq \bar{A}$ . But  $A \setminus A_r = \emptyset$ , thus  $A_r = \bar{A}$ .

The practical value of 1.9 is that it provides an algorithm to obtain a suitable input list for 1.7. We need this when we want to compute all closed sets which are lexicographically larger than a given one.

A final remark: If the algorithm 1.7 is implemented using a programming language that admits "local" variables then the legendary I/O-list can be reduced to  $A_r$ , plus a logical flag to indicate the beginning and the end of the lexic list.

## §2 PSEUDO-CLOSED SETS

Throughout this chapter, let  $M$  be a finite set and let

$$— : 2^M \rightarrow 2^M$$

be a closure operator. Our aim is to represent this operator by means of implications. To make this precise, define an IMPLICATION

(WITHIN  $M$ ) to be a pair  $A, B$  of subsets of  $M$ , denoted  $A \rightarrow B$ . An implication  $A \rightarrow B$  HOLDS for the given closure operator  $\bar{\phantom{x}}$  if  $B \subseteq \bar{A}$ . A set  $J$  of implication holds if each of these implications holds.

Conversely, let  $J$  be a set of implications within  $M$ . A set  $X \subseteq M$  RESPECTS  $J$  if for every implication  $A \rightarrow B$  in  $J$   $A \subseteq X$  implies  $B \subseteq X$ . A closure operator respects  $J$  if every closed set does. There is a unique finest closure operator respecting a given set of implications, as the following proposition shows:

2.1 PROPOSITION: Let  $J$  be a set of implications within  $M$ .

Then  $\{X \subseteq M \mid X \text{ respects } J\}$

is a closure system.

Proof:  $M$  obviously respects every set of implications. So all we have to show is that

$$\{X \subseteq M \mid X \text{ respects } J\}$$

is closed under intersections. Suppose  $\mathcal{Y}$  is a subfamily, i.e.

$\mathcal{Y} \subseteq \{X \subseteq M \mid X \text{ respects } J\}$ , and let  $S := \bigcap \mathcal{Y}$ . If  $A \rightarrow B$  is some implication from  $J$  with  $A \subseteq S$  then  $A \subseteq X$  for every  $X \in \mathcal{Y}$ , thus  $B \subseteq X$  for every  $X \in \mathcal{Y}$ , thus  $B \subseteq S$ .

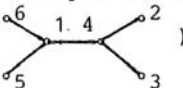
$A \rightarrow B$  is a CONSEQUENCE of a set  $J$  of implications if every set which respects  $J$  also respects  $A \rightarrow B$ . An equivalent condition is that  $A \rightarrow B$  holds for every closure operator for which  $J$  holds. A set of implications is REDUCED if none of these implications is a consequence of the others.

A set  $J$  of implications is COMPLETE (for a closure operator  $\bar{\phantom{x}}$ ) if the implications that hold for  $\bar{\phantom{x}}$  are exactly the consequences of  $J$ .

A QUASI-CLOSED SET is a set  $Q \subseteq M$  such that  $X \subseteq Q$  implies  $\bar{X} = \bar{Q}$  or  $\bar{X} \subseteq Q$  for all  $X$ . Clearly every closed set is quasi-closed.

A PSEUDO-CLOSED SET is a quasi-closed set which is not closed and satisfies a certain minimality condition. More precisely,  $P \subseteq M$  is pseudo-closed if and only if

- i)  $P$  is quasi-closed,
- ii)  $P$  is not closed,
- iii)  $Q \subseteq P$  and  $\bar{Q} = \bar{P}$  for quasi-closed  $Q$  implies  $Q = P$ .

2.2. EXAMPLE: For the closure operator of example 1.6 (where the closed sets are the subtrees of the tree ) there are exactly 10 pseudo-closed sets.

These are the two-element subsets which are not closed, like  $\{1,2\}$ ,  $\{1,3\}$ ,  $\{2,3\}$  etc.

DUQUENNE & GUIGUES have given, in a different setting, a theorem that analyzes the rôle of the pseudo-closed sets:

2.3 THEOREM (cf. DUQUENNE & GUIGUES [2]):

The set of implications

$$\{P \rightarrow \bar{P} \mid P \text{ pseudo-closed}\}$$

is reduced and complete.

Proof: Call this family of implications  $J$ . We have to show that  $J$  is reduced and complete. It is trivially true that  $J$  holds for  $\bar{\phantom{x}}$ , thus to show that  $J$  is complete we must prove that

every set which respects  $J$  is closed under  $\bar{\phantom{x}}$ . Let  $Q$  be a minimal counterexample, i.e. suppose that  $Q$  respects  $J$ , that  $Q \not\subseteq \bar{Q}$ , and that every proper subset of  $Q$  which respects  $J$  is closed. We claim that  $Q$  is quasi-closed. To see this consider some  $X \subseteq Q$ . By 2.1 there is a smallest set  $\tilde{X} \supseteq X$  which respects  $J$ . Since  $Q$  respects  $J$  we have  $\tilde{X} \subseteq Q$ , and since  $\bar{X}$  respects  $J$  we have  $\tilde{X} \subseteq \bar{X}$ . Thus if  $\tilde{X} = Q$  then  $\bar{X} = \bar{Q}$ . On the other hand if  $\tilde{X} \neq Q$  then, by the minimality assumption,  $\tilde{X} = \bar{X}$  and thus  $\tilde{X} \subseteq Q$ . This shows that  $Q$  is quasi-closed.

But this is impossible:  $Q$  is not closed, so there has to be some pseudo-closed set  $P$  with  $P \subseteq Q$  and  $\bar{P} = \bar{Q}$ , and  $P \not\rightarrow \bar{P}$  is in  $J$ . Obviously,  $Q$  does not respect this implication.

To see that  $J$  is reduced, let  $P$  be some pseudo-closed set. We prove that  $P$  respects  $J \setminus \{P \rightarrow \bar{P}\}$ : If  $A \subsetneq P$  is any pseudo-closed set then  $\bar{A} \not\subseteq \bar{P}$ . But since  $P$  is quasi-closed, this implies  $\bar{A} \subseteq P$ .

Is there a good algorithm to construct the pseudo-closed sets?

Actually, we do not know. All we can offer are procedures that construct the closed and the pseudo-closed sets, or even all quasi-closed sets. These algorithms rest on the following proposition:

#### 2.4 PROPOSITION:

- 1) The family of all quasi-closed sets is a closure system.
- 2) The family consisting of all sets which are either closed or pseudo-closed is a closure system.

The proof of 2.4 will be given after proposition 2.5. Note that 2.4 enables us to use the algorithms of §1 to generate all quasi-closed sets or, resp., all sets which are closed or pseudo-closed.

The missing tool is a method for computing the "quasi-closure" or the "pseudo-closure" of arbitrary sets  $A \subseteq M$ . This will also be implicitly given by 2.5.

### 2.5. PROPOSITION:

- 1) A set  $Q \subseteq M$  is quasi-closed if and only if  $Q \cap C$  is closed for every closed set  $C$  with  $Q \not\subseteq C$ .
- 2) A set  $R \subseteq M$  is closed or pseudo-closed if and only if  $\bar{P} \subseteq R$  for every pseudo-closed set  $P \subsetneq R$ .

Proof: 1), " $\Rightarrow$ ". Let  $Q$  be quasi-closed and let  $Q \cap C =: D$ , where  $C$  is some closed set with  $Q \not\subseteq C$ . Then  $\bar{D} \subseteq \bar{C}$ , and this implies  $\bar{D} \not\subseteq \bar{Q}$ , therefore  $\bar{D} \subseteq Q$ . Thus  $\bar{D} = D$ .

1), " $\Leftarrow$ ". Suppose  $Q \cap C$  is always closed when  $C$  is closed and  $Q \not\subseteq C$ . Let  $A \subseteq Q$  with  $\bar{A} \not\subseteq \bar{Q}$ . Then  $\bar{A} \cap Q$  is closed, hence  $\bar{A} \cap Q = \bar{A}$ , i.e.  $Q$  is quasi-closed.

2), " $\Rightarrow$ ". If  $R$  is pseudo-closed then  $R$  is quasi-closed, but not closed. This implies that  $\bar{P} \subseteq R$  for every set  $P \subseteq R$  with  $\bar{P} \not\subseteq \bar{R}$ . But if  $P \subsetneq R$  is pseudo-closed then clearly  $\bar{P} \not\subseteq \bar{R}$ .

2), " $\Leftarrow$ ". Suppose  $R$  is not pseudo-closed and satisfies the condition, i.e.  $\bar{P} \subseteq R$  for every pseudo-closed set  $P \subseteq R$ . Then  $R$  respects

$$\{P \rightarrow \bar{P} \mid P \text{ pseudo-closed}\}$$

and by 2.1, 2.2 therefore is closed.

The proof of 2.4 now reduces to showing that the characteristic properties described in 2.5 are preserved by arbitrary intersections. This is straightforward in both cases, we give the second case as an example: Let  $X$  be some index set and let all elements of

$\{R_x | x \in X\}$  be closed or pseudo-closed. We have to show that the same holds for  $R := \bigcap_{x \in X} R_x$ . Let  $P \subsetneq R$  be some pseudo-closed set. Then  $P \subsetneq R_x$  for all  $x \in X$ , thus  $\bar{P} \subseteq R_x$  for all  $x \in X$ , thus  $\bar{P} \subseteq R$ .

Propositions 2.4 and 2.5 can in several different way be utilized to construct algorithms for computing the pseudo-closed sets. Since we do not have much experience with these algorithms, and since we still hope for a better idea to construct these sets, we only give an example:

Using 2.5.2 we can decide if a set  $X \subseteq M$  with  $X = \bar{X}$  is pseudo-closed if we know all pseudo-closed sets properly contained in  $X$ , and their closures. Since  $P \subseteq X$  implies that  $P$  is lexicographically smaller than  $X$ , the knowledge of all pseudo-closed sets  $X$  suffices to decide if  $X$  is pseudo-closed, and also to construct the lexicographically smallest pseudo-closed set  $\succ X$ . The following algorithm is a recursive application of this construction.

2.6 ALGORITHM to compute all sets which are pseudo-closed with respect to a given closure operator  $\bar{\phantom{x}}$  on the finite set  $M := \{1, 2, \dots, m\}$ .

Variables (for the dimension of the arrays see the remark below):

$n$  is the number of pseudo-closed sets.

$P(1), P(2), \dots, P(n)$  is the list of pseudo-closed sets.

$\bar{P}(1), \bar{P}(2), \dots, \bar{P}(n)$  is the list of the closures of the  $P(i)$ .

$\text{LOG}(1), \text{LOG}(2), \dots, \text{LOG}(n)$  is an auxiliary logical vector to avoid unnecessary work.



Algorithm:

- (A)  $n \leftarrow 0; A \leftarrow \emptyset; \text{LOG}(1) \leftarrow \text{TRUE}$
- (B) If  $\emptyset \neq \emptyset$  then  $n \leftarrow 1; P(1) \leftarrow \emptyset; \bar{P}(1) \leftarrow \emptyset$
- (C) For  $i = m, m-1, \dots, 1$  step -1
- (D) If  $i \in A$  then goto (S)
- (E)  $B \leftarrow A \oplus \{i\}; C \leftarrow \bar{B}; (\text{LOG}(1), \dots, \text{LOG}(n)) \leftarrow (\text{FALSE}, \dots, \text{FALSE})$
- (F)  $\text{FLAG} \leftarrow \text{FALSE}$
- (G) For  $j = 1, 2, \dots, n$
- (H) If  $\text{LOG}(j) = \text{TRUE}$  then goto (O)
- (I) If  $P(j) \not\subseteq C$  then goto (N)
- (J) If  $P(j) \not\subseteq B$  then goto (O)
- (K) If  $\bar{P}(j) \subseteq B$  then goto (N)
- (L) If  $\bar{P}(j) \setminus A$  contains an element  $< i$  then goto (S)
- (M)  $\text{FLAG} \leftarrow \text{TRUE}; B \leftarrow B \cup \bar{P}(j)$
- (N)  $\text{LOG}(j) \leftarrow \text{TRUE}$
- (O) Next  $j$
- (P) If  $\text{FLAG} = \text{TRUE}$  then goto (F)
- (Q) If  $B = C$  then  $A \leftarrow C; \text{goto}$  (C)
- (R)  $n \leftarrow n+1; P(n) \leftarrow B; \bar{P}(n) \leftarrow C; A \leftarrow B; \text{goto}$  (C)
- (S) Next  $i$
- (T) End

Remarks:

How much storage space should be reserved for the arrays  $P, \bar{P}, LOG$  ?

In other words, how many pseudo-closed sets do we expect? We do not know the exact maximum, but it is easy to give examples with as many as  $\binom{m}{2}$  pseudo-closed sets. However, the full list of implications seems to be of little interest in these examples. In practice one will therefore give a "reasonable" dimension to the arrays  $P, \bar{P}$  and  $LOG$ .

If there are too many pseudo-closed sets there is also the possibility to compute only those pseudo-closed sets whose cardinality does not exceed some fixed number  $C_{\max}$ . This can be achieved by introducing an extra line after line (M) of the algorithm:

(M1) If  $|B| > C_{\max}$  then goto (S) .

Note that lines (C),(D),(L),(S),(T) of 2.6

correspond to (A),(B1),(B3),(C),(D) of 1.5.

Lines (E) — (P) correspond to (B2) of 1.5,

there the pseudo-closure is computed. The computation is based on prop. 2.4.2.

§ 3 APPLICATIONS IN CONCEPT ANALYSIS

Formal concept analysis has been introduced by WILLE as a mathematical tool for conceptual data analysis. We recall the basic notions in short:

A CONTEXT is a triple  $(G, M, I)$ , where  $G$  and  $M$  are sets (of OBJECTS and ATTRIBUTES, resp.) and where  $I$  is a relation  $I \subseteq G \times M$ . We read  $(g, m) \in I$  as "the object  $g$  has the attribute  $m$ ".

For  $A \subseteq G$ , we define

$$A' := \{m \in M \mid (g, m) \in I \text{ for all } g \in A\}$$

and dually for  $B \leq M$

$$B' := \{g \in G \mid (g, m) \in I \text{ for all } m \in B\}.$$

A pair  $(A, B)$  is a CONCEPT of  $(G, M, I)$  if  $A \leq G$ ,  $B \leq M$ ,  $A' = B$  and  $B' = A$ . If  $(A, B)$  is a concept then  $A$  is called the EXTENT and  $B$  then INTENT of  $(A, B)$ .

Clearly any concept  $(A, B)$  of  $(G, M, I)$  is uniquely determined both by its extent  $A$  and by its intent  $B$ . Moreover, a set  $A \leq G$  is an extent of some concept of  $(G, M, I)$  if and only if  $A = B'$  for some  $B \leq M$ , and this is equivalent to  $A = A''$ . It is therefore not ambiguous to call such a set  $A$  an extent of  $(G, M, I)$ . Dually, an intent of  $(G, M, I)$  is a set  $B \leq M$  satisfying  $B = B''$ .

3.1. EXAMPLE: A small context  $(G, M, I)$

can conveniently be represented by a rectangular  $G \times M$ -table, with crosses

indicating the relation  $I$ . In fig. 2

	a	b	c	d	e
1	X	X			X
2		X	X		
3	X		X	X	X
4		X		X	

fig. 2

$G = \{1, 2, 3, 4\}$ ,  $M = \{a, b, c, d, e\}$  and, e.g.  $(1, a) \in I$  but  $(1, c) \notin I$ . In this example  $(\{1, 4\}, \{b, d\})$  is a concept while  $(\{1, 2\}, \{b\})$  is not.

The starting point of formal concept analysis is the observation that the two mappings

$$\iota: 2^G \rightarrow 2^M \quad \text{and} \quad \iota': 2^M \rightarrow 2^G$$

form a Galois-connection. As an immediate consequence one obtains that the family of all extents and the family of all intents of  $(G, M, I)$  both are closure systems, the corresponding closure operators are the mappings  $X \mapsto X''$  on  $G$  and  $M$ , resp.

One of the basic tasks of concept analysis is the generation of all concepts of a given context. This problem has been looked at,

in different terms, by several authors ( e.g. [FAY\[1\]](#), [NORRIS\[3\]](#)).

Since a concept is uniquely determined by its intent and since the intents form a closure system, we can apply the results of § 1 to determine all concepts. The following algorithm is a version of 1.7, modified for our special purpose:

## 2.2 ALGORITHM "NEXT CONCEPT"

Purpose: Computes the successor of any given concept  $(A,B)$  of the context  $(G,M,I)$ . Here  $M = \{m_1, m_2, \dots, m_m\}$ , and the concepts are ordered lexicographically by their intents.

### Input/Output:

A number  $r \geq 0$

a list  $i_0 < i_1 < \dots < i_r$  of numbers (all  $\leq m$ )

a list  $A_0 \supset A_1 \supset \dots \supset A_r$  of extents

a list  $B_0 \subset B_1 \subset \dots \subset B_r$  of intents

satisfying the following conditions:

$$i_0 = 0, A_0 = G, A_r = A$$

$$A'_j = B_j, B'_j = A_j \quad \text{for } j = 0, 1, 2, \dots, r$$

$$B_j = (B_{j-1} \cup \{m_{i_j}\})'' \quad \text{for } j = 1, 2, \dots, r$$

$$\left. \begin{array}{l} m_{i_j} \text{ is the smallest} \\ \text{element of } B_j \setminus B_{j-1} \end{array} \right\} \quad \text{for } j = 1, 2, \dots, r$$

Algorithm:

- (A) For  $i = m, m-1, \dots, 1$  step -1  
 (B) If  $m_i \notin B_r$  then goto (P)  
 (C) If  $i < i_r$  then  $r \leftarrow r-1$ ; goto (C)  
 (D)  $i_{r+1} \leftarrow i$ ;  $A_{r+1} \leftarrow A_r \cap \{m_i\}$   
 (E) For  $j = 1, 2, \dots, i-1$   
 (F) If  $\{m_j\}' \supseteq A_{r+1}$ , then goto (P)  
 (G) Next  $j$   
 (H)  $B_{r+1} \leftarrow B_r \cup \{m_i\}$   
 (I) For  $j = i+1, i+2, \dots, m$   
 (J) If  $j \in B_{r+1}$  then goto (L)  
 (K) If  $\{m_j\}' \supseteq A_{r+1}$  then  $B_{r+1} \leftarrow B_{r+1} \cup \{m_j\}$   
 (L) Next  $j$   
 (M)  $r \leftarrow r+1$   
 (N) Return  
 (P) Next  $i$   
 (Q) Stop: No successor (input was  $B_r = M$ ).

Remarks:

The input list and the output list have the same structure. If the input satisfies the above conditions then the output automatically does and can be used for the next input (except when stop was reached). The pair  $(A_r, B_r)$  of the output list is the next concept. If the program is initially called with

$$r = 0 = i_0, A_0 = G, B_0 = G'$$

then successive runs lead through all concepts of  $(G, M, I)$ .

To obtain a correct input list for computing the successor of an arbitrary concept  $(A, B)$ , compute (cf. 1.9):

$$i_0 := 0, A_0 := G, B_0 := G'$$

and, recursively

```

if  $B \setminus B_j = \emptyset$  then  $r := j$ 
else  $i_{j+1} := \min\{x \mid m_x \in B \setminus B_j\}$ 
 $A_{j+1} := (B_j \cup \{m_{i_j}\})'$ 
 $B_{j+1} := A_{j+1}'$ 

```

**3.3 EXAMPLE:** The concepts of the context in example 3.1 as produced by 3.2:

The first input and also the first concept is

$$A_0 = \{1, 2, 3, 4\}, B_0 = \emptyset, r = 0$$

The later values are:

Concept no.	r	$i_1$ $i_2$ $i_3$	$A_1$ $A_2$ $A_3$	$B_1$ $B_2$ $B_3$
2	1	4	{1, 3, 4}	{d}
3	1	3	{2, 3}	{c}
4	1	2	{1, 2, 4}	{b}
5	2	2, 4	{1, 2, 4}, {1, 4}	{b}, {b, d}
6	2	2, 3	{1, 2, 4}, {2}	{b}, {b, c}
7	1	1	{1, 3}	{a, d}
8	2	1, 3	{1, 3}, {3}	{a, d}, {a, c, d, e}
9	2	1, 2	{1, 3}, {1}	{a, d}, {a, b, d}
10	3	1, 2, 3	{1, 3}, {1}, $\emptyset$	{a, d}, {a, b, d}, {a, b, c, d, e}

Martin Skorsky has used this algorithm to compute concept systems with approximately  $10^6$  concepts.

Another fundamental problem is to study the "logic" of a context, i.e. to describe the dependencies between the attributes. A typical such dependency would be "every object having the attributes  $a, b, c, \dots$  also has the attributes  $x, y, z, \dots$ ". This is equivalent to  $\{x, y, z, \dots\} \subseteq \{a, b, c, \dots\}$ ". In other words it is the same as the implication  $\{a, b, c, \dots\} \rightarrow \{x, y, z, \dots\}$  for the closure system of the intents.

We have prepared for this in §2, where the pseudo-closed sets were introduced to describe the implications. Specialized to our purpose, the definitions are:

A QUASI-INTENT of a context  $(G, M, I)$  is a set  $Q \subseteq M$  such that  $X \subseteq Q$ ,  $X'' \neq Q$  implies  $X'' \subseteq Q$ , for all  $X \subseteq M$ .

A PSEUDO-INTENT is a set  $P \subseteq M$  such that

- i)  $P$  is a quasi-intent
- ii)  $P$  is not an intent
- iii) If  $Q \subseteq P$  is a quasi-intent with  $Q'' = P$  then  $Q = P$ .

It was shown in 2.3 that the set of implications

$$\{P \rightarrow P'' \mid P \text{ pseudo-intent}\}$$

is a complete and reduced set, i.e. a minimal set from which all other implications follow. From proposition 2.5 we conclude a method for deciding whether a given set is a quasi-intent or a pseudo-intent:

#### 3.4 PROPOSITION:

- 1) A set  $Q \subseteq M$  is a quasi-intent of  $(G, M, I)$  if and only if  $Q \cap \{g\}' = (Q \cap \{g\}')''$  for every  $g \in G$  with  $\{g\}' \not\subseteq Q$ .
- 2) A set  $P \subseteq M$  is a pseudo-intent of  $(G, M, I)$  if and only if it is not an intent and satisfies for all  $R \subseteq M$  the following:  
If  $R$  is a pseudo-intent and properly contained in  $P$ , then  $R'' \subseteq P$ .

The set of all pseudo-intents can be computed with algorithm 2.6.

A particularly interesting application occurs if the context is not explicitly available because the number of objects is too large. We then use a version of algorithm 2.6 which allows the user to enlarge the context while the algorithm is running. The result is a program for a kind of "computer-aided theorem proving".

We shall not give this program in detail. The reader who has mastered the details of algorithm 2.6 will have no difficulties to write the algorithm after reading the following example, due to R.WILLE.

The objects in this example are the binary relations, i.e. subsets  $R \subseteq S \times S$ , where  $S$  is an arbitrary set. We consider the following properties that binary relations may or may not have:

	name	definition
r	reflexive	$xRx$ for all $x \in S$
i	irreflexive	$\neg xRx$ for all $x \in S$
s	symmetric	$xRy \Rightarrow yRx$ for all $x, y \in S$
as	asymmetric	$xRy \Rightarrow \neg yRx$ for all $x, y \in S$
an	antisymmetric	$xRy$ and $yRx \Rightarrow x=y$ for all $x, y \in S$
t	transitive	$xRy$ and $yRz \Rightarrow xRz$ for all $x, y, z \in S$
nt	negatively tr.	$\neg xRy$ and $\neg yRz \Rightarrow \neg xRz$ for all $x, y, z \in S$
c	complete	$xRy$ or $yRx$ for all $x, y \in S$
sc	strongly compl.	$xRy$ or $yRx$ for all $x, y \in S$

What are the implications that hold between these properties? For a single implication it is not difficult to decide if it holds or not: We can either prove it or give a counterexample. There are only finitely many possible implications, but far too many to test them all. What we aim for is an efficient way of finding counterexamples and "good" implications.



As long as we have no examples, all implications are possible. Therefore we start by considering all relations on a one-element and on a two-element set. There are, up to isomorphism, twelve such relations:

S	R	r	i	s	a	s	a	n	t	n	t	c	s	c
$\{0\}$	$\emptyset$	X	X	X	X	X	X	X	X	X	X	X	X	X
$\{0\}$	$\{(0,0)\}$	X		X		X	X	X	X	X	X	X	X	X
$\{0,1\}$	$\emptyset$		X	X	X	X	X	X	X	X	X	X	X	X
$\{0,1\}$	$\{(0,0)\}$			X		X	X	X	X	X	X	X	X	X
$\{0,1\}$	$\{(0,0), (1,1)\}$	X		X		X	X	X	X	X	X	X	X	X
$\{0,1\}$	$\{(0,0), (0,1)\}$					X	X	X	X	X	X	X	X	X
$\{0,1\}$	$\{(0,0), (1,0)\}$					X	X	X	X	X	X	X	X	X
$\{0,1\}$	$S \times S \setminus \{(0,1)\}$	X				X	X	X	X	X	X	X	X	X
$\{0,1\}$	$S \times S \setminus \{(0,0)\}$			X						X	X	X	X	X
$\{0,1\}$	$\{(0,1)\}$		X		X	X	X	X	X	X	X	X	X	X
$\{0,1\}$	$\{(0,1), (1,0)\}$		X	X						X	X	X	X	X
$\{0,1\}$	$S \times S$	X		X						X	X	X	X	X

Now we have a context to start with (in general, this context may be empty). Of course, only those implications can hold for all binary relations that hold for this context, but the converse is not true. Note that four of our examples are superfluous (those indicated with the arrows) because their intents are obtainable as intersections of other intents and therefore respect all implications which the others respect. We shall omit these in the sequel.

Now we use algorithm 2.6 to compute the pseudo-intents. The

lectically first pseudo-intent turns out to be  $\{sc\}$ , with

$\{sc\}'' = \{r, t, nt, c, sc\}$ . In other words, the implication

$$\{sc\} \rightarrow \{r, t, nt, c, sc\}$$

holds in all our examples. Does it hold in general? Of course

not. A counterexample is, e.g.  $S = \{0,1,2\}$ ,  $R = S \times S \setminus \{(0,1), (1,2), (2,0)\}$ .

This relation is reflexive, antisymmetric, complete and strongly complete and has none of the other properties. We include this example in our context and ask again for the smallest pseudo-intent. Still, this is  $\{sc\}$ , but now  $\{sc\}'' = \{r, c, sc\}$ , and we have to check if the implication  $\{sc\} \rightarrow \{r, c, sc\}$ , which we abbreviate to

$$sc \rightarrow r, c,$$

holds for all relations. It clearly does, every strongly complete relation is complete and reflexive.

The next pseudo-intent produced by 2.6 is  $\{t, c\}$ , with  $\{t, c\}'' = \{t, nt, c\}$ . This suggests the implication

$$t, c \rightarrow nt,$$

which indeed holds for binary relations. The same is true for the next suggested implication

$$an, nt \rightarrow t,$$

which comes from the pseudo-intent  $\{an, nt\}$  with  $\{an, nt\}'' = \{an, t, nt\}$ . Then we obtain the pseudo-intent  $\{as\}$ , and  $\{as\}'' = \{i, as, an, t, nt\}$  in our context of examples. But the implication

$$as \rightarrow i, an, t, nt$$

does not hold in general, as the following example shows:

Let  $S = \{0, 1, 2\}$ ,  $R = \{(0, 1), (1, 2), (2, 0)\}$ . This relation has the properties  $i, as, an, nt$ , and we include it in our list.

Since we have just changed our context of examples, one might think that we have to start algorithm 2.6 from the beginning. Fortunately this is not necessary, according to the following proposition:

3.4 PROPOSITION:

Let  $g$  be an object of the context  $(G, M, I)$  and denote by  $(G_g, M, I_g)$  the context obtained from  $(G, M, I)$  by deleting the object  $g$ ,

i.e.  $G_g := G \setminus \{g\}$ ,  $I_g := I \cap (G_g \times M)$ .

Furthermore, let  $A \subseteq M$  be some set.

If  $\{g\}'$  respects all implications of the form  $P \rightarrow P''$ , where  $P$  is a pseudo-intent of  $(G_g, M, I_g)$  which is lexicographically smaller than  $A$ , then the pseudo-intents of  $(G, M, I)$  which are smaller than  $A$  are the same as those of  $(G_g, M, I_g)$ .

Proof: Consider the smallest counterexample and apply 2.5.2 to obtain a contradiction.

By this proposition no new pseudo-intent less than our current one can occur. Therefore we can continue with algorithm 2.6 for the changed context. Still  $\{as\}$  is an undecided pseudo-intent, but now, in the extended context,  $\{as\}'' = \{i, an, as\}$ , which suggests the implication

$$as \rightarrow i, an$$

which in fact holds for all relations.

The next two implications

$$s, c \rightarrow nt$$

$$s, an \rightarrow t$$

also hold, then

$$i, t \rightarrow as, an, nt$$

can be disproved by a counterexample, etc.

The complete result is given in the following tables.

A complete list of examples:

No	S	R
1	{0}	$\emptyset$
2	{0}	$\{(0,0)\}$
3	{0,1}	$\emptyset$
4	{0,1}	$\{(0,0),(1,1)\}$
5	{0,1}	$S \times S \setminus \{(0,1)\}$
6	{0,1}	$\{(0,1)\}$
7	{0,1}	$\{(0,1),(1,0)\}$
8	{0,1}	$S \times S$
9	{0,1,2}	$S \times S \setminus \{(0,1),(1,2),(2,0)\}$
10	{0,1,2}	$\{(0,1),(1,2),(2,0)\}$
11	{0,1,2}	$\{(0,1)\}$
12	{0,1,2}	$\{(0,1),(1,0)\}$
13	{0,1,2}	$S \times S \setminus \{(0,1)\}$
14	{0,1,2}	$S \times S \setminus \{(0,1),(1,0)\}$

The reduced context of examples

	r	i	s	a	s	a	n	t	c	s	c
1		x	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					
5	x				x	x	x	x	x		
6		x		x	x	x	x	x			
7		x	x					x	x		
8	x		x			x	x	x	x		
9	x				x				x	x	
10		x		x	x				x		
11		x		x	x	x					
12		x	x								
13	x							x	x	x	
14	x		x								

A reduced and complete list of implications

$sc \rightarrow r, c$   
 $t, c \rightarrow nt$   
 $an, nt \rightarrow t$   
 $as \rightarrow i, an$   
 $s, c \rightarrow nt$   
 $s, an \rightarrow t$   
 $i, t \rightarrow as, an$   
 $i, an \rightarrow as$   
 $i, s, as, an, t \rightarrow nt$   
 $r, c \rightarrow sc$   
 $r, nt \rightarrow c, sc$   
 $r, s, nt, c, sc \rightarrow t$   
 $r, i \rightarrow \text{all properties.}$

Note that the premises of the above implications are exactly the pseudo-intents of the context.

It is not guaranteed by our method that the list of counterexamples is reduced (as it is in this example), the addition of new examples may make previous examples superfluous. The elimination of reducible examples has no influence on the implications.

---

REFERENCES:

- [1] G.FAY, An algorithm for finite Galois connections. Technical report, Institute for Industrial Economy, Budapest(1973).
- [2] J.-L. GUIGUES and V. DUQUENNE, Informative implications derived from a table of binary data. Preprint, Groupe Mathématiques et Psychologie, Université René Descartes, Paris (1984).
- [3] E.M.NORRIS, An algorithm for computing the maximal rectangles in a binary relation. Rev. Roum. Math. Pures et Appl., Tome XXIII, No 2, pp 243-250, Bucarest (1978).
- [4] R.WILLE, Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. RIVAL (ed.), Ordered sets. Reidel Publ. Comp., Dordrecht-Boston (1982), pp 445-470.

---

Technische Hochschule Darmstadt  
 Fachbereich Mathematik  
 Forschungsgruppe Begriffsanalyse  
 D-6100 Darmstadt, West Germany.