Formal Concept Analysis: Methods and Applications in Computer Science

Bernhard Ganter
TU Dresden
ganter@math.tu-dresden.de

Summer 2002

Adapted and extended by:
Gerd Stumme
Universität Kassel
stumme@cs.uni-kassel.de

Used for the FCA lecture:
Robert Jäschke
Leibniz Universität Hannover
jaeschke@L3S.de

Winter 2015/2016

Contents

Chapter 1

Formal Concept Analysis in a nutshell

This section is meant to be an ‘appetizer’. It provides a brief overview over Formal Concept Analysis, in order to allow for a better understanding of the overall picture. This section introduces the most basic notions of Formal Concept Analysis, namely formal contexts, formal concepts, concept lattices, many-valued contexts and conceptual scaling. These definitions are repeated and discussed in more detail in the remainder of the book.

1.1 Formal contexts, concepts, and concept lattices

Formal Concept Analysis (FCA) was introduced as a mathematical theory modeling the concept of ‘concepts’ in terms of lattice theory. To allow a mathematical description of extensions and intentions, FCA starts with a *(formal) context.*

Definition 1 A *(formal) context* is a triple $\mathbb{K} := (G, M, I)$, where G is a set whose elements are called *objects*, M is a set whose elements are called *attributes*, and I is a binary relation between G and M (i. e. $I \subseteq G \times M$). $(g, m) \in I$ is read “object g *has* attribute m”. ◊

Figure 1.1 shows a formal context where the object set G comprises all airlines of the Star Alliance group and the attribute set M lists their destinations. The binary relation I is given by the cross table and describes which destinations are served by which Star Alliance member.

Definition 2 For $A \subseteq G$, let

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

and, for $B \subseteq M$, let

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

A *(formal) concept* of a formal context (G, M, I) is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. The sets A and B are called the *extent* and the *intent* of the formal concept (A, B), respectively. The *subconcept–superconcept relation* is formalized by

$$(A_1, B_1) \leq (A_2, B_2) :\Longleftrightarrow A_1 \subseteq A_2 \quad (\Longleftrightarrow B_1 \supseteq B_2) \ .$$

	Latin America	Europe	Canada	Asia Pacific	Middle East	Africa	Mexico	Caribbean	United States
Air Canada	×	×	×	×	×		×	×	×
Air New Zealand		×		×					×
All Nippon Airways		×		×					×
Ansett Australia				×					
The Austrian Airlines Group		×	×	×	×	×			×
British Midland		×							
Lufthansa	×	×	×	×	×	×	×		×
Mexicana	×		×				×	×	×
Scandinavian Airlines	×	×		×		×			×
Singapore Airlines		×	×	×	×	×			×
Thai Airways International	×	×		×				×	×
United Airlines	×	×	×	×			×	×	×
VARIG	×	×		×		×	×		×

Figure 1.1: A formal context about the destinations of the Star Alliance members

The set of all formal concepts of a context $\mathbb{K}$ together with the order relation $\leq$ is always a complete lattice,[1] called the *concept lattice* of $\mathbb{K}$ and denoted by $\underline{\mathfrak{B}}(\mathbb{K})$. $\Diamond$

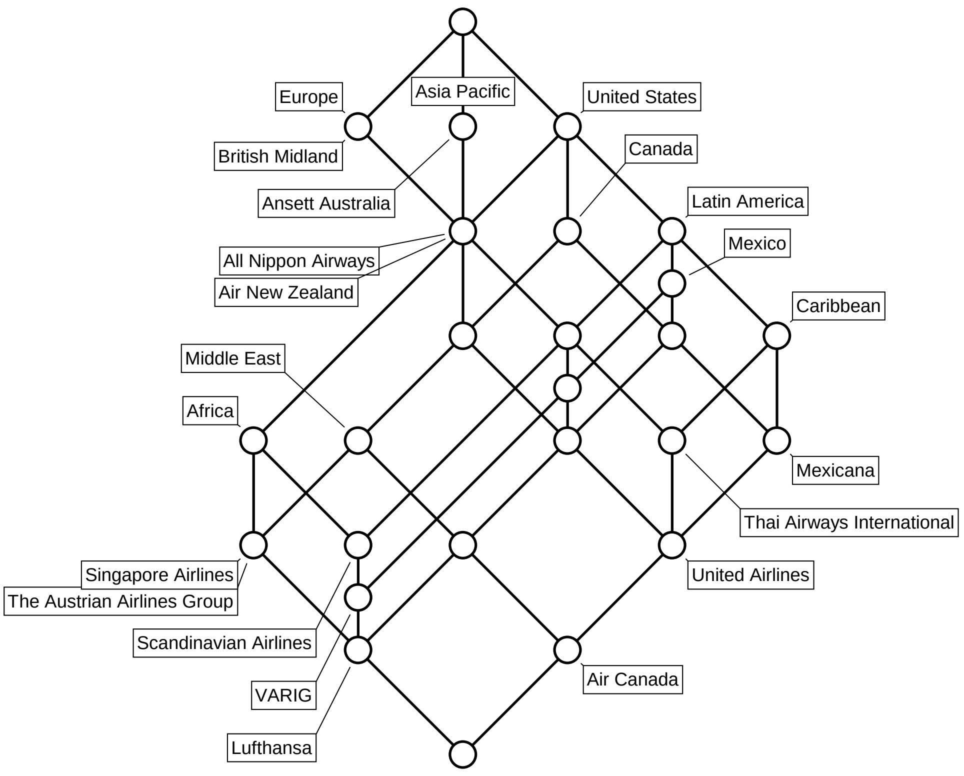


Figure 1.2: The concept lattice of the context in Figure 1.1

[1] I. e., for each subset of concepts, there is always a unique greatest common subconcept and a unique least common superconcept.

Figure 1.2 shows the concept lattice of the context in Figure 1.1 by a *line diagram.* In a line diagram, each node represents a formal concept. A concept $\mathfrak{c}_1$ is a subconcept of a concept $\mathfrak{c}_2$ if and only if there is a path of descending edges from the node representing $\mathfrak{c}_2$ to the node representing $\mathfrak{c}_1$. The name of an object g is always attached to the node representing the smallest concept with g in its extent; dually, the name of an attribute m is always attached to the node representing the largest concept with m in its intent. We can read the context relation from the diagram because an object g has an attribute m if and only if the concept labeled by g is a subconcept of the one labeled by m. The extent of a concept consists of all objects whose labels are attached to subconcepts, and, dually, the intent consists of all attributes attached to superconcepts. For example, the concept labeled by 'Middle East' has {Singapore Airlines, The Austrian Airlines Group, Lufthansa, Air Canada} as extent, and {Middle East, Canada, United States, Europe, Asia Pacific} as intent.

In the top of the diagram, we find the destinations which are served by most of the members: Europe, Asia Pacific, and the United States. For instance, beside British Midland and Ansett Australia, all airlines are serving the United States. Those two airlines are located at the top of the diagram, as they serve the fewest destinations — they operate only in Europe and Asia Pacific, respectively.

The further we go down in the concept lattice, the more globally operating are the airlines. The most destinations are served by the airlines at the bottom of the diagram: Lufthansa (serving all destinations beside the Caribbean) and Air Canada (serving all destinations beside Africa). Also, the further we go down in the lattice, the fewer served are the destinations. For instance, Africa, the Middle East, and the Caribbean are served by relatively few Star Alliance members.

Dependencies between the attributes can be described by implications. For $X, Y \subseteq M$, we say that the *implication* $X \to Y$ *holds* in the context, if each object having all attributes in X also has all attributes in Y. For instance, the implication {Europe, United States} $\to$ {Asia Pacific} holds in the Star Alliance context. It can be read directly in the line diagram: the largest concept having both 'Europe' and 'United States' in its intent (i. e., the concept labeled by 'All Nippon Airways' and 'Air New Zealand') also has 'Asia Pacific' in its intent. Similarly, one can detect that the destinations 'Africa' and 'Canada' together imply the destination 'Middle East' (and also 'Europe', 'Asia Pacific', and 'United States').

Concept lattices can also be visualized in *nested line diagrams.* For obtaining a nested line diagram, one splits the set of attributes in two parts, and obtains thus two formal contexts. For each formal context, one computes its concept lattice and a line diagram. The nested line diagram is obtained by enlarging the nodes of the first line diagram and by drawing the second diagram inside. The second lattice is used to further differentiate each of the extents of the concepts of the first lattice. Figure 1.3 shows a nested line diagram for the Star Alliance context. It is obtained by splitting the attribute set as follows: $M = \{$ Europe, Asia Pacific, Africa, Middle East $\} \cup \{$ United States, Canada, Latin America, Mexico, Caribbean $\}$. The order relation can be read by replacing each of the lines of the large diagram by eight parallel lines linking corresponding nodes in the inner diagrams. The concept lattice given in Figure 1.2 is embedded (as a join–semilattice) in this diagram, it consists of the bold nodes. The concept mentioned above (labeled by 'Middle East') is for instance represented by the left-most bold node in the lower right part.

The bold concepts are referred to as 'realized concepts', as, for each of them, the set of all attributes labeled above is an intent of the formal context. The non-realized concepts are not only displayed to indicate the structure of the inner scale, but also because they indicate implications: Each non-realized concept indicates that the attributes in its intent imply the attributes contained in the largest realized concept below. For instance, the first implication discussed above is indicated

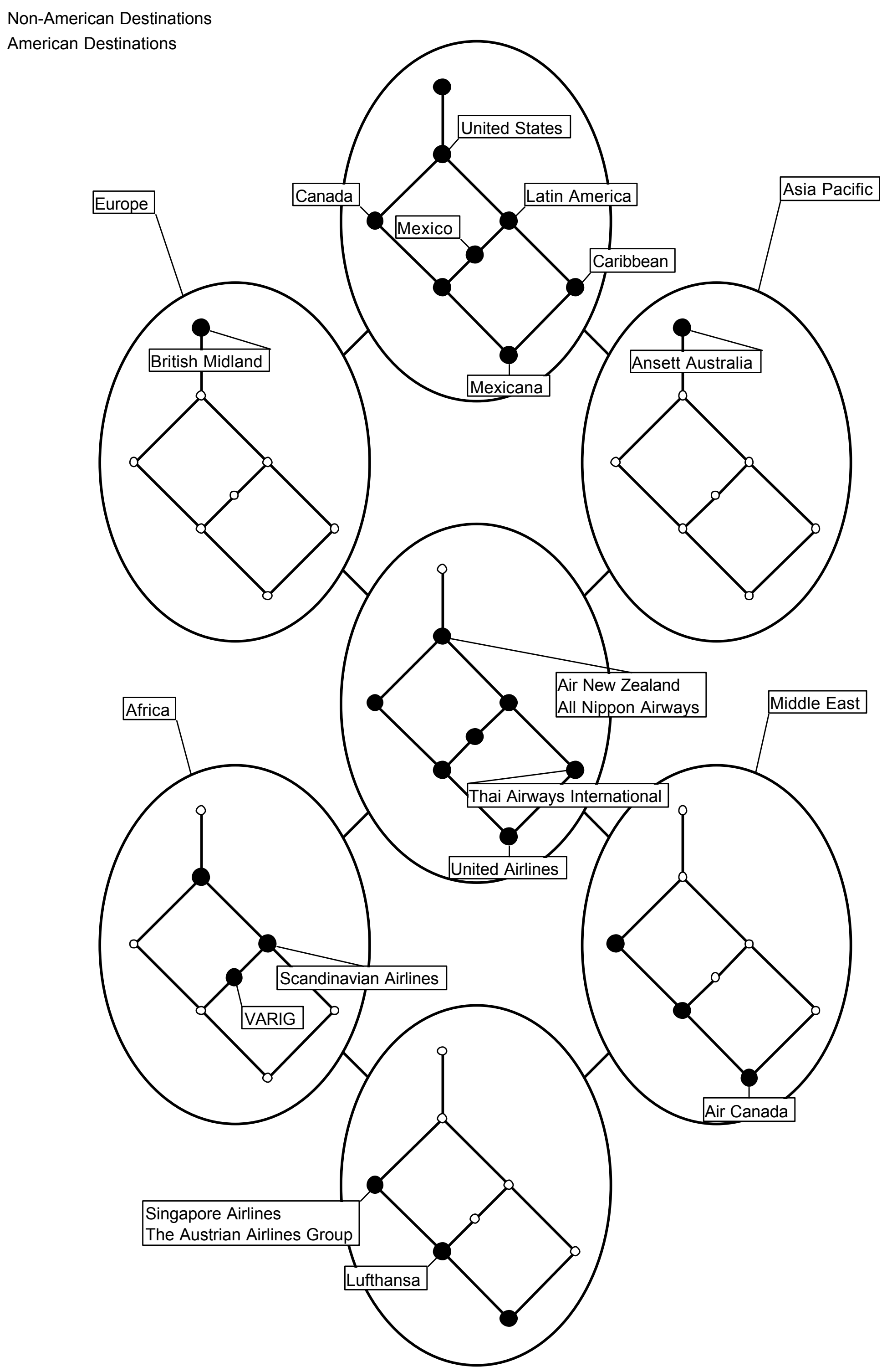


Figure 1.3: A nested diagram of the concept lattice in Figure 1.2

by the non-realized concept having as intent 'Europe' and 'United States', it is represented by the empty node below the concept labeled by 'British Midland'. The largest realized sub-concept below is the one labeled by 'All Nippon Airways' and 'Air New Zealand' — which additionally has 'Asia Pacific' in its intent. Hence the implication { Europe, United States } → { Asia Pacific } holds. The second implication from above is indicated by the non-realized concept left of the concept labeled by 'Scandinavian Airlines', and the largest realized concept below, which is the one labeled by 'Singapore Airlines' and 'The Austrian Airlines Group'.

1.2 Many-valued contexts and conceptual scaling

In most applications, there are not only Boolean attributes in the databases. The conceptual model we use for this is a *many-valued context*. In database terms, a many-valued context is a relation of a relational database with one key attribute whose domain is the set G of objects. Here, we consider only one (denormalized) database relation at a time.

In order to obtain a concept lattice from a many-valued context, it has to be translated into a one-valued (formal) context. The translation process is described by *conceptual scales*:

Definition 3 A *many-valued context* is a tuple $(G, M, (W_m)_{m\in M}, I)$ where G and M are sets of *objects* and *attributes*, resp., W_m is a set of *values* for each $m \in M$, and $I \subseteq G \times \bigcup_{m\in M}(\{m\} \times W_m)$ is a relation such that $(g, m, w_1) \in I$ and $(g, m, w_2) \in I$ imply $w_1 = w_2$.

A *conceptual scale* for an attribute $m \in M$ is a one-valued context $\mathbb{S}_m := (G_m, M_m, I_m)$ with $W_m \subseteq G_m$. The context $\mathbb{R}_m := (G, M_m, J_m)$ with $gJ_mn :\iff \exists w{\in}W_m\colon\ (g, m, w){\in}I \wedge (w, n){\in}I_m$ is called the *realized scale* for the attribute $m \in M$.

A *Conceptual Information System* consists of a many-valued context together with a set of conceptual scales. ◊

The set M consists of all attributes of the database scheme, while the sets M_m contain the attributes which are shown to the user by the Conceptual Information System.

Example 1 The information system INFO-80 supports planning, realization, and control of business transactions related to flight movements at Frankfurt Airport. A Conceptual Information System has been established, based on the TOSCANA software, in order to facilitate access to the data of INFO-80 ([21]).

Here we consider 18389 outbound flight movements effectuated at Frankfurt Airport during one month. For each of these flight movements 110 attributes are registered automatically and stored in a database (e. g., destination, estimated time of departure, actual time of departure, number of passengers, terminal position, . . .). 77 conceptual scales have been designed (some of the 110 database attributes were of minor importance for data analysis). By combining the scales and zooming into them with other scales, the analyst can explore dependencies and irregularities.

Figure 1.4 shows the realized scale *Destination Southern Europe/Mediterranean Sea*. Because of the large number of flight movements, TOSCANA displays only the number of flight movements assigned to each concept. If desired, the user can drill-down to the flight number and to more detailed information. For instance, the concept labeled by 'Marokko' (Morocco) has 53 objects (i. e., flight numbers) in its extent, and the attributes 'Marokko' and 'Nordafrika' (Northern Africa) in its intent. Its upper neighbor (which is labeled by 'Nordafrika') has $53+14+111 = 178$ objects in its extent, and 'Nordafrika' in its intent.

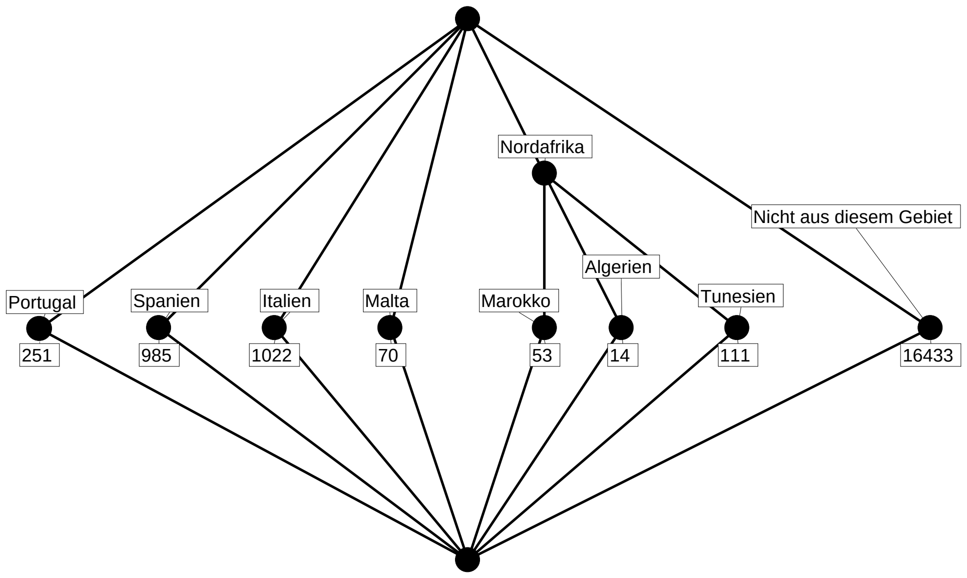


Figure 1.4: Realized scale *Destination Southern Europe/Mediterranean Sea*

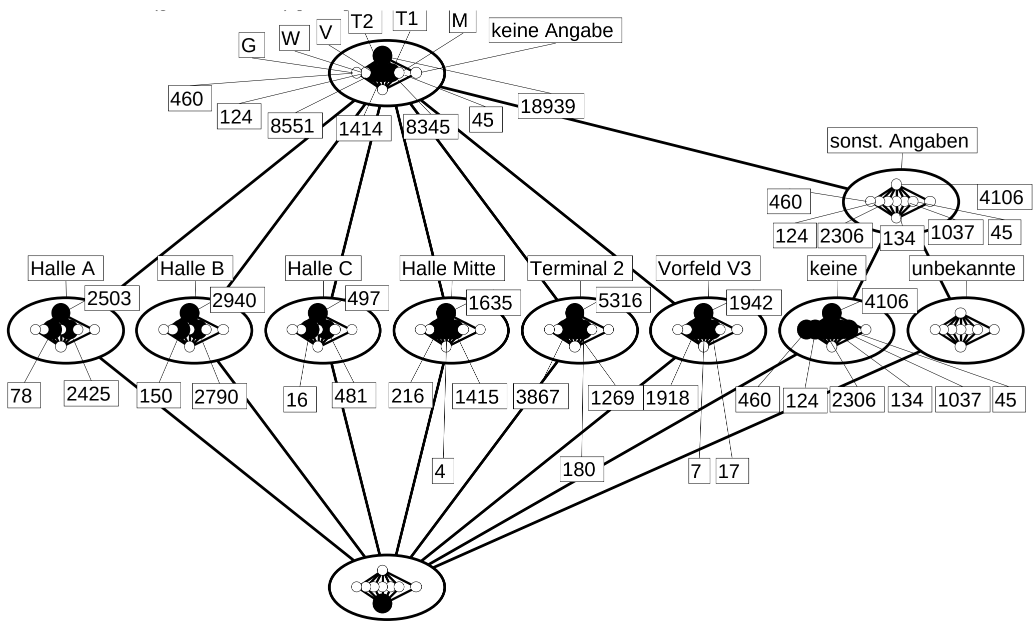


Figure 1.5: Nested line diagram of the scales *Position of Baggage Conveyor* and *Position of Aircraft*

For analyzing how objects are distributed over two different scales, one can combine the two conceptual scales. The result of combining the two conceptual scales *Position of Baggage Conveyor* and *Position of Aircraft* is displayed in the nested line diagram in Figure 1.5.

Each of the 17 lines of the outer scale represents seven parallel lines linking corresponding concepts of the inner scale. The concept lattice we are interested in is embedded in this *direct product*. The embedding is indicated by the bold circles.

Again, each non-realized concept (i. e., the empty circles) indicates an implication. For example, the leftmost circle in the leftmost ellipse has the attribute 'Halle A' (hall A) and 'G' (general aviation, i. e., private airplanes) in its intent. These two attributes form the premise of the implication. The conclusion of the implication is given by the intent of the largest realized concept below. In this case, it is the bottom concept which has all attributes in its extent. Hence hall A as position of the baggage conveyor and the general aviation apron as position of the aircraft implies everything else. This means that the premise is not realized by any object. There are no general aviation aircraft with a baggage conveyor assigned at hall A. There are concepts in the nested line diagram which are expected to be non-realized as well. For instance, an aircraft at Terminal 1 should not have a baggage conveyor assigned at Terminal 2. But in the diagram, we see that the concept with these two attributes is realized. There are 180 flight movements having the attribute 'Terminal 2' of the outer scale and the attribute 'T1' of the inner scale.

We can detect some more unnormal combinations. There are four aircraft that docked at Terminal 2, while their assigned baggage conveyors are at Terminal 1. To seven aircraft at Terminal 2 and 17 aircraft at Terminal 1, conveyors on the 'Vorfeld V3' (Apron V3) were assigned. In all these cases, one can drill down to the original data by clicking on the number to obtain the flight movement numbers, which in turn lead to the data sets stored in the INFO-80 system.

The knowledge that the combinations are unnormal, is not coded explicitly in INFO-80, but is only implicitly present as expert knowledge. Since these combinations happen at the airport, they cannot be forbidden by database constraints. But once such conflicts are discovered, one can implement an alarm which informs the operator of the baggage transportation system about such cases.

If there are too many scales involved in the knowledge discovery process, then zooming into the outer scale reduces the size of the displayed diagram. For instance, by zooming into the concept labeled by 'Terminal 2' in Fig. 1.5, we obtain the diagram of the conceptual scale *Position of Aircraft*, but with the objects being restricted to those flight movements where the assigned baggage conveyor is at Terminal 2. Then one can continue the exploration by adding a new conceptual scale on the inner level. This interactive human-centered knowledge discovery process is described in detail in [**?**].

This section gave a short introduction to the core notions of FCA. We will discuss them (and more advanced topics) in more detail in the remainder of this book.

Part I

Contexts, Concepts, and Concept Lattices

Chapter 2

Concept lattices

Formal Concept Analysis studies how *objects* can be hierarchically grouped together according to their common *attributes*. One of the aspects of FCA thus is *attribute logic*, the study of possible attribute combinations. Most of the time, this will be very elementary. Those with a background in Mathematical Logic might say that attribute logic is just Propositional Calculus, and thus Boolean Logic, or even a fragment of this. Historically, the name *Propositional Logic* is misleading: Boole himself used the intuition of attributes ("signs") rather than of propositions. So in fact, attribute logic goes back to Boole.

But our style is different from that of logicians. Our logic is *contextual*, which means that we are interested in the logical structure of concrete data (of the *context*). Of course, the general rules of mathematical logic are important for this and will be utilized.

2.1 Basic notions

2.1.1 Formal contexts and cross tables

Definition 4 A **Formal Context** (G, M, I) consists of two sets G and M and of a binary relation $I \subseteq G \times M$. The elements of G are called the **objects**, those of M the **attributes** of (G, M, I). If $g \in G$ and $m \in M$ are in relation I, we write $(g, m) \in I$ or $g\ I\ m$ and read this as *"object g* **has** *attribute m"*. $\Diamond$

The simplest format for writing down a formal context is a **cross table**: we write a rectangular table with one row for each object and one column for each attribute, having a cross in the intersection of row g with column m iff $(g, m) \in I$. The simplest data type for computer storage is that of a bit matrix[1].

Note that the definition of a formal context is very general. There are no restrictions about the nature of objects and attributes. We may consider physical objects, or persons, numbers, processes, structures, etc., virtually everything. Anything that is a *set* in the mathematical sense may be taken as the set of objects or of attributes of some formal context. We may interchange the rôle of objects and attributes: if (G, M, I) is a formal context, then so is the **dual context** (M, G, I^{-1}) (with $(m, g) \in I^{-1} :\iff (g, m) \in I$). It is also not necessary that G and M are disjoint, they need not even be different.

On the other hand, the definition is rather restrictive when applied to real world phenomena. Language phrases like "all human beings" or "all chairs" do not denote

[1]It is not easy to say which is the *most efficient* data type for formal contexts. This depends, of course, on the operations we want to perform with formal contexts. The most important ones are the derivation operators, to be defined in the next subsection.

sets in our sense. There is no "set of all chairs", because the decision if something is a chair is not a matter of fact but a matter of subjective interpretation. The notion of "formal concept" which we shall base on the definition of "formal context" is much, much narrower than what is commonly understood as a concept of human cognition. The step from "context" to "formal context" is quite an incisive one. It is the step from "real world" to "data". Later on, when we get tired of saying "formal concepts of a formal context", we will sometimes omit the word "formal". But we should keep in mind that it makes a big difference.

2.1.2 The derivation operators

Given a selection $A \subseteq G$ of objects from a formal context (G, M, I), we may ask which attributes from M are common to all these objects. This defines an operator that produces for every set $A \subseteq G$ of objects the set $A^{\uparrow}$ of their common attributes.

Definition 5 For $A \subseteq G$, we let

$$A^{\uparrow} := \{m \in M \mid g\ I\ m \text{ for all } g \in A\} \ .$$

Dually, we introduce for a set $B \subseteq M$ of attributes

$$B^{\downarrow} := \{g \in G \mid g\ I\ m \text{ for all } m \in B\} \ .$$

These two operators are the **derivation operators** for (G, M, I). $\diamondsuit$

Thus, set $B^{\downarrow}$ denotes the set consisting of those objects in G that have all the attributes from B.

Usually, we do not distinguish the derivation operators in writing and use the notation A', B' instead. This is convenient, as long as the distinction is not explicitly needed.

If A is a set of objects, then A' is a set of attributes, to which we can apply the second derivation operator to obtain A'' (more precisely: $(A^{\uparrow})^{\downarrow}$), a set of objects. Dually, starting with a set B of attributes, we may form the set B'', which is again a set of attributes. We have the following simple facts:

Proposition 1 *For subsets $A, A_1, A_2 \subseteq G$ we have*

1. $A_1 \subseteq A_2 \Rightarrow A_2' \subseteq A_1'$,
2. $A \subseteq A''$,
3. $A' = A'''$.

Dually, for subsets $B, B_1, B_2 \subseteq M$ we have

1'. $B_1 \subseteq B_2 \Rightarrow B_2' \subseteq B_1'$,

2'. $B \subseteq B''$,

3'. $B' = B'''$.

The elementary **proof** is omitted here. The reader may confer to [16] for details. The mathematically interested reader may notice that the derivation operators constitute a **Galois connection** between the (power sets of the) sets G and M.

The not so mathematically oriented reader should try to express the statements of the Proposition in common language. We give an example: Statement 1.) says that *if a selection of objects is enlarged, then the attributes which are common to all objects of the larger selection are among the common attributes of the smaller selection.* Try to formulate 2.) and 3.) in a similar manner!

2.1.3 Formal concepts, extent and intent

In what follows, (G, M, I) always denotes a formal context.

Definition 6 (A, B) is a **formal concept** of (G, M, I) iff

$$A \subseteq G, \quad B \subseteq M, \quad A' = B, \quad \text{and } A = B'.$$

The set A is called the **extent** of the formal concept (A, B), and the set B is called its **intent**. $\Diamond$

According to this definition, a formal concept has two parts: its extent and its intent. This follows an old tradition in philosophical concept logic, as expressed in the *Logic of Port Royal, 1654* [AN68], and in the International Standard ISO 704 (translation of the German Standard DIN 2330).

The description of a concept by extent and intent is redundant, because each of the two parts determines the other (since $B = A'$ and $A = B'$). But for many reasons this redundant description is very convenient.

When a formal context is written as a cross table, then every formal concept (A, B) corresponds to a (filled) rectangular subtable, with row set A and column set B. To make this more precise, note that in the definition of a formal context there is no *order* on the sets G or M. Permuting the rows or the columns of a cross table therefore does not change the formal context it represents. A *rectangular subtable* may, in this sense, omit some rows or columns; it must be rectangular after an appropriate rearrangement of the rows and the columns. It is then easy to characterize the rectangular subtables that correspond to formal contexts: they are full of crosses and maximal with respect to this property:

Lemma 2 *(A, B) is a formal concept of (G,M,I)iff $A \subseteq G$, $B \subseteq M$, and A and B are each maximal (with respect to set inclusion) with the property $A \times B \subseteq I$.*

A formal context may have many formal concepts. In fact, it is not difficult to come up with examples where the number of formal concepts is exponential in the size of the formal context. The set of all formal concepts of (G, M, I) is denoted

$$\mathfrak{B}(G, M, I).$$

Later on we shall discuss an algorithm to compute all formal concepts of a given formal context.

2.1.4 Conceptual hierarchy

Formal concepts can be (partially) ordered in a natural way. Again, the definition is inspired by the way we usually order concepts in a *subconcept–superconcept hierarchy*: “Pig” is a subconcept of “mammal”, because every pig is a mammal. Transferring this to formal concepts, the natural definition is as follows:

Definition 7 Let (A_1, B_1) and (A_2, B_2) be formal concepts of (G, M, I). We say that (A_1, B_1) is a **subconcept** of (A_2, B_2) (and, equivalently, that (A_2, B_2) is a **superconcept** of (A_1, B_1)) iff $A_1 \subseteq A_2$. We use the $\leq$-sign to express this relation and thus have

$$(A_1, B_1) \leq (A_2, B_2) :\Longleftrightarrow A_1 \subseteq A_2.$$

The set of all formal concepts of (G, M, I), ordered by this relation, is denoted

$$\underline{\mathfrak{B}}(G, M, I)$$

and is called the **concept lattice** of the formal context (G, M, I). $\Diamond$

This definition is natural, but irritatingly asymmetric. What about the intents? Well, a look at Proposition 1 shows that for concepts (A_1, B_1) and (A_2, B_2)

$$A_1 \subseteq A_2 \quad \text{is equivalent to} \quad B_2 \subseteq B_1.$$

Therefore

$$(A_1, B_1) \leq (A_2, B_2) :\iff A_1 \subseteq A_2 \quad (\iff B_2 \subseteq B_1).$$

The concept lattice of a formal context is a partially ordered set:

Definition 8 A partially ordered set is a pair $(P, \leq)$ where P is a set, and $\leq$ is a binary relation on P (i. e.,$\leq$ is a subset of $P \times P$) which is

1. reflexive ($x \leq x$ for all $x \in P$),
2. anti-symmetric ($x \leq y \wedge x \neq y \Rightarrow \neg y \leq x$ for all $x \in P$),
3. and transitive ($x \leq y \wedge y \leq z \Rightarrow x \leq z$ for all $x, y, z \in P$),

$x \geq y$ stands for $y \leq$, and $x < y$ for $x \leq y$ with $x \neq y$. $\Diamond$

Beside concept lattices, partially ordered sets appear frequently in Mathematics and Computer Science. We provide some examples.

Example 2
- Every tree is a partially ordered set.
- The set $\mathbb{R}$ of all real numbers together with the ordinary $\leq$–relation.
- On any set M, the equality $=$ is also a (trivial) order relation.
- For a set M, the powerset $\mathfrak{P}(M)$ with set inclusion $\subseteq$ is also a partially ordered set.
- The set $\mathbb{N}$ of all natural numbers with the divides–relation $|$.

Observe that we do not assume a **total order**, which would require the additional condition $x \leq y$ or $y \leq x$ for $x, y \in P$. Observe also that partially ordered sets allow for 'multiple inheritance', as the two last examples show.

Concept lattices have additional properties beside being partially ordered sets, that is why we call them 'lattices'. This will be the topic of the next section.

2.2 The algebra of concepts

2.2.1 Reading a concept lattice diagram

The concept lattice of (G, M, I) is the set of all formal concepts of (G, M, I), ordered by the subconcept–superconcept order. Ordered sets of moderate size can conveniently be displayed as **order diagram**s. We explain how to *read* a concept lattice diagram by means of an example given in Figure 2.1. Later on we will discuss how to *draw* such diagrams.

Figure 2.1 refers to the following situation: Think of two squares, of equal size, that are drawn on paper. There are different ways to arrange two squares: they may be disjoint (= have no point in common), may overlap (= have a common interior point), may share a vertex, an edge or a line segment of the boundary (of length > 0), they may be parallel or not.

Figure 2.1 shows a concept lattice unfolding these possibilities. It consists of twelve formal concepts, these being represented by the twelve small circles in the

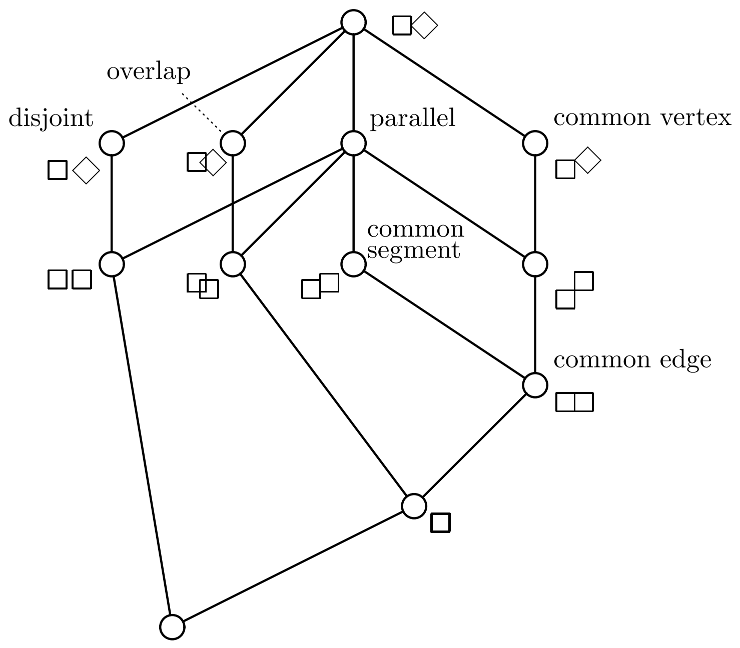


Figure 2.1: A concept lattice diagram. The objects are pairs of unit squares. The attributes describe their mutual position. See also Figure 5.6 (p. 104).

diagram. The names of the six attributes are given. Each name is attached to one of the formal concepts and is written slightly above the respective circle. The ten objects are represented by little pictures; each showing a pair of unit squares. Again, each object is attached to exactly one formal concept; the picture representing the object is drawn slightly below the circle representing the object concept.

Some of the circles are connected by edges. These express the concept order. With the help of the edges we can read from the diagram which concepts are subconcepts of which other concepts, and which objects have which attributes. To do so, one has to follow *ascending paths* in the diagram.

For example, consider the object ◫. From the corresponding circle we can reach, via ascending paths, four attributes: “common edge”, “common segment”, “common vertex”, and “parallel”. ◫ does in fact have these properties, and does not have the other ones: the two squares are neither “disjoint” nor do they “overlap”.

Similarly, we can find those objects that have a given attribute by following all descending paths starting at the attribute concept. For example, to find all objects which “overlap”, we start at the attribute concept labeled “overlap” and follow the edges downward. We can reach three objects (namely □◇, ⧉ , and □, the latter symbolizing two squares at the same position). Note that we cannot reach □□, because only at concept nodes it is allows to make a turn.

With the same method, we can read the intent and the extent of every formal concept in the diagram. For example, consider the concept circle labeled ⧉ . Its extent consists of all objects that can be reached from that circle on an descending path. The extent therefore is {⧉ , □}. Similarly, we find by an inspection of the ascending paths that the intent of this formal concept is {overlap, parallel}.

The diagram contains all necessary information. We can read off the objects, the attributes, and the incidence relation I. Thus we can perfectly reconstruct the formal context from the diagram ("the original data").[2] Moreover, for each formal concept we can easily determine its extent and intent from the diagram.

So in a certain sense, concept lattice diagrams are perfect. But there are, of course, limitations. Take another look at Figure 2.1. Is it *correct*? Is it *complete*? The answer is that, since a concept lattice faithfully unfolds the formal context, the information displayed in the lattice diagram can be only as correct and complete as the formal context is. In our specific example it is easy to check that the given examples in fact do have the properties as indicated. But a more difficult problem is if our selection of objects is representative. Are there possibilities to combine two squares, that lead to an attribute combination not occurring in our sample?

We shall come back to that question later.

2.2.2 Supremum and infimum

Can we compute with formal concepts? Yes, we can. The concept operations are however quite different from addition and multiplication of numbers. They resemble more of the operations *greatest common divisor* and *least common multiple*, that we know from integers.

Definition 9 Let $(M, \leq)$ be a partially ordered set, and A be a subset of M. A **lower bound** of A is an element s of M with $s \leq a$, for all $a \in A$. An **upper bound** of A is defined dually. If there exists a largest element in the set of all lower bounds of A, then it is called the **infimum** (or **join**) of A. It is denoted *inf* A or $\bigwedge A$. The **supremum** (or **meet**) of A (*sup* A, $\bigvee A$) is defined dually. For $A = \{x, y\}$, we write also $x \wedge y$ for their infimun, and $x \vee y$ for their supremum. $\Diamond$

Similarly as for arbitrary sums we use the symbol $\sum$ instead of $+$, we use the large symbols $\bigvee$ and $\bigwedge$ for arbitrary suprema and infima.

Lemma 3 *For any two formal concepts (A_1, B_1) and (A_2, B_2) of some formal context we obtain*

- *the infimum (***greatest common subconcept***) of (A_1, B_1) and (A_2, B_2) as*

$$(A_1, B_1) \wedge (A_2, B_2) := (A_1 \cap A_2, (B_1 \cup B_2)''),$$

- *the supremum (***least common superconcept***) of (A_1, B_1) and (A_2, B_2) as*

$$(A_1, B_1) \vee (A_2, B_2) := ((A_1 \cup A_2)'', B_1 \cap B_2).$$

It is not difficult to prove (Exercise 4) that what is suggested by this definition is in fact true: $(A_1, B_1) \wedge (A_2, B_2)$ is in fact a formal concept (of the same context), $(A_1, B_1) \wedge (A_2, B_2)$ is a subconcept of both (A_1, B_1) and (A_2, B_2), and any other common subconcept of (A_1, B_1) and (A_2, B_2) is also a subconcept of $(A_1, B_1) \wedge (A_2, B_2)$. Similarly, $(A_1, B_1) \vee (A_2, B_2)$ is a formal concept, it is a superconcept of (A_1, B_1) and of (A_2, B_2), and it is a subconcept of any common superconcept of these two formal concepts.

With some practice, one can read off infima and suprema from the lattice diagram. Choose any two concepts from Figure 2.1 and follow the descending paths from the corresponding nodes in the diagram. There is always a highest point where these paths meet, that is, a highest concept that is below both, namely, the infimum. Any other concept below both can be reached from the highest one on a

[2]This reconstruction is assured by the Basic Theorem given below.

descending path. Similarly, for any two formal concepts there is always a lowest node (the supremum of the two), that can be reached from both concepts via ascending paths. And any common superconcept of the two is on an ascending path from their supremum.

2.2.3 Complete lattices

The operations for computing with formal concepts, infimum and supremum, are not as weird as one might suspect. In fact, we obtain with each concept lattice an algebraic structure called a "lattice", and such structures occur frequently in mathematics and computer science. "Lattice theory" is an active field of research in mathematics. A *lattice* is an algebraic structure with two operations (called "meet" and "join" or "infimum" and "supremum") that satisfy certain natural conditions:[3]

Definition 10 A partially ordered set $\mathbb{V} := (V, \leq)$ is a **lattice**, if there exists, for every pair of elements $x, y \in V$, their infimum $x \wedge y$ and their supremum $x \vee y$. $\Diamond$

We shall not discuss the algebraic theory of lattices in this lecture. Many universities offer courses in lattice theory, and there are excellent textbooks.[4]

Concept lattices have an additional nice property: they are **complete lattice**s. This means that the operations of infimum and supremum do not only work for an input consisting of two elements, but for arbitrary many. In other words: each collection of formal concepts has a greatest common subconcept and a least common superconcept. This is even true for infinite sets of concepts. The operations "infimum" and "supremum" are not necessarily binary, they work for any input size.

Definition 11 A partially ordered set $\mathbb{V} := (V, \leq)$ is a **complete lattice**, if for every set $A \subseteq V$ exist its infimum $\bigwedge V$ and its supremum $\bigvee A$. $\Diamond$

The definition requests the existence of infimum and supremum for every set A, hence also for the empty set $A := \emptyset$. Following the definition, we obtain that $\bigwedge \emptyset$ has to be the (unique) largest element of the lattice. It is denoted by $\mathbf{1_V}$. Dually, $\bigvee \emptyset$ has to be the smallest element of the lattice; it is denoted by $\mathbf{0_V}$.

The arbitrary arity of infimum and supremum is very useful, but will make essentially no difference for our considerations, because we shall mainly be concerned with finite formal contexts and finite concept lattices. Well, this is not completely true. In fact, although the concept lattice in Figure 2.1 is finite, its ten objects are representatives for *all* possibilities to combine two unit squares. Of course, there are infinitely many such possibilities. It is true that we shall consider finite concept lattices, but our examples may be taken from an infinite reservoir.

2.2.4 The Basic Theorem

We give now a mathematically precise formulation of the algebraic properties of concept lattices. The theorem below is not difficult, but basic for many other results. Its formulation contains some technical terms that we have not mentioned so far.

To understand the notion of a "supremum-dense" set, think of the positive integers with the least common multiple (LCM) in the rôle of the supremum operation. Some numbers can be written as a LCM of other integers. For example,

[3]Unfortunately, the word "lattice" is used with different meanings in mathematics. It also refers to generalized grids.

[4]*An introduction to lattices and order* by B. Davey and H. Priestley is particularly popular among CS students.

$12 = \mathrm{LCM}\{3, 4\}$. Therefore, 12 is called *LCM-reducible.* Prime numbers cannot be written as an LCM of other numbers. Therefore, prime numbers are LCM-irreducible. But prime numbers are not the only LCM-irreducible numbers. For example, 25 is not the least common multiple of other numbers, and thus is LCM-irreducible. It is easy to see that the LCM-irreducible integers are precisely the *prime powers*[5]. The set of prime powers is *LCM-dense*, which means that every positive integer can be written as a LCM of prime powers. Any larger set, that is, any set containing all prime powers, is also LCM-dense, of course.

There are no GCD-irreducible numbers. Every positive integer can be written as the GCD of other numbers. But there are GCD-dense sets of numbers. For example, the set of all integers greater than 1000 is GCD-dense.

In a complete lattice, an element is called **supremum-irreducible** if it cannot be written as a supremum of other elements, and **infimum-irreducible** if it can not be expressed as an infimum of other elements. It is very easy to locate the irreducible elements in a diagram of a finite lattice: the supremum-irreducible elements are precisely those from which there is exactly one edge going downward. An element is infimum-irreducible if and only if it is the start of exactly one upward edge. In Figure 2.1 there are precisely nine supremum-irreducible concepts and precisely five infimum-irreducible concepts. Exactly four concepts have both properties, they are **doubly irreducible**.

A set of elements of a complete lattice is called **supremum-dense**, if every lattice element is a supremum of elements from this set. Dually, a set is called **infimum-dense**, if the infima that can be computed from this set exhaust all lattice elements.

Definition 12 Two complete lattices $\mathbb{V}$ and $\mathbb{W}$ are **isomorphic** ($\mathbb{V} \cong \mathbb{W}$), if there exists a bijective mapping $\varphi\colon V \to W$ with $x \leq y \iff \varphi(x) \leq \varphi(y)$. The mapping φ is then called **lattice isomorphism** between $\mathbb{V}$ and $\mathbb{W}$. $\Diamond$

Example 3 The lattice $(\{1, 2, 3, 5, 6, 10, 15, 30\}, |)$ is isomorphic to the powerset $(\mathfrak{P}(\{2, 3, 5\}), \subseteq)$ with the isomorphism $\varphi\colon \{1, 2, 3, 5, 6, 10, 15, 30\} \to \mathfrak{P}(\{2, 3, 5\})$ with $\varphi(x) := \{y \in \{2, 3, 5\} \mid y \text{ divides } x\}$.

Now we have defined all the terminology necessary for stating the main theorem of Formal Concept Analysis.

Theorem 4 (The basic theorem of Formal Concept Analysis.) *The concept lattice of any formal context (G, M, I) is a complete lattice. For an arbitrary set $\{(A_i, B_i) \mid i \in I\} \subseteq \mathfrak{B}(G, M, I)$ of formal concepts, the supremum is given by*

$$\bigvee_{i\in I}(A_i, B_i) = \left((\bigcup_{i\in I} A_i)'', \bigcap_{i\in I} B_i\right)$$

and the infimum is given by

$$\bigwedge_{i\in I}(A_i, B_i) = \left(\bigcap_{i\in I} A_i, (\bigcup_{i\in I} B_i)''\right).$$

A complete lattice L is isomorphic to $\underline{\mathfrak{B}}(G, M, I)$ iff there are mappings $\tilde{\gamma} : G \to L$ and $\tilde{\mu} : M \to L$ such that $\tilde{\gamma}(G)$ is supremum-dense and $\tilde{\mu}(M)$ is infimum-dense in L, and

$$g\ I\ m \iff \tilde{\gamma}(g) \leq \tilde{\mu}(m).$$

In particular, $L \cong \underline{\mathfrak{B}}(L, L, \leq)$.

[5]The number 1 requires special treatment. We shall not go into this.

The theorem is less complicated as it first may seem. We give some explanations below. Readers in a hurry may skip these and continue with the next section.

The first part of the theorem gives the precise formulation for infimum and supremum of arbitrary sets of formal concepts. The second part of the theorem gives (among other information) an answer to the question if concept lattices have any special properties. The answer is "no": every complete lattice is (isomorphic to) a concept lattice. This means that for every complete lattice we must be able to find a set G of objects, a set M of attributes and a suitable relation I, such that the given lattice is isomorphic to $\underline{\mathfrak{B}}(G, M, I)$. The theorem does not only say how this can be done, it describes in fact *all* possibilities to achieve this.

In Figure 2.1, every object is attached to a unique concept, the corresponding *object concept*. Similarly for each attribute there corresponds an *attribute concept*. These can be defined as follows:

Definition 13 Let (G, M, I) be some formal context. Then

- for each object $g \in G$ the corresponding **object concept** is

$$\gamma(g) := (\{g\}'', \{g\}'),$$

- and for each attribute $m \in M$ the **attribute concept** is given by

$$\mu(m) := (\{m\}', \{m\}'').$$

The set of all object concepts of (G, M, I) is denoted $\gamma(G)$, the set of all attribute concepts is $\mu(M)$. $\Diamond$

Using Definition 6 and Proposition 1 it is easy to check that these expressions in fact define formal concepts of (G, M, I).

We have that $\gamma(g) \leq (A, B) \iff g \in A$. A look at the first part of the Basic Theorem shows that each formal concept is the supremum of all the object concepts below it (Exercise 5). Therefore, the set $\gamma(G)$ of all object concepts is supremum-dense. Dually, the attribute concepts form an infimum dense set in $\underline{\mathfrak{B}}(G, M, I)$. The Basic Theorem says that, conversely, any supremum-dense set in a complete lattice L can be taken as the set of objects and any infimum-dense set be taken as a set of attributes for a formal context with concept lattice isomorphic to L.

We conclude with a simple observation that often helps to find errors in concept lattice diagrams. The fact that the object concepts form an supremum-dense set implies that every supremum-irreducible concept must be an object concept (the converse is not true). Dually, every infimum-irreducible concept must be an attribute concept. This yields the following rule for concept lattice diagrams:

Proposition 5 *Given a formal context (G, M, I) and a finite order diagram, labeled by the objects from G and the attributes from M. For $g \in G$ let $\tilde{\gamma}(g)$ denote the element of the diagram that is labeled with g, and let $\tilde{\mu}(m)$ denote the element labeled with m. Then the given diagram is a correctly labeled diagram of $\underline{\mathfrak{B}}(G, M, I)$ if and only if it fulfills the following conditions:*

1. *The diagram is a correct lattice diagram,*
2. *every supremum-irreducible element is labeled by some object,*
3. *every infimum-irreducible element is labeled by some attribute,*
4. $g\, I\, m \iff \tilde{\gamma}(g) \leq \tilde{\mu}(m)$.

2.2.5 The duality principle

The definitions of lattices and complete lattices are self-dual: If $(V, \leq)$ is a (complete) lattice, then $(V, \leq)^d := (V, \geq)$ is also a (complete) lattice. If a theorem holds for a (complete) lattice, then the 'dual theorem' also holds, i. e., the theorem where all occurrences of $\leq, \vee, \wedge, \bigvee, \bigwedge, \mathbf{0_V}, \mathbf{1_V}$ etc. are replaced by $\geq, \wedge, \vee, \bigwedge, \bigvee, \mathbf{1_V}, \mathbf{0_V}$, resp.

For concept lattices, their dual can be obtained by "permuting" the formal context:

Lemma 6 *Let (G, M, I) be a formal context. Then $(\underline{\mathfrak{B}}((G, M, I)))^d \cong \underline{\mathfrak{B}}(M, G, I^{-1})$, where $I^{-1} := \{(m, g) \in M \times G \mid (g, m) \in I\}$.*

2.3 Computing and drawing concept lattices

2.3.1 Computing all concepts

There are several algorithms that help drawing concept lattices. We shall discuss some of them below. But we find it instructive to start by some small examples that can be drawn by hand. For computing concept lattices, we will investigate a fast algorithm later. We start with a naive method, and proceed then to a method which is suitable for manual computation.

In principle, it is not difficult to find all the concepts of a formal context. The following proposition summarizes the naive possibilities of generating all concepts:

Lemma 7 *Each concept of a context (G, M, I) has the form (X'', X') for some subset $X \subseteq G$ and the form (Y', Y'') for some subset $Y \subseteq M$. Conversely, all such pairs are concepts.*

Every extent is the intersection of attribute extents and every intent is the intersection of object intents.

The first part of the lemma suggests the following, first algorithm for computing all concepts:

1. $\underline{\mathfrak{B}}(G, M, I) := \emptyset$;
2. For all subsets X of G, add (X'', X') to $\underline{\mathfrak{B}}(G, M, I)$;

However, this is rather inefficient, and not practicable even for relatively small contexts. The second part of the proposition at least yields the possibility to calculate the concepts of a small context by hand.

The following method is more efficient, and is recommended for computations by hand. It is based on the following observations:

1. It suffices to determine all concept extents [or all concept intents] of (G, M, I),

   since we can always determine the other part of a formal concept with the help of the derivation operators.

2. The intersection of arbitrary many extents is an extent [and the intersection of arbitrary intents is an intent].

   This follows easily from the formulae given in the Basic Theorem. By the way: a convention that may seem absurd on the first glance allows to include in "arbitrary many" also the case "zero". The convention says that the intersection of zero intents equals M and the intersection of zero extents equals G.

3. One can determine all concept extents from knowing all **attribute extent**s $\{m\}'$, $m \in M$ [and all concept intents from all **object intent**s $\{g\}'$, $g \in G$] because
   - every extent is an intersection of attribute extents,
   - [and every intent is the intersection of object intents].

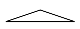
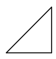
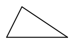
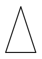
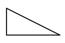
   This is a consequence of the fact that the attribute concepts are infimum-dense and the object concepts are supremum-dense.

This leads to the following

Instruction how to determine all formal concepts of a small formal context

1. Initialize a list of concept extents. To begin with, write for each attribute $m \in M$ the attribute extent $\{m\}'$ to this list (if not already present).
2. For any two sets in this list, compute their intersection. If the result is a set that is not yet in the list, then extend the list by this set. With the extended list, continue to build all pairwise intersections.
3. If for any two sets of the list their intersection is also in the list, then extend the list by the set G (provided it is not yet contained in the list). The list then contains all concept extents (and nothing else).
4. For every concept extent A in the list compute the corresponding intent A' to obtain a list of all formal concepts (A, A') of (G, M, I).

An example We illustrate the method by means of an example from elementary geometry. The objects of our example are seven triangles. The attributes are five standard properties that triangles may or may not have:

Triangles				
abbreviation	coordinates			diagram
T1	(0,0)	(6,0)	(3,1)	
T2	(0,0)	(1,0)	(1,1)	
T3	(0,0)	(4,0)	(1,2)	
T4	(0,0)	(2,0)	$(1,\sqrt{3})$	
T5	(0,0)	(2,0)	(5,1)	
T6	(0,0)	(2,0)	(1,3)	
T7	(0,0)	(2,0)	(0,1)	

Attributes	
symbol	property
a	equilateral
b	isosceles
c	acute angled
d	obtuse angled
e	right angled

We obtain the following formal context

	a	b	c	d	e
$T1$		×		×	
$T2$		×			×
$T3$			×		
$T4$	×	×	×		
$T5$				×	
$T6$		×	×		
$T7$					×

Following the above instruction, we proceed:

1. Write the attribute extents to a list.

No.		extent	found as
e_1	:=	$\{T_4\}$	$\{a\}'$
e_2	:=	$\{T_1, T_2, T_4, T_6\}$	$\{b\}'$
e_3	:=	$\{T_3, T_4, T_6\}$	$\{c\}'$
e_4	:=	$\{T_1, T_5\}$	$\{d\}'$
e_5	:=	$\{T_2, T_7\}$	$\{e\}'$

2. Compute all pairwise intersections, and

3. add G

No.		extent	found as
e_1	:=	$\{T_4\}$	$\{a\}'$
e_2	:=	$\{T_1, T_2, T_4, T_6\}$	$\{b\}'$
e_3	:=	$\{T_3, T_4, T_6\}$	$\{c\}'$
e_4	:=	$\{T_1, T_5\}$	$\{d\}'$
e_5	:=	$\{T_2, T_7\}$	$\{e\}'$
e_6	:=	$\emptyset$	$e_1 \cap e_4$
e_7	:=	$\{T_4, T_6\}$	$e_2 \cap e_3$
e_8	:=	$\{T_1\}$	$e_2 \cap e_4$
e_9	:=	$\{T_2\}$	$e_2 \cap e_5$
e_{10}	:=	$\{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$	step 3

4. Compute the intents

Concept No.	(extent	,	intent)
1	$(\{T_4\}$	,	$\{a, b, c\})$
2	$(\{T_1, T_2, T_4, T_6\}$	,	$\{b\})$
3	$(\{T_3, T_4, T_6\}$	,	$\{c\})$
4	$(\{T_1, T_5\}$	,	$\{d\})$
5	$(\{T_2, T_7\}$	,	$\{e\})$
6	$(\emptyset$	,	$\{a, b, c, d, e\})$
7	$(\{T_4, T_6\}$	,	$\{b, c\})$
8	$(\{T_1\}$	,	$\{b, d\})$
9	$(\{T_2\}$	,	$\{b, e\})$
10	$(\{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$	,	$\emptyset)$

We have now computed all ten formal concepts of the triangles–context. The last step can be skipped if we are not interested in an explicit list of all concepts, but just in computing a line diagram.

2.3.2 Drawing concept lattices

Based on one of the lists 3. or 4., we can start to draw a diagram. Before doing so, we give two simple definitions.

Definition 14 Let (A_1, B_1) and (A_2, B_2) be formal concepts of (G, M, I).

We say that (A_1, B_1) is a **proper subconcept** of (A_2, B_2), if $(A_1, B_1) \leq (A_2, B_2)$ and, in addition, $(A_1, B_1) \neq (A_2, B_2)$ holds. As an abbreviation, we write $(A_1, B_1) < (A_2, B_2)$.

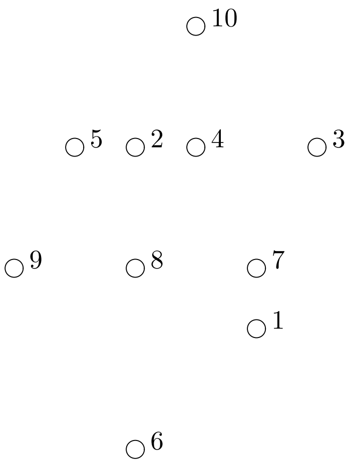
We say that (A_1, B_1) is a **lower neighbour** of (A_2, B_2), if $(A_1, B_1) < (A_2, B_2)$, but no formal concept (A, B) of (G, M, I) exists with $(A_1, B_1) < (A, B) < (A_2, B_2)$. The abbreviation for this is $(A_1, B_1) \prec (A_2, B_2)$. $\Diamond$

Instruction how to draw a line diagram of a small concept lattice

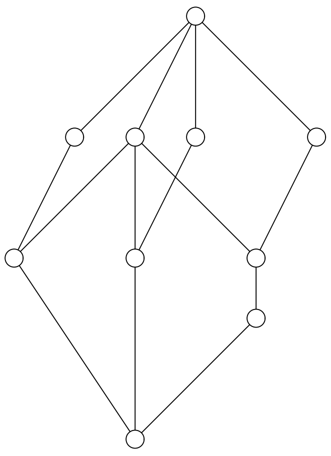
5. Take a sheet of paper and draw a small circle for every formal concept, in the following manner: a circle for a concept is always positioned higher than the all circles for its proper subconcepts.
6. Connect each circle with the circles of its lower neighbors.
7. Label with attribute names: attach the attribute m to the circle representing the concept $(\{m\}', \{m\}'')$.
8. Label with object names: attach each object g to the circle representing the concept $(\{g\}'', \{g\}')$.

We follow these instructions.

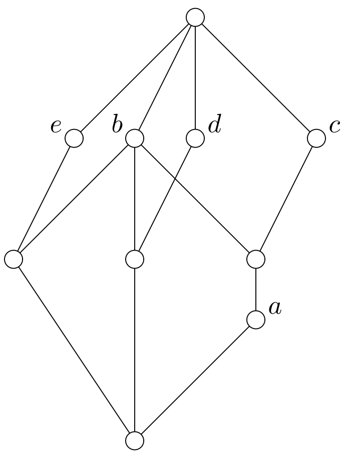
5. Draw a circle for each of the formal concepts:



6. Connect circles with their lower neighbours:



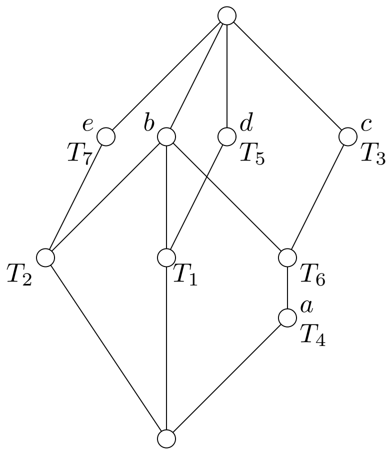
7. Add the attribute names:



8. Determine the object concepts

Object g	object intent $\{g\}'$	No. of concept
T_1	$\{b, d\}$	8
T_2	$\{b, e\}$	9
T_3	$\{c\}$	3
T_4	$\{a, b, c\}$	1
T_5	$\{d\}$	4
T_6	$\{b, c\}$	7
T_7	$\{e\}$	5

and add the object names to the diagram:



Ready! Usually it takes some tries before a nice, readable diagram is achieved. Finally we can make the effort to avoid abbreviations and to increase the readability. The result is shown in Figure 2.2.

2.3.3 Clarifying and reducing a formal context

There are context manipulations that simplify a formal context without changing the diagram, except for the labellig. It is usually advisable to do these manipulations first, before starting computations.

The simplest operation is **clarification**, which refers to indentifying "equal rows" of a formal context, and "equal columns" as well. What is meant is that

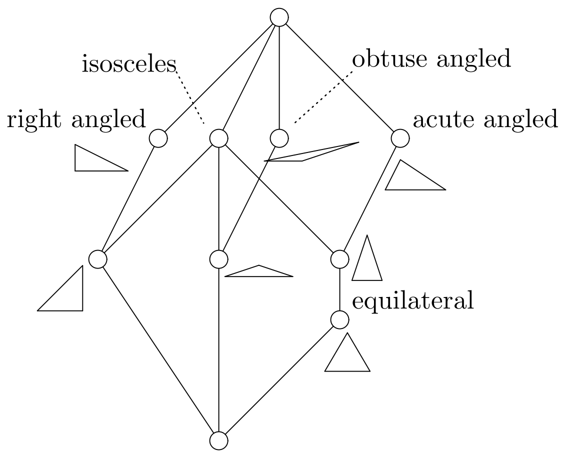


Figure 2.2: A diagram of the concept lattice of the triangle context.

if a context contains objects $g_1, g_2, \ldots$ with $\{g_i\}' = \{g_j\}'$ for all i, j, that is, objects which have exactly the same attributes, then these can be replaced by a single object, the name of which is just the list of names of these objects. The same can be done for attributes with identical attribute extent.

Definition 15 We say that a formal context is **clarified** if no two of its object intents are equal and no two of its attribute extents are equal. $\diamondsuit$

A stronger operation is **reduction**, which refers to omitting attributes that are equivalent to combinations of other attributes (and dually for objects). For defining reduction it is convenient to work with a clarified context.

Definition 16 An attribute m of a clarified context is called **reducible** if there is a set $S \subseteq M$ of attributes with $\{m\}' = S'$, otherwise it is **irreducible**. Reduced objects are defined dually. A formal context is called **reduced**, if all objects and all attributes are irreducible. $\diamondsuit$

$\{m\}' = S'$ means that an object g has the attribute m if and only if it has all the attributes from S. If we delete the column m from our cross table, no essential information is lost because we can reconstruct this column from the data contained in other columns (those of S). Moreover, deleting that column does not change the number of concepts, nor the concept hierarchy, because $\{m\}' = S'$ implies that m is in the intent of a concept if and only if S is contained in that intent. The same is true for reducible objects and concept extents. Deleting a reducible object from a formal context does not change the structure of the concept lattice.

It is even possible to remove several reducible objects and attributes simultaneously from a formal context without any effect on the lattice structure, as long as the number of removed elements is finite.

Definition 17 Let (G, M, I) be a finite context, and let G_{irr} be the set of irreducible objects and M_{irr} be the set of irreducible attributes of (G, M, I). The context $(G_{irr}, M_{irr}, I \cap G_{irr} \times M_{irr})$ is the **reduced context** corresponding to (G, M, I).

For a finite lattice L let $J(L)$ denote the set of its supremum-irreducible elements and let $M(L)$ denote the set of its infimum-irreducible elements. Then $(J(L), M(L), \leq)$ is the **standard context** for the lattice L. $\diamondsuit$

Proposition 8 *A finite context and its reduced context have isomorphic concept lattices. For every finite lattice L there is (up to isomorphism) exactly one reduced context, the concept lattice of which is isomorphic to L, namely its standard context.*

2.3.4 Other algorithms for computing concept lattices

The algorithm which we shall use in what follows to generate all formal concepts of a given formal context can be formulated without referring to concept lattices. It will be an *algorithm to generate all closed sets of a closure system*, called NEXTCLOSURE. It can be applied to concept lattices because for each formal context (G, M, I), the set $\mathbf{Ext}(G, M, I)$ of all concept extents is a closure system, as well as the set $\mathbf{Int}(G, M, I)$ of all the concept intents of (G, M, I).

———(to be written)———

- Titanic
- Nourine
- ...

2.3.5 Computer programs for concept generation and lattice drawing

There are several free or conditionally free programs available for concept lattice generation and/or drawing. A well known one, old fashioned but reliable, is CONIMP (for DOS), available under `www.mathematik.tu-darmstadt.de`. S. Yevtushenko provides the Concept Explorer, which allows for editing formal contexts and computing their concept lattices. It can be downloaded from `http://sourceforge.net/projects/conexp/`. A rather advanced package, invented for applications in data analysis, is the NAVICON DECISION SUITE by NAVICON GmbH. A demo version is available under `www.navicon.de`.

2.4 Additive and Nested Line Diagrams

In this section, we discuss possibilities to generate line diagrams both automatically or by hand. A list of some dozens of concepts may already be quite difficult to survey, and it requires practice to draw good line diagrams of concept lattices with more than 20 elements.

The best and most versatile form of representation for a concept lattice is a well drawn line diagram. It is however tedious to draw such a diagram by hand and one would wish an automatic generation by means of a computer. We know quite a few algorithms to do this, but none which provides a general satisfactory solution. It is by no means clear which qualities make up a *good* diagram. It should be transparent, easily readable and should facilitate the interpretation of the data represented. How this can be achieved in each individual case depends however on the aim of the interpretation and on the structure of the lattice. Simple optimization criteria (minimization of the number of edge crossings, drawing in layers, etc.) often bring about results that are unsatisfactory. Nevertheless, automatically generated diagrams are a great help: they can serve as the starting point for drawing by hand. Therefore, we will describe simple methods of generating and manipulating line diagrams by means of a computer.

2.4.1 Additive line diagrams

We will now explain a method where a computer generates a diagram and offers the possibility of improving it interactively. Programming details are irrelevant in this context. We will therefore only give a **positioning rule** which assigns points in the plane to the elements of a given ordered set $(P, \leq)$. If a and b are elements of P with $a < b$, the point assigned to a must be lower than the point assigned to b (i.e., it must have a smaller y-coordinate). This is guaranteed by our method. We will leave the computation of the edges and the checking for undesired coincidences of vertices and edges to the program. We do not even guarantee that our positioning is injective (which of course is necessary for a correct line diagram). This must also be checked if necessary.

Definition 18 A **set representation** of an ordered set $(P, \leq)$ is an order embedding of $(P, \leq)$ in the power-set of a set X, i.e., a map

$$\operatorname{rep} : P \to \mathfrak{P}(X)$$

with the property

$$x \leq y \iff \operatorname{rep} x \subseteq \operatorname{rep} y.$$

$\Diamond$

An example of a set representation for an arbitrary ordered set $(P, \leq)$ is the assignment

$$X := P, \quad a \mapsto (a].$$

In the case of a concept lattice,

$$X := G, \quad (A, B) \mapsto A$$

is a set representation.

$$X := M, \quad (A, B) \mapsto M \setminus B$$

is another set representation, and both can be combined to

$$X := G \dot\cup M, \quad (A, B) \mapsto A \cup (M \setminus B).$$

It is sufficient to limit oneself to the irreducible objects and attributes.

For an **additive line diagram** of an ordered set $(P, \leq)$ we need a set representation $\operatorname{rep} : P \to \mathfrak{P}(X)$ as well as a **grid projection**

$$\operatorname{vec} : X \to \mathbb{R}^2,$$

assigning a real vector with a positive y-coordinate to each element of X. By

$$\operatorname{pos} p := n + \sum_{x \in \operatorname{rep} p} \operatorname{vec} x$$

we obtain positioning of the elements of P in the plane. Here, n is a vector which can be chosen arbitrarily in order to shift the entire diagram. By only allowing positive y–coordinates for the grid projection we make sure that no element p is positioned below an element q with $q < p$.

Every finite line diagram can be interpreted as an additive diagram with respect to an appropriate set representation. For concept lattices we usually use the representation by means of the irreducible objects and/or attributes. The resulting diagrams are characterized by a great number of parallel edges, which improves their readability. Experience shows that the set representation by means of the irreducible attributes is most likely to result in an easily interpretable diagram. Figure 2.2 for instance was obtained by selecting the irreducible attributes for the set representation.

Since the second set representation given above is somehow unnatural, we introduce for this purpose the dual set representation.

Definition 19 A **dual set representation** of an ordered set $(P, \leq)$ is an order–inversing embedding of $(P, \leq)$ in the power-set of a set X, i.e., a map

$$\mathrm{rep}' : P \to \mathfrak{P}(X)$$

with the property

$$x \leq y \iff \mathrm{rep}'x \supseteq \mathrm{rep}'y.$$

◊

Now

$$X := M, \quad \mathrm{rep}' \colon (A, B) \mapsto M \setminus B$$

is a dual set representation We request now that the grid projection allows only negative y–coordinates. The following shows that the two ways are indeed equivalent: Let $\mathrm{vec}' \colon X \to \mathbb{R}^2$ be given by $\mathrm{vec}'(m) := (-x, -y)$ where $\mathrm{vec}(m) = (x, y)$ for all $m \in X$. Then all y–coordinates are indeed negative. We obtain then the following equality:

$$\begin{aligned}
\mathrm{pos}(A, B) &= n + \sum_{m \in M \setminus A} \mathrm{vec}(m) \\
&= n + \sum_{m \in M} \mathrm{vec}(m) + \sum_{m \in A} - \mathrm{vec}(m) \\
&= n' + \sum_{m \in A} \mathrm{vec}'(m) \\
&= \mathrm{pos}'(A, B)
\end{aligned}$$

where $n' := n + \sum_{m \in M} \mathrm{vec}(m)$.

It is particularly easy to manipulate these diagrams: If we change – the set representation being fixed – the grid projection for an element $x \in X$, this means that all images of the order filter $\{p \in P \mid x \in \mathrm{rep}\, p\}$ are shifted by the same distance and that all other points remain in the same position. In the case of the set representation by means of the irreducibles these order filters are precisely principal filters or complements of principal ideals, respectively. This means that we can manipulate the diagram by shifting principal filters or principal ideals, respectively, and leaving all other elements in position.

Even carefully constructed line diagrams loose their readability from a certain size up, as a rule from around 50 elements up. One gets considerably further with *nested line diagrams* which will be introduced next. However, these diagrams do not only serve to represent larger concept lattices. They offer the possibility to visualize how the concept lattice changes if we add further attributes.

2.4.2 Nested line diagrams

Nested line diagrams permit a satisfactory graphical representation of somewhat larger concept lattices. The basic idea of the nested line diagram consists of clustering parts of an ordinary diagram and replacing bundles of parallel lines between these parts by one line each. Thus, a nested line diagram consists of ovals, which contain clusters of the ordinary line diagram and which are connected by lines.

In the simplest case two ovals which are connected by a simple line are congruent. Here, the line indicates that corresponding circles within the ovals are direct neighbors, resp.

Furthermore, we allow that two ovals connected by a single line do not necessarily have to be congruent, but they may each contain a part of two congruent figures. In this case, the two congruent figures are drawn in the ovals as a "background structure", and the elements are drawn as bold circles if they are part of the respective substructures. The line connecting the two boxes then indicates that the respective pairs of elements of the background shall be connected with each other. An example is given in Figure 2.3. It is a screenshot of a library information system which was set up for the library of the Center on Interdisciplinary Technology Research of Darmstadt University of Technology.

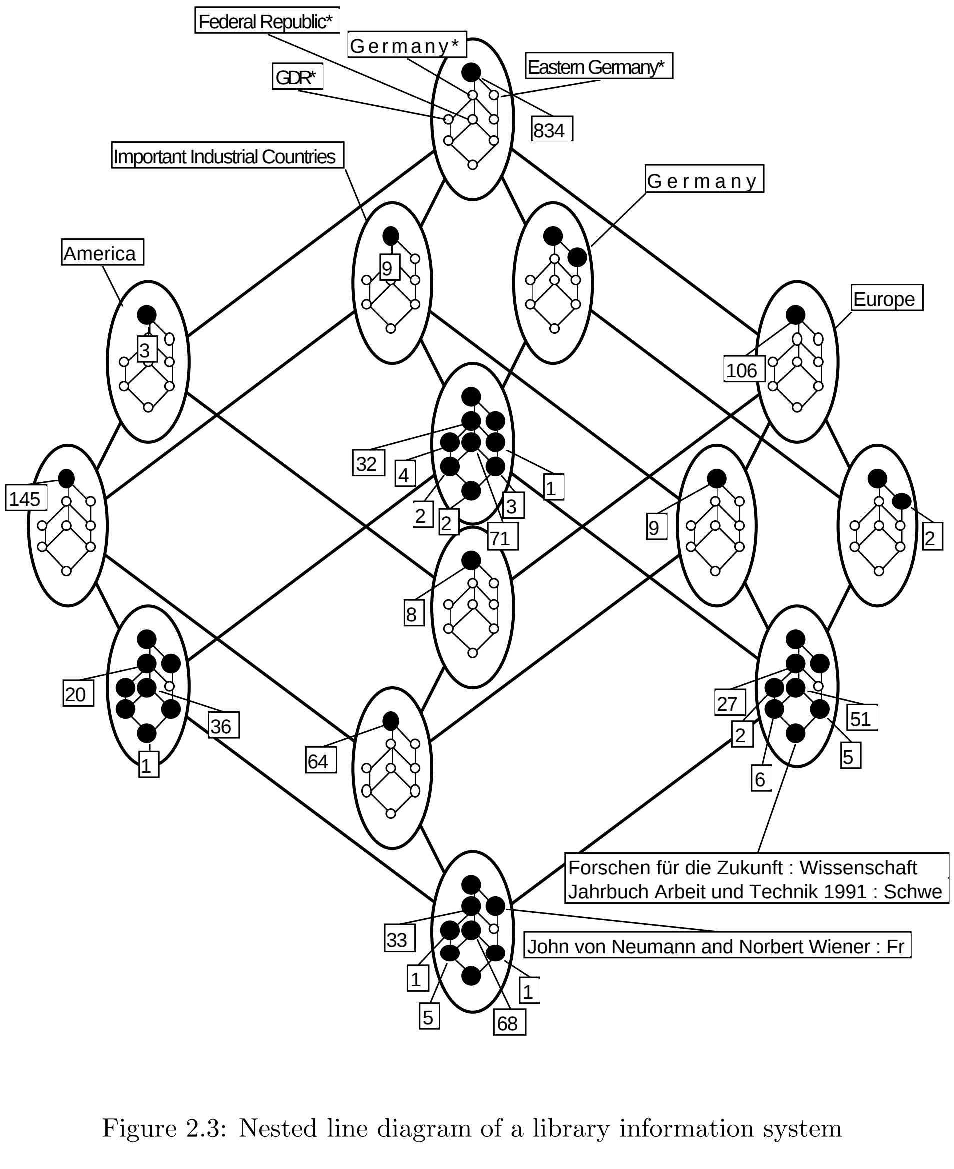

Federal Republic*
Germany*
GDR*
Eastern Germany*
834
Important Industrial Countries
Germany
America
9
Europe
3
106
145
32
4
1
2
2
3
71
9
2
8
20
36
1
27
51
2
5
6
64
Forschen für die Zukunft : Wissenschaft
Jahrbuch Arbeit und Technik 1991 : Schwe
John von Neumann and Norbert Wiener : Fr
33
1
1
5
68

Figure 2.3: Nested line diagram of a library information system

Nested line diagrams originate from partitions of the set of attributes. The basis is the following theorem:

Theorem 9 *Let (G, M, I) be a context and $M = M_1 \cup M_2$. The map*

$$(A, B) \mapsto (((B \cap M_1)', B \cap M_1), ((B \cap M_2)', B \cap M_2))$$

is a $\bigvee$-preserving order embedding of $\mathfrak{B}(G, M, I)$ in the direct product of $\underline{\mathfrak{B}}(G, M_1, I \cap G \times M_1)$ and $\underline{\mathfrak{B}}(G, M_2, I \cap G \times M_2)$. The component maps

$$(A, B) \mapsto ((B \cap M_i)', B \cap M_i)$$

are surjective on $\underline{\mathfrak{B}}(G, M_i, I \cap G \times M_i)$.

Proof If (A, B) is a concept of (G, M, I), then $B \cap M_i$ is the set of all attributes common to the objects of A in the context $(G, M_i, I \cap G \times M_i)$, i.e., it is an intent of this context. Hence the above-mentioned assignment is really a map into the product. The union of the intents $B \cap M_1$ and $B \cap M_2$ again yields B, i.e., the map is injective. The fact that it is furthermore $\bigvee$-preserving (and thus an order-embedding) can again be seen from the concept intents. It remains to be shown that the component maps are surjective. Let C be an intent of $(G, M_i, I \cap G \times M_i)$. Then $B := C^{II}$ is an intent of (G, M, I) with $B \cap M_i = C$, i.e., the image of the concept (B', B) of (G, M, I) under the ith component map is the concept with the intent C. $\square$

In order to sketch a nested line diagram, we proceed as follows: First of all we split up the attribute set: $M = M_1 \cup M_2$. This splitting up does not have to be disjoint. More important for interpretation purposes is the idea that the sets M_i bear meaning. Now, we draw line diagrams of the subcontexts $\mathbb{K}_i := (G, M_i, I \cap G \times M_i)$, $i \in \{1, 2\}$ and label them with the names of the objects and attributes, as usual. Then we sketch a nested diagram of the product of the concept lattices $\underline{\mathfrak{B}}(\mathbb{K}_i)$ as an auxiliary structure. For this purpose we draw a large copy of the diagram of $\underline{\mathfrak{B}}(\mathbb{K}_1)$, representing the lattice elements not by small circles but by congruent ovals, which contain each a diagram of $\underline{\mathfrak{B}}(\mathbb{K}_2)$.

By Theorem 9 the concept lattice $\underline{\mathfrak{B}}(G, M, I)$ is embedded in this product as a $\bigvee$-semilattice. If a list of the elements of $\underline{\mathfrak{B}}(G, M, I)$ is available, we can enter them into the product according to their intents. If not, we enter the object concepts the intents of which can be read off directly from the context, and form all suprema.

This at the same time provides us with a further, quite practicable method of determining a concept lattice by hand: split up the attribute set as appropriate, determine the (small) concept lattices of the subcontexts, draw their product in form of a nested line diagram, enter the object concepts and close it against suprema. This method is particularly advisable in order to arrive at a useful diagram quickly.

2.5 Examples

———(to be written)———

2.6 Exercises

1. Give the formal context for the concept lattice shown in Figure 2.1.
2. Give a proof of Proposition 1.
3. Let (G, M, I) be a formal context.
   (a) Show that, for all $B_1, B_2 \subseteq M$, $(B_1 \cup B_2)' = B_1' \cap B_2'$ holds.
   (b) Give a counter-example for $(B_1 \cap B_2)' \neq B_1' \cup B_2'$.

4. Let (A_1, B_1) and (A_2, B_2) be formal concepts of a formal context (G, M, I). Show that the expression 'greatest common subconcept' in Lemma 3 for $(A_1 \cap A_2, (B_1 \cup B_2)'')$ is justified by showing
   (a) $(A_1 \cap A_2, (B_1 \cup B_2)'')$ is a formal concept (of the same context).
   (b) $(A_1 \cap A_2, (B_1 \cup B_2)'')$ is a subconcept of both (A_1, B_1) and (A_2, B_2).
   (c) Any other common subconcept of (A_1, B_1) and (A_2, B_2) is also a subconcept of $(A_1 \cap A_2, (B_1 \cup B_2)'')$.

5. Show that, in a concept lattice, the set of all object concepts is supremum-dense. Hint: Show that each formal concept is the supremum of the set of all its subconcepts which are also object concepts.

6. Let $(V, \leq)$ be a partially ordered set where every subset A of V has an infimum. Show that $(V, \leq)$ is a complete lattice.

7. Show that every non-empty finite lattice is a complete lattice.

8. Show that in every lattice the following equivalences hold:

$$x \leq y \iff x \wedge y = x \iff x \vee y = y$$

   Remark: From the knowledge about infima and suprema of a lattice, we are thus able to reconstruct its order relation.

9. Prove Lemma 6: For any formal context (G, M, I), the concept lattices $(\underline{\mathfrak{B}}((G, M, I)))^d$ and $\underline{\mathfrak{B}}(M, G, I^{-1})$ are isomorphic.

10. Let $\mathbb{V}$ and $\mathbb{V}$ be two complete lattices, and $\varphi \colon V \to W$ be a bijective mapping. Show that φ is an isomorphism of complete lattices if and only if, for all subsets X of V,

$$\varphi\left(\bigvee X\right) = \bigvee \{\varphi(x) \mid x \in X\}$$

   and

$$\varphi\left(\bigwedge X\right) = \bigwedge \{\varphi(x) \mid x \in X\} \quad .$$

11. (a) Draw the lattice $\underline{L} := (L, |)$ with $L := \{2^i 3^j 5^k \mid 0 \leq i, j \leq 1; 0 \leq k \leq 2\}$.
   (b) Which are the supremum-irreducible elements?
   (c) Which are the infimum-irreducible elements?
   (d) Give a formal context (G, M, I) such that its concept lattice is isomorphic to $\underline{L}$. Give the isomorphism explicitly.
   (e) How could the fact that $\underline{L}$ and $\underline{\mathfrak{B}}(G, M, I)$ are isomorphic be shown with the Main Theorem?

12. (a) Sketch the lattice $(\mathbb{N}, |)$.
   (b) Is $(\mathbb{N}, |)$ a complete lattice? If not, how can it be 'repaired'?

13.

	small	*medium*	*large*	*near*	*far*	*moon*	*inhibited*
Earth	×			×		×	×
Jupiter		×		×	×	×	
Mars	×			×		×	×
Mercury	×			×			

   (a) Clarify the formal context.
   (b) Reduce the clarified context.

(c) Draw a line diagram of the reduced context.

(d) Draw a line diagram of the original context.

14.

	old	young	male	female
Father	×		×	
Mother	×			×
Sun		×	×	
Daughter		×		×

Draw the concept lattice of the family context as additive line diagram

(a) with the set representation

$$X := G, \quad (A, B) \mapsto A$$

and the grid projection

Sun $\mapsto (-6, 4)$, Daughter $\mapsto (-1, 4)$, Father $\mapsto (1, 4)$, Mother $\mapsto (6, 4)$.

(b) with the dual set representation

$$X := M, \quad (A, B) \mapsto B$$

and the grid projection

young $\mapsto (-6, -4)$, male $\mapsto (-1, -4)$, female $\mapsto (1, -4)$, old $\mapsto (6, -4)$.

15.

	Father	Mother	Sun	Daughter
old	×	×		
young			×	×
male	×		×	
female		×		×

(a) Draw the concept lattice of the dual of the family context.

(b) State explicitly the isomorphism between the dual concept lattice of the family context (see previous exercise) and the concept lattice of the dual context.

16.

	< 60	≥ 60	1 term	2 terms	CDU	SPD	FDP
Heuss		×		×			×
Lübke		×		×	×		
Heinemann		×	×			×	
Scheel	×		×				×
Carstens	×		×		×		
Weizsäcker		×		×	×		

(a) Draw a nested line diagram of the concept lattice of the Bundespräsidenten context with $M_1 := \{< 60, \geq 60, 1 \text{ term}, 2 \text{ terms}\}$ and $M_2 := \{$ CDU, SPD, FDP $\}$.

(b) Draw a non-nested diagram of the concept lattice.

17.

	Latin America	Europe	Canada	Asia Pacific	Middle East	Africa	Mexico	Caribbean	United States
Air Canada	×	×	×	×	×		×	×	×
Air New Zealand		×		×					×
All Nippon Airways		×		×					×
Ansett Australia				×					
The Austrian Airlines Group		×	×	×	×	×			×
British Midland		×							
Lufthansa	×	×	×	×	×	×	×		×
Mexicana	×		×				×	×	×
Scandinavian Airlines	×	×		×		×			×
Singapore Airlines		×	×	×	×	×			×
Thai Airways International	×	×		×				×	×
United Airlines	×	×	×	×			×	×	×
VARIG	×	×		×		×	×		×

(a) Draw a nested line diagram of the concept lattice of the Star Alliance context with $M_1 := \{$ Europe, Asia Pacific, Africa, Middle East$\}$ and $M_2 := \{$ United States, Canada, Latin America, Mexico, Caribbean $\}$.

(b) Draw a non-nested diagram of the concept lattice.

2.7 Bibliographic Notes

———(to be written)———

Part II

Closure Systems and Implications

Chapter 3

Closure systems

The algorithms that will be the central theme of our course were developed for concept lattices, but can be rephrased without reference to Formal Concept Analysis. The reason is that the algorithm essentially relies on a single property of concept lattices, namely that the set of concept intents is closed under intersections. The technique can be formulated for arbitrary intersection closed families of sets, that is, for *closure systems*. Readers who are familiar with closure systems but not with Formal Concept Analysis may prefer this approach.

But note that this means no generalization. We will show that closure systems are not more general than systems of concept intents.

3.1 Definition and examples

Closure systems occur frequently in mathematics and computer science. Their definition is very simple, but not very intuitive when encountered for the first time. The reason is their higher level of abstraction: closure systems are *sets of sets* with certain properties.

Let us recall some elementary notions how to work with sets of sets. For clarity, we shall normally use small latin letters for elements, capital latin letters for sets and calligraphic letters for sets of sets. Given a (nonempty) set $\mathcal{S}$ of sets, we may ask

- which elements occur in these sets? The answer is given by the **union** of $\mathcal{S}$, denoted by
$$\bigcup \mathcal{S} := \{x \mid \exists_{S \in \mathcal{S}} \;\; x \in S\}.$$
- which elements occur in each of these sets? The answer is given by the **intersection** of $\mathcal{S}$, denoted
$$\bigcap \mathcal{S} := \{x \mid \forall_{S \in \mathcal{S}} \;\; x \in S\}.$$

Some confusion with this definition is caused by the fact that a set of sets may (of course) be empty. Applying the above definition to the case $\mathcal{S} := \emptyset$ is no problem for the union, since

$$\bigcup \emptyset = \{x \mid \exists_{S \in \mathcal{S}} x \in S\} = \{x \mid \text{false}\} = \emptyset \;\; .$$

But it gives a problem for the intersection, because then the condition $\forall_{S \in \mathcal{S}} \;\; x \in S$ is fulfilled by *all* x (because there is nothing to fulfill). But there is no *set of all* x; such sets are forbidden in set theory, because they would lead to contradictions.

For the case $\mathcal{S} = \emptyset$ the intersection is defined only with respect to some base set M. If we work with the subsets of some specified set M (as we often do, for example with the set of all attributes of some formal context), then we define

$$\bigcap \emptyset := M.$$

A set M with, say, n elements, has 2^n subsets. The set of all subsets of a set M is denoted $\mathfrak{P}(M)$ and is called the **power set** of the set M. To indicate that $\mathcal{S}$ is a set of subsets of M we may therefore simply write $\mathcal{S} \subseteq \mathfrak{P}(M)$.

3.1.1 Closure systems

A closure system on a set M is a set of subsets that contains M and is closed under intersections. More formally,

Definition 20 A **closure system** on a set M is a set $\mathcal{C} \subseteq \mathfrak{P}(M)$ satisfying

- $M \in \mathcal{C}$, and
- if $\mathcal{D} \subseteq \mathcal{C}$, then $\bigcap \mathcal{D} \in \mathcal{C}$.

$\Diamond$

Closure systems are everywhere:

- The subtrees of any tree form a closure system, because the intersection of subtrees in any case is a subtree.

  That this is true can be seen as follows: recall that in a tree any two vertices are connected by a (unique) path. A set of vertices induces a subtree if any only if it contains with any two of its vertices also all vertices on the path between these two. Now consider any selection of subtrees and let S be the set of vertices common to all these subtrees (i.e., their intersection). Let v, w be two vertices in S. The path between v and w belongs to every subtree containing v and w and therefore to each of the selected subtree. It is therefore also contained in their intersection. Thus, D is the vertex set of a subtree.

- Take any algebraic structure, for example a group or a vector space, and take the set of its subalgebras (subgroups, sub-spaces, resp.). This is a closure system, because the intersection of arbitrary subgroup is again a subgroup, the intersection of arbitrary subspaces is again a subspace, and, more generally, the intersection of subalgebras is a subalgebra.

  Similarly, consider the set $M := A^*$ of all (finite) words over some alphabet A. In other words, consider the free monoid over A. The set of its submonoids is a closure system, because any intersection of submonoids is closed under multiplication and contains the empty word.

- Or take the set of subintervals of the real interval $[0, 1]$, including the empty interval. Since any intersection of intervals is again an interval, we have a closure system.

  This example can be generalized to n dimensions, where we obtain the closure system of convex sets.

- The set of downsets of any ordered set $(M, \leq)$ is a closure system on M. A **downset** (or **order ideal**) of $(M, \leq)$ is a subset $D \subseteq M$ such that whenever $d \in D$ and $m \in M$ with $m \leq d$, then $m \in D$. Dually, the set of all **order filters** (or **upsets**, respectively) is a closure system.

- Consider all **preorders** on some fixed base set S, that is, all transitive reflexive relations $R \subseteq S \times S$. Any intersection of transitive reflexive relations is again transitive and reflexive. Therefore, preorders form a closure system.

3.1.2 Closure operators

Definition 21 A **closure operator** φ on M is a map assigning a **closure** $\varphi X \subseteq M$ to each set $X \subseteq M$, which is

monotone: $X \subseteq Y \Rightarrow \varphi X \subseteq \varphi Y$

extensive: $X \subseteq \varphi X$, and

idempotent: $\varphi\varphi X = \varphi X$.

(Conditions to be fulfilled for all $X, Y \subseteq M$.) $\Diamond$

For every example in the above list of closure systems we can also give a closure operator. In each of the following examples, φ is a closure operator on the set M.

- Let (M, E) be a tree with vertex set M and let φ be the mapping that maps each set X of vertices to the vertex set of the smallest subtree containing X.
- For a group with carrier set M, define φX to be the subgroup generated by X.

  For a vector space with carrier set M, define φX to be the subspace generated by X.

  For a set X of words over an alphabet A, let $\varphi X := X^*$ be the submonoid of $M := A^*$ which is generated by X.
- For any $X \subseteq M := [0, 1]$ let φX be the smallest interval containing X (i.e., the convex closure of X).
- If $(M, \leq)$ is an ordered set then for any $X \subseteq M$ let

  $$\varphi X := \{m \in M \mid m \leq x \text{ for some } x \in X\}$$

  be the downset generated by X.
- For any relation R on a set S, in other words, for any subset $R \subseteq S \times S =: M$, let φR denote the reflexive transitive closure of R.

The examples indicate why closure operators are so frequently met: their axioms describe the natural properties of a *generating process*. We start with some generating set X, apply the generating process and obtain the generated set, φX, the closure of X. Such generating processes occur in fact in many different variants in mathematics and computer science.

3.1.3 The closure systems of intents and of extents

Closure systems and closure operators are closely related. In fact, there is a natural way to obtain from each closure operator a closure system and vice versa. It works as follows:

Lemma 10 *For any closure operator, the set of all closures is a closure system. Conversely, given any closure system $\mathcal{C}$ on M, there is for each subset X of M a unique smallest set $C \in \mathcal{C}$ containing X. Taking this as the closure of X defines a closure operator. The two transformations are inverse to each other.*

Proof Exercise 1. $\square$

Thus closure systems and closure operators are essentially the same. We can add to this:

Theorem 11 *A closure system $\mathcal{C}$ on a set M can be considered as a complete lattice, ordered by set inclusion $\subseteq$. The infimum of any subfamily $\mathcal{D} \subseteq \mathcal{C}$ is equal to $\bigcap \mathcal{D}$, and the supremum is the closure of $\bigcup \mathcal{D}$. Conversely we can find for any complete lattice L a closure system that is isomorphic to L.*

Proof Exercise 2. □

So closure systems and complete lattices are also very closely related.

It comes as no surprise that concept lattices can be subsumed under this relationship. It follows from the Basic Theorem (Thm. 4 (p. 24)) that the set of all concept intents of a formal context is closed under intersections and thus is a closure system on M. Dually, the set of all concept extents always is a closure system on G. The corresponding closure operators are just the two operators

$$X \mapsto X''$$

on M and G, respectively.

Conversely, given any closure system $\mathcal{C}$ on a set M, we can construct a formal context such that $\mathcal{C}$ is the set of concept intents. It can be concluded from the Basic Theorem that for example $(\mathcal{C}, M, \ni)$ is such a context. In particular, whenever a closure operator on some set M is considered, we may assume that it is the closure operator

$$A \mapsto A''$$

on the attribute set of some formal context (G, M, I).

Thus, closure systems and closure operators, complete lattices, systems of concept intents, and systems of concept extents: all these are very closely related. It is not appropriate to say that they are "essentially the same", but it is true that all these structures have the same degree of expressiveness; none of them is a generalization of another. A substantial result proved for one of these structures can usually be transferred to the others, without much effort.

3.2 Formal contexts in Computer Science: examples

We have seen that closure systems can be represented by formal contexts, and this is often a very convenient way of handling a closure system. It is often beneficial to ask if a given closure system can nicely be described by a formal context.

On the other hand, formal contexts occur in mathematics and computer science without being named that way. It is no surprise that in such situations closure operators and complete lattices can be introduced.

3.2.1 Ontology Learning.

Ontologies are "explicit specification[s] of a conceptualization" [Gr94]. They usually consist of a set of concepts (not to be confused with formal concepts from FCA), a hierarchical is-a relation and other (non-hierarchical) relations between the concepts, and eventually axioms describing constraints on the relations and concepts. One task in learning ontologies from data is the construction of the is-a hierarchy. Suppose that the concepts are already learned (e. g., by applying linguistic and statistical methods [MaS00]) and stored in the set M. The set G contains instances, or documents annotated with the concepts. The relation I indicates if an instance belongs to a concept, or if a document is annotated with a concept. In [SM01], this approach has been used in FCA–MERGE, a technique for supporting the merging of ontologies. The concept lattice provides an is-a hierarchy on the set of the ontology concepts. Additionally, it suggests new concepts which may simplify the structure of the ontology.

The use of (iceberg) concept lattices is not only restricted to knowledge discovery. Here we give some more examples of typical applications, in which FCA has been successfully applied in the past (before the introduction of TITANIC). Their purpose is to show that the weight function (whose existence is a necessary condition for the applicability of Titanic) naturally appears in a wide variety of domains.

3.2.2 Configuration space analysis.

In software re-engineering, one task is to analyze the source code of a given program where no (or relatively few) documentation is given. In [KS94], the use of Formal Concept Analysis for analyzing the configuration space of C++ programs is discussed. The set G of objects contains the lines of code, the set M consists basically of the C++ preprocessor symbols which appear in the code, and the relation I indicates which lines of code are governed by which preprocessor symbols. Decompositions of the concept lattice indicate possibilities for refactoring the code.

3.2.3 Transformation of class hierarchies.

In object-oriented languages, one aim is to simplify the class hierarchy according to a (number of) given program(s). In [ST98], this problem has been attacked by using concept lattices. In the scenario, the set M of attributes contains all data members and methods of a given class hierarchy, and the set G of objects consists of all variables and pointers of the program(s). The relation I basically indicates which variables and pointers are related to which data members and methods. The resulting concept lattice provides an improved hierarchy which can be used for restructuring the class hierarchy according to software engineering principles without the need to modify the source code.

3.2.4 Model classes and theories, equational classes and equational theories

Consider propositional formulae over a set X of variables. The potential models of such formulae are truth value assignments $\varepsilon : X \to \{\text{TRUE}, \text{FALSE}\}$. To express that a certain ε is a model of a formula f, one writes $\varepsilon \models f$. Denoting by $\mathcal{F}(X)$ the set of all formulae over X and by $\mathcal{M}(X) := \{\text{TRUE}, \text{FALSE}\}^X$ the set of all truth value assignments, we can define the formal context

$$(\mathcal{M}(X), \mathcal{F}(X), \models).$$

Given any subset $A \subseteq \mathcal{M}(X)$ of assignments, we may ask which formulae hold in all these; the set of these formulae is the **propositional theory** of A. Conversely, for each set $F \subseteq \mathcal{F}$ of formulae we have the set of assignments for which all these formulae hold; these are the **models**[1] of F. These two mappings

$$\begin{aligned}
\mathrm{Th} : \mathfrak{P}(\mathcal{M}(X)) \to \mathfrak{P}(\mathcal{F}(X)), \qquad & A \mapsto \{f \in \mathcal{F}(X) \mid a \models f \quad \forall\, a \in A\} \\
\mathrm{Mod} : \mathfrak{P}(\mathcal{F}(X)) \to \mathfrak{P}(\mathcal{M}(X)), \qquad & F \mapsto \{m \in \mathcal{M}(x) \mid m \models f \quad \forall\, f \in F\}
\end{aligned}$$

are of course the derivation operators of the context $(\mathcal{M}(X), \mathcal{F}(X), \models)$. We obtain two closure operators, one on $\mathcal{M}(X)$ and one on $\mathcal{F}(X)$. They are just the two closure operators

$$X \mapsto X''$$

associated to the derivation operators.

$$A \mapsto \mathrm{Mod}(\mathrm{Th}(A)) \qquad \text{for } A \subseteq \mathcal{M}(X)$$

maps each set of truth value assignments to itself, because each model set can completely be described by its theory;

$$F \mapsto \mathrm{Th}(\mathrm{Mod}(F)) \qquad \text{for } F \subseteq \mathcal{F}(X)$$

extends each set F of formulae by those formulae that follow (semantically) from F (to the theory generated by F).

The formal concepts of this formal context have as intents just the propositional theories and as extents their model classes.

A very similar example can be obtained for (universal) algebras of a fixed signature, except that in that case we have to introduce some restrictions in order to avoid proper classes (classes that are not sets). Writing

$$\underline{A} \models e$$

to express that the equation e holds in the algebra $\underline{A}$, we obtain, as above, model classes ("varieties") and theories ("equational theories").

3.2.5 Equivalence relations

A binary relation $\Theta \subseteq S \times S$ on a set S is called an **equivalence relation** if it is reflexive, transitive, and symmetric. It is easy to check that these properties are preserved under arbitrary intersections. Therefore the set of all equivalence relations on S is a closure system. The associated closure operator takes as input any relation $R \subseteq S \times S$ and outputs its reflexive, symmetric, transitive closure, that is, the smallest equivalence relation containing R.

It is easy to describe the supremum- and the infimum-irreducible equivalence relations in terms of their equivalence classes:

- An equivalence relation is supremum-irreducible iff all its classes, except one, have only one element and the exceptional class has exactly two elements.
- An equivalence relation is infimum-irreducible iff it has exactly two equivalence classes.

A formal context can be defined as

$$\left(\binom{S}{2}, \mathfrak{P}(S \setminus \{s\}) \setminus \{\emptyset\}, I\right), \qquad \text{(for some fixed } s \in S\text{)}$$

[1]A truth value assignment ε can be specified by giving the set $T \subseteq X$ of those variables to which the value TRUE is assigned:

$$T := \{x \in X \mid \varepsilon(x) = \text{TRUE}\}.$$

Models of a propositional theory can therefore be identified with certain subsets of the set X.

where $\binom{S}{2}$ denotes the set of all two-element subsets of S, and for $a, b \in S$, $C \subseteq S \setminus \{s\}$,

$$\{a, b\} \; I \; C : \iff |\{a, b\} \cap C| \neq 1.$$

The concept extents of this formal context are precisely all equivalence relations on S.

3.2.6 Orders, order filters, order ideals

From an arbitrary ordered set $(P, \leq)$ we can obtain several interesting parts, for example its *order ideals*, its *order filters*, and its *cuts*. Order filters and -ideals have been introduced above (on page 44). A **cut** of $(P, \leq)$ is a pair (A, B) of subsets $A, B \subseteq P$ such that

- A is the set of all lower bounds of B, and
- B is the set of all upper bounds of A.

From an ordered set we can also derive several formal contexts. It is not difficult to show that

- The cuts of $(P, \leq)$ are precisely the formal concepts of the context $(P, P, \leq)$.
- The order ideals of $(P, \leq)$ are precisely the concept extents of the formal context $(P, P, \not\geq)$.
- The order filters of $(P, \leq)$ are precisely the concept intents of the formal context $(P, P, \not\geq)$. In fact, (A, B) is a formal concept of $(P, P, \not\geq)$ iff A is a downset, B is an upset and $A = P \setminus B$.

3.2.7 Bracketings and permutations

3.3 The Next Closure algorithm

We present a simple algorithm that solves the following task: For a given closure operator an a finite set M, it computes all closed sets.

There are many ways to achieve this. Our algorithm is particularly simple. We shall discuss efficiency considerations below. It also allows many useful modifications, some of which will be used in our more advanced applications. We start with the simplest version.

3.3.1 Representing sets by bit vectors

We start by giving our base set M an arbitrary linear order, so that

$$M = \{m_1 < m_2 < \cdots < m_n\},$$

where n is the number of elements of M. Then every subset $S \subseteq M$ can conveniently be described by its **characteristic vector**

$$\varepsilon_S : M \to \{0, 1\},$$

given by

$$\varepsilon_S(m) := \begin{cases} 1 & \text{if} m \in S \\ 0 & \text{if} m \notin S \end{cases}.$$

For example, if the base set is

$$M := \{a < b < c < d < e < f < g\},$$

then the characteristic vector of the subset $S := \{a, c, d, f\}$ is 1011010. In concrete examples we prefer to write a cross instead of a 1 and a blank or a dot instead of a 0, similarly as in the cross tables representing formal contexts. The characteristic vector of the subset $S := \{a, c, d, f\}$ will therefore be written as

×	.	×	×	.	×	.

.

In this notation it is easy to see if a given set is a subset of another given set, etc.

The set $\mathfrak{P}(M)$ of all subsets of the base set M is naturally ordered by the subset–order $\subseteq$. This is a complete lattice order, and $(\mathfrak{P}(M), \subseteq)$ is called the **power set lattice** of M. The subset-order is a *partial order.* We can also introduce a *linear* or *total* order of the subsets, for example the **lexicographic** or **lectic order** $\leq$, defined as follows: Let $A, B \subseteq M$ be two distinct subsets. We say that A is *lectically smaller* than B, if the smallest element in which A and B differ belongs to B. Formally,

$$A < B \quad :\iff \quad \exists_i \, (i \in B,\ i \notin A,\ \forall_{j<i} \, (j \in A \iff j \in B)).$$

For example $\{a, c, e, f\} < \{a, c, d, f\}$, because the smallest element in which the two sets differ is D, and this element belongs to the larger set. This becomes even more apparent when we write the sets as vectors and interpret them as binary numbers:

$$\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ & & & \updownarrow & & & \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}.$$

Note that the lectic order extends the subset-order, i.e.,

$$A \subseteq B \Rightarrow A \leq B.$$

The following notation is helpful:

$$A <_i B \quad :\iff \quad (i \in B,\ i \notin A,\ \forall_{j<i} \, (j \in A \iff j \in B)).$$

In words: $A <_i B$ iff i is the smallest element in which A and B differ, and $i \in B$.

Proposition 12 *1. $A < B$ if and only if $A <_i B$ for some $i \in M$.*

2. If $A <_i B$ and $A <_j C$ with $i < j$, then $C <_i B$.

3.3.2 Closures in lectic order

We consider a closure operator

$$A \mapsto A''$$

on the base set M. To each subset $A \subseteq M$ it gives[2] its closure $A'' \subseteq M$. Our task is to find a list of all these closures. In principle we might just follow the definition, compute for each subset $A \subseteq M$ its closure A'' and include that in the list. The problem is that different subsets may have identical closures. So if we want a list that contains each closure *exactly once*, we will have to check many times if a computed closure already exists in the list. Moreover, the number of subsets is exponential: a set with n elements has 2^n subsets. The naive algorithm "for each $A \subseteq M$, compute A'' and check if the result is already listed" therefore requires an exponential number of lookups in a list that may have exponential size.

A better idea is to generate the closures in some predefined order, thereby guaranteeing that every closure is generated only once. The reader may guess that we

[2]For our algorithm it is not important *how* the closure is computed.

shall generate the closures in lectic order. We will show how to compute, given a closed set, the *lectically next* one. Then no lookups are necessary. Actually, it will not even be necessary to store the list. For many applications it will suffice to generate the list elements on demand. Therefore we do not have to store exponentially many closed sets. Instead, we shall store just *one*!.

To find the next closure we define for $A \subseteq M$ and $m_i \in M$

$$A \oplus m_i := ((A \cap \{m_1, \ldots, m_{i-1}\}) \cup \{m_i\})''.$$

We illustrate this definition by an example: Let $A := \{a, c, d, f\}$ and $m_i := e$.

$$\begin{array}{|c|c|c|c|c|c|c|} \multicolumn{4}{c}{} & \multicolumn{1}{c}{\downarrow} & \multicolumn{2}{c}{} \\ \hline \times & . & \times & \times & . & \times & . \\ \hline \end{array}.$$

We first remove all elements that are greater or equal m_i from A:

$$\begin{array}{|c|c|c|c|c|c|c|} \multicolumn{4}{c}{} & \multicolumn{1}{c}{\downarrow} & \multicolumn{2}{c}{} \\ \hline \times & . & \times & \times & . & . & . \\ \hline \end{array}.$$

Then we insert m_i

$$\begin{array}{|c|c|c|c|c|c|c|} \multicolumn{4}{c}{} & \multicolumn{1}{c}{\downarrow} & \multicolumn{2}{c}{} \\ \hline \times & . & \times & \times & \times & . & . \\ \hline \end{array}$$

and form the closure. Since we have not yet specified the closure operator $\cdot''$ (i. e., we have not given a formal context), the example stops here with

$$A \oplus e = \{a, c, d, e\}''.$$

Proposition 13 1. *If $i \notin A$ then $A < A \oplus i$.*

2. *If B is closed and $A <_i B$ then $A \oplus i \subseteq B$, in particular $A \oplus i \leq B$.*

3. *If B is closed and $A <_i B$ then $A <_i A \oplus i$.*

Theorem 14 *The smallest closed set larger than a given set $A \subset M$ with respect to the lectic order is*

$$A \oplus i,$$

i being the largest element of M with $A <_i A \oplus i$.

Now we are ready to give the algorithm for generating all extents of a given context (G, M, I): The lectically smallest extent is $\emptyset''$. For a given set $A \subset G$ we find the lectically next extent by checking all elements i of $G \setminus A$, starting from the largest one and continuing in a descending order until for the first time $A <_i A \oplus i$. $A \oplus i$ then is the "next" extent we have been looking for. These three steps are made explicit in Figures 3.1 to 3.3.

```
Algorithm First Closure
Input:  A closure operator $X \mapsto X''$ on a finite set $M$.
Output:  The closure $A$ of the empty set.
begin
  $A := \emptyset''$;
end.
```

Figure 3.1: First Closure.

```
Algorithm NEXT CLOSURE
Input:   A closure operator $X \mapsto X''$ on a finite set $M$,
          and a subset $A \subseteq M$.
Output:  $A$ is replaced by the lectically next closed set.
begin
  $i$ := largest element of $M$;
  $i$ := succ($i$);
  success := false;
  repeat
    $i$ := pred($i$);
    if $i \notin A$ then
    begin
      $A := A \cup \{i\}$;
      $B := A''$;
      if $B \setminus A$ contains no element $< i$ then
      begin
        A:= B;
        success := true;
      end;
    end else $A := A \setminus \{i\}$;
  until success or $i =$ smallest element of $M$.
end.
```

Figure 3.2: Next Closure.

```
Algorithm ALL CLOSURES
Input:   A closure operator $X \mapsto X''$ on a finite set $M$.
Output:  All closed sets in lectic order.
begin
  First_Closure;
  repeat
    Output $A$;
    Next_Closure;
  until not success;
end.
```

Figure 3.3: Generating all closed sets.

There are several implementations of this algorithm. The best-known is probably the program ConImp by Peter Burmeister, which is particularly common on DOS-computers. For the world of Unix there is a version named Concepts by Christian Lindig. Both programs are at present available for non-commercial purposes.[3]

3.3.3 Examples

———(to be written)———

[3]e.g. free of charge via the Internet:
ftp.mathematik.th-darmstadt.de:/pub/department/software/conceptanalysis
or ftp.ips.cs.tu-bs.de:/pub/local/softech/misc.

3.3.4 The number of concepts may be exponential

The problem of computing concept lattices has exponential worst-case complexity: The context $\mathbb{K} := (\{1,\dots,n\},\{1,\dots,n\},\neq)$ has n objects and n attributes, while its concept lattice $\underline{\mathfrak{B}}(\mathbb{K})$ has 2^n concepts. Therefore all algorithms necessarily have an exponential worst-case complexity. However, it is of interest to analyze the situation in more detail.

3.3.5 Computation time per concept is polynomial

———(to be written)———

3.4 Iceberg Lattices and Titanic

A current research domain common to both the AI and the database community is Knowledge Discovery in Databases (KDD). Here FCA has been used as a formal framework for implication and association rules discovery and reduction and for improving the response times of algorithms for mining association rules. We will discuss association rules later in Section 4.6.4.

In this section we show that, vice versa, FCA can also benefit from ideas used for mining association rules by presenting a second, efficient algorithm for computing concept lattices, called TITANIC.

In fact, TITANIC can be used for a more general problem: Computing arbitrary closure systems when the closure operator comes along with a so-called weight function.

We also introduce the notion of *iceberg concept lattices*. Iceberg concept lattices show only the top-most part of a concept lattice. Iceberg concept lattices have different uses in KDD: as conceptual clustering tool, as a visualization method – especially for *very large* databases –, and as we will see later, as a condensed representation of frequent itemsets, as a base of association rules, and as a visualization tool for association rules.

3.4.1 Iceberg Concept Lattices

In the worst case, the size of concept lattices grows exponentially with the size of the context. Hence for most applications one has to consider strategies for dealing with such large concept lattices.

In this section, we present an approach based on *frequent itemsets* as known from data mining [2]: Our *iceberg concept lattices* will consist only of the top-most concepts of the concept lattice. These are the concepts which provide the most global structuring of the domain:

Definition 22 Let $B \subseteq M$, and let minsupp $\in [0,1]$. The *support count* of the attribute set (also called itemset) B in $\mathbb{K}$ is $\mathrm{supp}(B) := \frac{|B'|}{|G|}$. B is said to be a *frequent* attribute set if $\mathrm{supp}(B) \geq$ minsupp.

A concept is called *frequent concept* if its intent is frequent. The set of all frequent concepts of a context $\mathbb{K}$ is called the *iceberg concept lattice* of the context $\mathbb{K}$. $\Diamond$

Lemma 15 *For all $B \subseteq M$, we have* $\mathrm{supp}(B) = \mathrm{supp}(B'')$.

Proof By applying Proposition 1, we obtain $\mathrm{supp}(B) = \frac{|B'|}{|G|} = \frac{|B'''|}{|G|}\mathrm{supp}(B'')$ $\square$

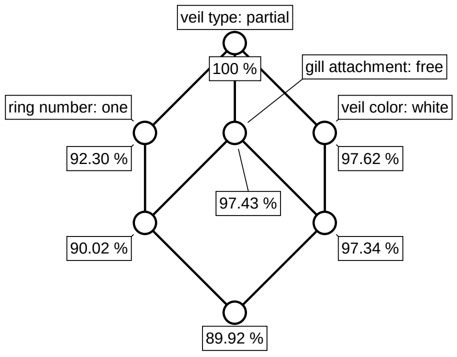


Figure 3.4: Iceberg concept lattice of the mushroom database with minsupp = 85 %

Because the support function is monotonously decreasing (i. e., $B_1 \subseteq B_2 \Rightarrow \text{supp}(B_1) \geq \text{supp}(B_2)$), the iceberg concept lattice is an order filter of the whole concept lattice, and thus in general only a join-semi-lattice. But when we add a new bottom element, it becomes a lattice again. This makes it possible to apply the same algorithm (which will be introduced in the following sections) for computing concept lattices and iceberg concept lattices. But before talking about their computation, let's have a closer look to iceberg concept lattices:

Example 4 As running example, we use the Mushroom database from the *UCI KDD Archive* (`http://kdd.ics.uci.edu/`). It consists of a database with 8,416 objects (mushrooms) and 22 (nominally valued) attributes. We obtain a formal context by creating one (Boolean) attribute for each of the 80 possible values of the 22 database attributes. The resulting formal context has thus 8,416 objects and 80 attributes. Its concept lattice consists of 32,086 concepts, hence is by far too large to be displayed. But for a first glance, it is sufficient to see its top-most part: Figure 3.4 shows the Mushroom iceberg concept lattice for a minimum support of 85 %.

In the diagram one can clearly see that all mushrooms in the database have the attribute 'veil type: partial'. Furthermore the diagram tells us that the three next-frequent attributes are: 'veil color: white' (with 97.62 % support), 'gill attachment: free' (97.43 %), and 'ring number: one' (92.30 %). There is no other attribute having a support higher than 85 %. But even the combination of all these four concepts is frequent (with respect to our threshold of 85 %): 89.92 % of all mushrooms in our database have one ring, a white partial veil, and free gills. This concept with a quite complex description contains more objects than the concept described by the fifth-most attribute, which has a support below our threshold of 85 %, since it is not displayed in the diagram.

In the diagram, we can detect the implication

$$\{\text{ring number: one, veil color: white}\} \Rightarrow \{\text{gill attachment: free}\} .$$

It means that all mushrooms with one ring and a white veil have free gills. Implications are discussed in more detail in the next chapter. This implication is indicated in the diagram by the fact that there is no concept having 'ring number: one' and 'veil color: white' (and 'veil type: partial') in its intent, but not 'gill attachment: free'. This implication has a support of 89.92 % (and as it is an implication, a confidence of 100 %). It is globally valid in the Mushroom database, i. e., it does not change when we consider a different minimum support.

If we want to see more details, we have to decrease the minimum support. Figure 3.5 shows the Mushroom iceberg concept lattice for a minimum support of

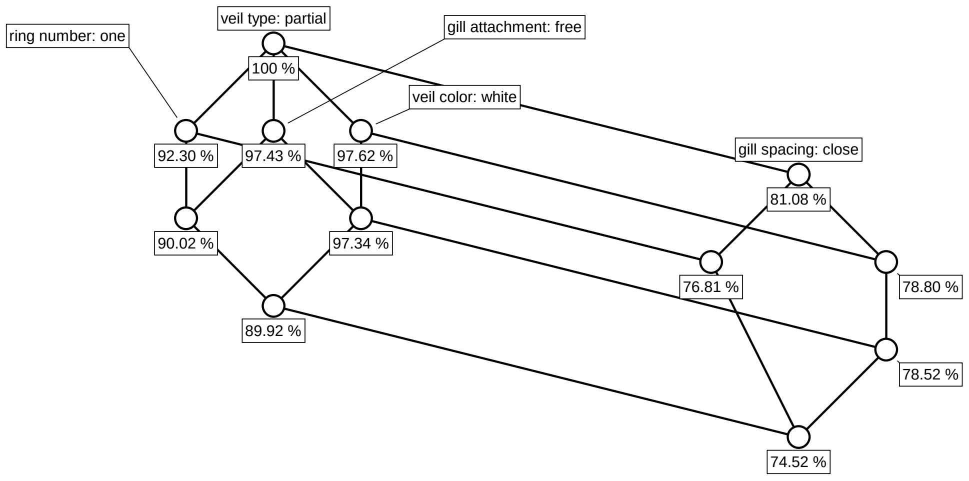


Figure 3.5: Iceberg concept lattice of the mushroom database with minsupp = 70 %

Table 3.1: Number of frequent closed itemsets and frequent itemsets for the Mushrooms example

minsupp	# frequent closed itemsets	# frequent itemsets
85 %	7	8
70 %	12	32
55 %	32	116
0 %	32.086	2^{80}

70 %. One observes that, of course, its top-most part is just the iceberg lattice for minsupp = 85 %. Additionally, we obtain five new concepts, having the possible combinations of the next-frequent attribute 'gill spacing: close' (having support 81.08 %) with the previous four attributes. The fact that the combination {veil type: partial, gill attachment: free, gill spacing: close} is not realized as a concept intent indicates another implication:

$$\{\text{gill attachment: free, gill spacing: close}\} \Rightarrow \{\text{veil color: white}\} \qquad (*)$$

This implication has 78.52 % support (the support of the most general concept having all three attributes in its intent) and — being an implication — 100 % confidence.

By further decreasing the minimum support, we discover more and more details. Figure 3.6 shows the MUSHROOMS iceberg concept lattice for a minimum support of 55 %. It shows four more partial copies of the 85 % iceberg lattice, and three new, single concepts.

The Mushrooms example shows that iceberg concept lattices are suitable especially for strongly correlated data. In Table 3.1, the size of the iceberg lattice (i. e., the number of all frequent closed itemsets) is compared with the number of all frequent itemsets. It shows for instance, that, for the minimum support of 55 %, only 32 frequent closed itemsets are needed to provide all information about the support of all 116 frequent itemsets one obtains for the same threshold.

The observation that the top-most part of the iceberg lattice appears partially again in combination with other attributes can be used for an alternative visualization: Figure 3.7 shows the iceberg concept lattice as a nested line diagram. The diagram provides exactly the same information than Figure 3.6, but in a more

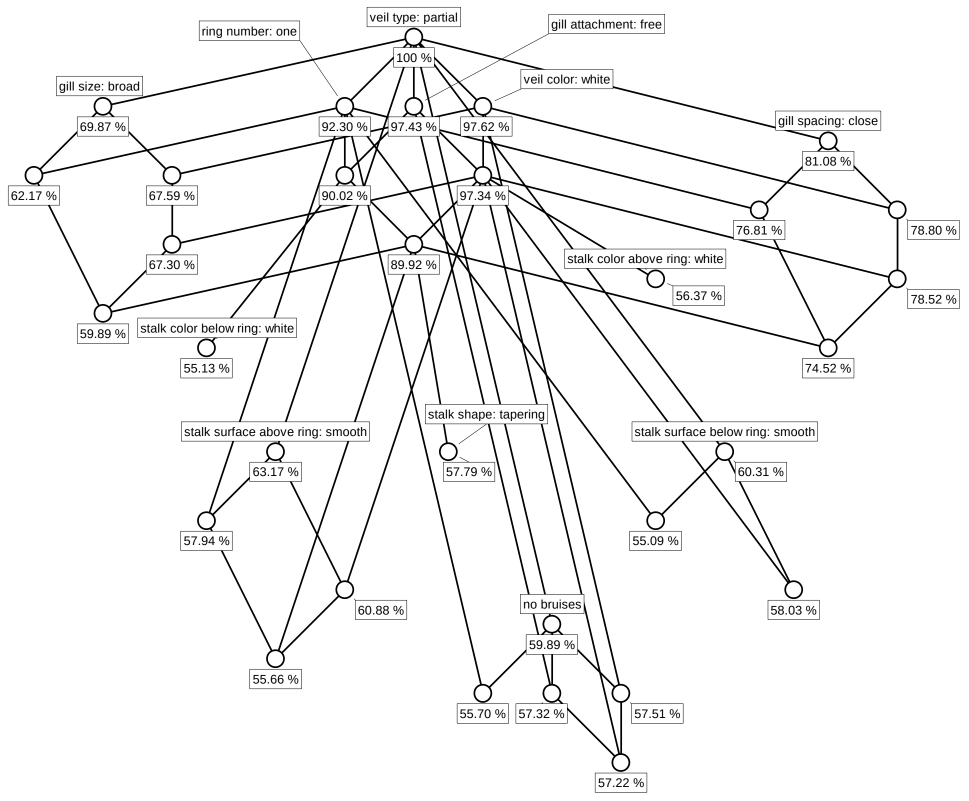


Figure 3.6: Iceberg concept lattice of the mushroom database with minsupp = 55 %

structured way.

Each of the 'satellites' contains a partial copy of the top-most iceberg lattice. Only those concepts are copied which are, together with the new attribute(s), still frequent. The lines of the outer diagram have to be read as a bundle of parallel lines, linking corresponding concepts. For instance, the concept on the right side of the diagram labeled by '78.80 %' is not only an immediate subconcept of the one labeled by '81.08 %, but also of the one labeled by '97.62 %'.

The empty circles indicate *unrealized concepts*: They are still frequent, but all objects in an unrealized concept share at least one more attribute. For instance, the unrealized concept on the right side left of the concept labeled by '78.80 %' has as intent {gill spacing: close, gill attachment: free, veil type: partial}. But implication (*) tells us that all objects having these attributes also have the attribute 'veil color: white'. Therefore, 'veil color: white' has to be in each realized concept which contains the three other attributes. The largest of them is just the first realized concept below: the one with 78.52 % support. This way, each unrealized concept indicates an implication: the attributes of its intent always imply all attributes in the intent of its largest realized subconcept. For instance, the two unrealized concepts below the attribute 'no bruises' indicate the implications

$$\{\text{no bruises, gill attachment: free}\} \Rightarrow \{\text{veil color: white}\}$$
$$\{\text{no bruises, veil color: white}\} \Rightarrow \{\text{gill attachment: free}\}$$

respectively, each having 57.22 % support.

For attributes which are labeled at concepts having no subconcepts in the diagram, we cannot decide whether they are part of interesting implications. For

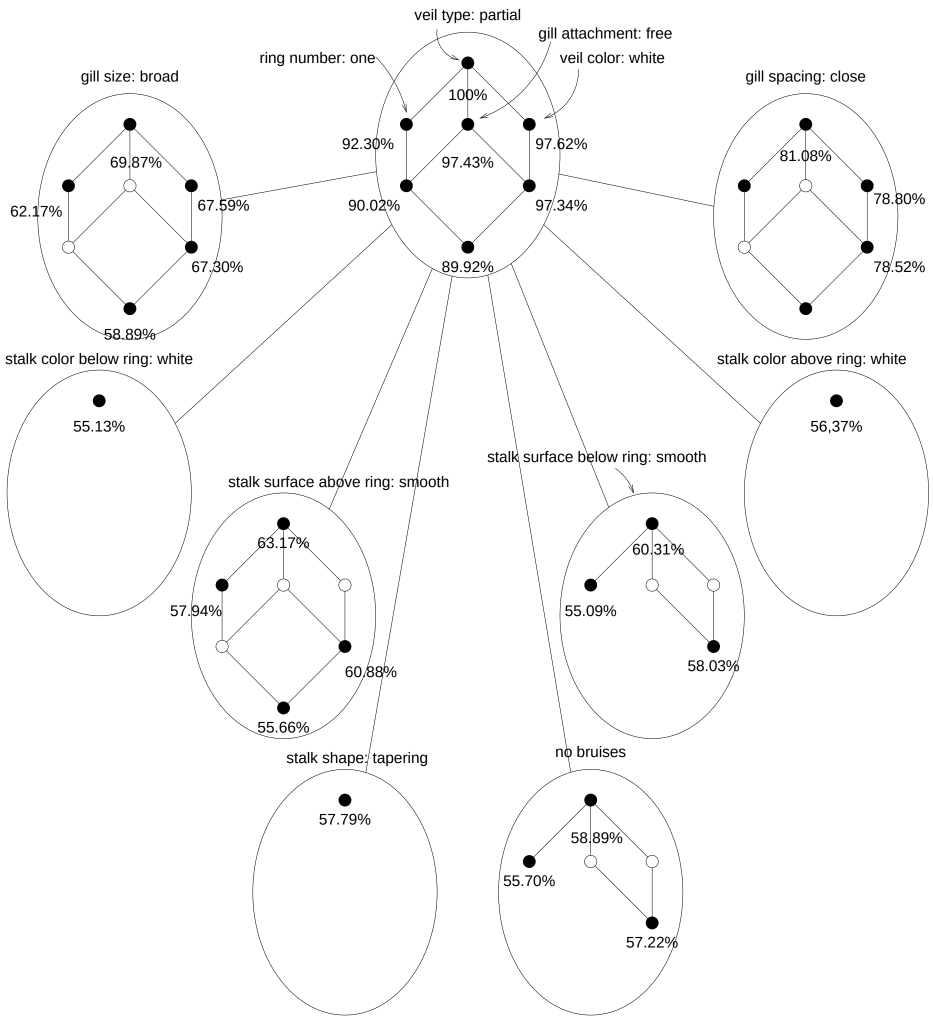


Figure 3.7: Nested line diagram of the iceberg concept lattice in Figure 3.6

instance, the diagram does not show whether there is an implication having 'stalk color below ring: white' in its premise or conclusion (other than the trivial implication {stalk color below ring: white} ⇒ {veil type: partial}). If there are any such rules, then their support is below the actual minimum support of 55 %. In order to study them, the threshold has to be decreased further.

In the way nested line diagrams are introduced in Section 2.4.2, the attributes are grouped manually according to their semantics. Related attributes are grouped together. This usually involves a human expert to decide which attributes are related. The support function, on the other hand, allows an automatic grouping: In Figure 3.7, the inner diagram contains the top-most attributes, the outer diagram the next-most attributes. The resulting diagram shows the most important attributes for structuring the domain. The knowledge engineer only has to fix the minimum support thresholds for the different layers.

Observe that the iceberg concept lattices in this example are used for *conceptual clustering*, which is a specific technique for *un-supervised learning*. Our aim was to gain new insights about the mushrooms in the database, independent from a specific purpose. In particular, the aim was not to learn how to distinguish between poisonous and edible mushrooms. The question if and how iceberg concept lattices can be used in such a *supervised learning* scenario is an interesting open problem.

In general, Cluster Analysis comprises a set of unsupervised machine learning techniques which split sets of objects into clusters (subsets) such that objects within a cluster are as similar as possible while objects from different clusters are as different as possible. Conceptual Clustering techniques additionally aim at determining not only clusters — i. e., concept extensions — but to provide at the same time intentional descriptions of these extensions. This aim fits well with the understanding of concepts formalized in FCA. Therefore FCA was considered as a framework for conceptual clustering from the early 1990ies on.

Compared to 'usual' clustering, conceptual clustering techniques pay their added value (the intentional description) with increased computation time. In FCA, there exist basically three ways to overcome this problem: local focusing (e. g., [CR93]), vertical reduction by conceptual scaling (see Chapter**??**, and horizontal reduction. Iceberg concept lattices are a horizontal approach to reduce the amount of information (and the computation time) of a concept lattice. In comparison to other conceptual clustering approaches, iceberg concept lattices have structural properties which can be stated explicitly: they do not depend on diverse parameters (except the minimum support threshold) whose semantics are often difficult to interpret, nor on the order in which the input is presented to the algorithm, nor on any particularities of the implementation. Another distinction to other hierarchical clustering results is that they allow for multiple hierarchies (and not only for trees), so that all potentially interesting specialization paths are contained in the resulting hierarchy.

Up to now, we have discussed the use of iceberg concept lattices as a conceptual clustering technique, equipped with a visualization method, which is very well suited especially for analyzing *very large* databases containing strongly correlated data. Now we briefly discuss some more uses of iceberg concept lattices in KDD:

A condensed representation of frequent itemsets. The computation of frequent attribute sets [itemsets] is the first (and most expensive) step in the computation of association rules. One reason is that one needs to count the support for each itemset. By using the fact that $\text{supp}(B) = \text{supp}(B'')$, for $B \subseteq M$, we can derive the supports of all itemsets from the supports of the frequent concept intents only. In strongly correlated data, only relatively few of the frequent itemsets are also concept intents. Hence only few support counts have to be effected in the database. This is used for the PASCAL algorithm [BTPSL00] which is related to TITANIC, and which efficiently computes frequent itemsets.

A starting point for computing bases of association rules. One problem in mining association rules is the large number of rules which are usually returned. In [BPTSL00] and [STBPL01], different bases for association rules are introduced, which prune redundant rules, but from which all valid rules can still be derived. The computation of the bases does not require all frequent itemsets, but only frequent concept intents.

A visualizing technique for association rules. We have already discussed how implications (i. e., association rules with 100 % confidence) can be read from the line diagram. The Luxenburger basis for approximate association rules (i. e., association rules with less than 100 % confidence), which is presented in [STBPL01], can also be visualized directly in the line diagram of an iceberg concept lattice. The Luxenburger basis is derived from [23]. It contains only those rules $B_1 \to B_2$ where B_1 and B_2 are frequent concept intents and where the concept (B_1', B_1) is an immediate superconcept of (B_2', B_2). Hence there corresponds to each approximate rule in the Luxenburger base exactly one edge in the line diagram. Figure 3.8

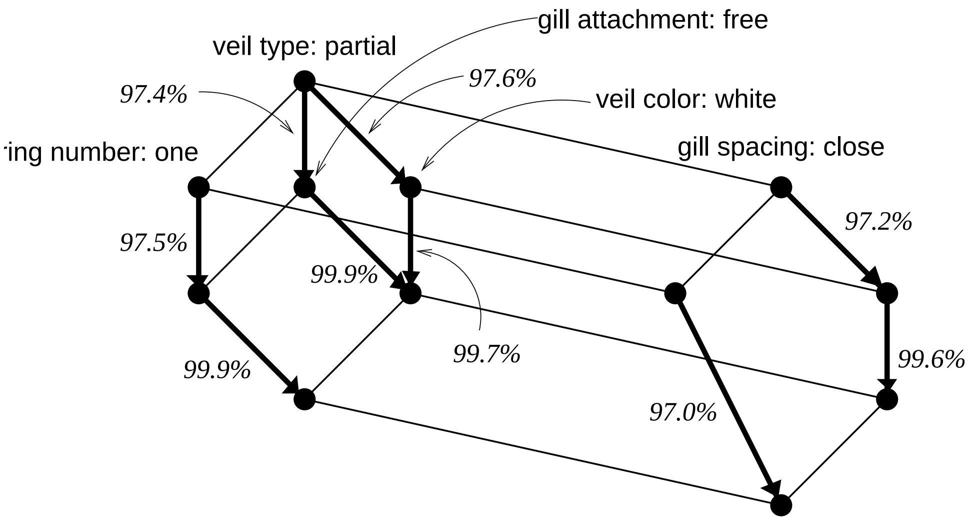


Figure 3.8: Visualization of the Luxenburger basis for minsupp = 70 % and minconf= 95 %

visualizes all rules in the Luxenburger basis for minsupp = 70 % and minconf = 95 %. For instance, the rightmost arrow stands for the association rule {veil color: white, gill spacing: close} → {gill attachment: free}, which holds with a confidence of 99.6 %. Its support is the support of the concept the arrow is pointing to: 78.52 %, as shown in Figure 3.5. Edges without label indicate that the confidence of the rule is below the minimum confidence threshold. The visualization technique is described in more detail in [STBPL01]. In comparison with other visualization techniques for association rules, the visualization of the Luxenburger basis within the iceberg concept lattice benefits of the smaller number of rules to be represented (without loss of information!), and of the presence of a 'reading direction' provided by the concept hierarchy.

3.4.2 Computing Closure Systems: the Problem

Instead of giving an algorithm for computing (iceberg) concept lattices, we provide an algorithm for a more general task: computing closure systems using a weight function. The reason is that closure systems are important in a variety of applications.

Section 3.1.3 shows that the set of all intents of a context (G, M, I) is a closure system on M, and that $B \mapsto B''$ is the corresponding closure operator. Thus computing concept lattices is a special case of the following, more general task:

Let h be a closure operator on a finite set M. The task is to determine efficiently the closure system $\mathcal{H}_h$ related to the closure operator h when there exists a *weight function* compatible with the closure operator:

Definition 23 A *weight function* on $\mathfrak{P}(M)$ is a function $s\colon \mathfrak{P}(M) \to P$ from the powerset of M to a totally ordered set $(P, \leq)$ having a largest element $s_{\max}$. For a set $X \subseteq M$, $s(X)$ is called the *weight* of X. The weight function is *compatible with a closure operator h* if

(*i*) $X \subseteq Y \Rightarrow s(X) \geq s(Y)$,

(*ii*) $h(X) = h(Y) \Rightarrow s(X) = s(Y)$, (*iii*) $X \subseteq Y \wedge s(X) = s(Y) \Rightarrow h(X) = h(Y)$.

$\Diamond$

Remark. In the sequel, we will consider $(P, \leq)$ to be the interval $[0, 1]$ in the real numbers, but the theory presented in this paper can be applied to arbitrary totally ordered sets.

Remark. If $X \subseteq Y \Rightarrow s(X) \leq s(Y)$ holds instead of (*i*) (as, e. g., for functional dependencies), then all 'min' in the sequel have to be replaced by 'max'.

Now we can formally state the problem:

Problem. Let h be a closure operator on a finite set M, and let s be a compatible weight function. Determine the closure system $\mathcal{H}_h$ related to the closure operator h by using the weight function s.

3.4.3 Computing Closure Systems Based on Weights

We discuss the problem of computing the closure system by using a weight function in three parts:

1. How can we compute the closure of a given set using the weight function only, and not the closure operator?
2. How can we compute the closure system by computing as few closures as possible?
3. Since the weight function is usually not stored explicitly, how can we derive the weights of as many sets as possible from the weights already computed?

Questions 2 and 3 are not independent from each other. Hence we will not provide an optimal answer for each of them, but one which improves the overall benefit.

Weight-based computation of closures

We use the constraints on the function s for determining the closure of a set by comparing its weight with the weights of its immediate supersets.

Proposition 16 *Let $X \subseteq M$. Then*

$$h(X) = X \cup \{m \in M \setminus X \mid s(X) = s(X \cup \{m\})\} .$$

Proof "$\subseteq$": Suppose that there exists $m \in h(X) \setminus X$ with $s(X) \neq s(X \cup \{m\})$. Then $h(X) \neq h(X \cup \{m\})$ by condition 2 of Definition 23. Hence $m \notin h(X)$. Contradiction.

"$\supseteq$": Let $m \in M \setminus X$ with $s(X) = s(X \cup \{m\})$. Then $h(X) = h(X \cup \{m\})$ by condition 3 of Definition 23. Hence $m \in h((X \cup \{m\})) = h(X)$. $\Box$

Hence if we know the weights of all sets, then we can compute the closure operator ($\rightarrow$ Algorithm 9, steps 3–7).[4] In the next subsection we discuss for which sets it is necessary to compute the closure in order to obtain all closed sets. In Subsection 3.4.3 we discuss how the weights needed for those computations can be determined.

[4]In this section, we give some references to the algorithms in the following section. These references can be skipped at the first reading.

A level-wise approach for computing all closed sets

One can now compute the closure system $\mathcal{H}$ by applying Proposition 16 to all subsets X of M. But this is not efficient, since many closed sets will be determined several times.

Definition 24 We define an equivalence relation θ on the powerset $\mathfrak{P}(M)$ of M by $(X, Y) \in \theta :\iff h(X) = h(Y)$, for $X, Y \subseteq M$. The equivalence class of X is given by $[X] := \{Y \subseteq M \mid (X, Y) \in \theta\}$. $\Diamond$

If we knew the equivalence relation θ in advance, it would be sufficient to compute the closure for one set of each equivalence class only. But since we have to determine the relation during the computation, we have to consider more than one element of each class in general. As known from algorithms for mining association rules, we will use a level-wise approach.

Definition 25 A *k-set* is a subset X of M with $|X| = k$. For $\mathcal{X} \subseteq \mathfrak{P}(M)$, we define $\mathcal{X}_k := \{X \in \mathcal{X} \mid X \text{ is } k\text{-set}\}$. For $\mathcal{X} = \mathfrak{P}(M)$, we also write $\mathfrak{P}_k(M)$ for $\mathcal{X}_k$. $\Diamond$

At the kth iteration, the weights of all k-sets which remained from the pruning strategy described below are determined; and the closures of all $(k-1)$-sets which passed the pruning in the $(k-1)$th iteration are computed.

The first sets of an equivalence class that we reach using such a level-wise approach are the minimal sets in the class:

Definition 26 A set $X \subseteq M$ is a *key set* (or *minimal generator*) if X is minimal (with respect to set inclusion) in $[X]$. The set of all key sets is denoted by $\mathcal{K}$. $\Diamond$

We have $\mathcal{H} = \{h(X) \mid X \in \mathcal{K}\}$, because there is at least one key set in each equivalence class of θ. Hence it is sufficient to compute the closures of all key sets.

In a sense the key sets are the first sets one reaches when traversing the powerset $\mathfrak{P}(M)$ level-wise:

Proposition 17 *The set $\mathcal{K}$ is an order ideal of $(\mathfrak{P}(M), \subseteq)$; i. e., $Y \in \mathcal{K}$ and $X \subseteq Y$ implies $X \in \mathcal{K}$, for all $X, Y \subseteq M$.*

Proof Let $X \subseteq Y$ and X be a non-key set. Then there exists a minimal $Z \in [X]$ with $Z \subset X$.[5] From $h(Z) = h(X)$ it follows that $h(Y) = h(Y \setminus (X \setminus Z))$. Hence Y is not minimal in $[Y]$ and thus by definition not a key set. $\Box$

The definition of an order ideal is equivalent to $X \notin \mathcal{K},\ X \subseteq Y \Rightarrow Y \notin \mathcal{K}$, for all $X, Y \subseteq M$. This allows to use a pruning strategy for determining the key sets. Originally the strategy we are going to apply was presented in [3], but only for a special case: as a heuristic for determining all frequent sets (which are, in our terminology, all sets with weights above a user-defined threshold). We recall this strategy, and show that it can be applied to arbitrary order ideals of the powerset of M:

Definition 27 Let $\mathcal{I}$ be an order ideal of $\mathfrak{P}(M)$. A *candidate set* for $\mathcal{I}$ is a subset of M such that all its proper subsets are in $\mathcal{I}$. $\Diamond$

The definition is justified by the fact that all combinations of the candidate sets can appear as $(k+1)$th level of an order ideal for which the first k levels are known. This statement is the subject of the first part of the following lemma. The second part states that non-candidate sets cannot appear at the $(k+1)$th level.

[5]We use $X \subset Y$ to say that $X \subseteq Y$ and $X \neq Y$.

Lemma 18 *Let $\mathcal{X} \subseteq \mathfrak{P}_k(M)$, and let $\mathcal{Y}$ be the set of all candidate $(k+1)$-sets for the order ideal $\downarrow\!\mathcal{X} := \{Y \in \mathfrak{P}(M) \mid \exists X \in \mathcal{X} \colon Y \subseteq X\}$ (i. e., the order ideal generated by $\mathcal{X}$).*

1. *For each subset $\mathcal{Z}$ of $\mathcal{Y}$, there exists an order ideal $\mathcal{I}$ of $\mathfrak{P}(M)$ with $\mathcal{I}_k = \mathcal{X}$ and $\mathcal{I}_{k+1} = \mathcal{Z}$.*
2. *For each order ideal $\mathcal{I}$ of $\mathfrak{P}(M)$ with $\mathcal{I}_k = \mathcal{X}$ the inclusion $\mathcal{I}_{k+1} \subseteq \mathcal{Y}$ holds.*

Proof *1.* Let $\mathcal{I} := (\downarrow\!\mathcal{X}) \cup \mathcal{Z}$. Let $Y \in \mathcal{I}$ and $X \subset Y$. We have to show that $X \in \mathcal{I}$. If $Y \in \downarrow\!\mathcal{X}$ then $X \in \downarrow\!\mathcal{X} \subseteq \mathcal{I}$ because $\downarrow\!\mathcal{X}$ is an order ideal. If $Y \in \mathcal{Z}$ then $X \in \downarrow\!\mathcal{X} \subseteq \mathcal{I}$ by Definition 27.

2. Suppose that there exists $Y \in \mathcal{I}_{k+1} \setminus \mathcal{Y}$. As $Y \notin \mathcal{Y}$, there exists $X \subset Y$ with $|X| = k$ and $X \notin \mathcal{I}_k$. Hence $Y \notin \mathcal{I}_{k+1}$. Contradiction. □

The efficient generation of the set of all candidate sets for the next level is described in the following proposition (→ Algorithm 2). We assume that M is linearly ordered, e. g., $M = \{1, \ldots, n\}$.

Proposition 19 *Let $\mathcal{X} \subseteq \mathfrak{P}_{k-1}(M)$. Let*

$$\widetilde{\mathcal{C}} := \{\{x_1 < x_2 < \cdots < x_k\} \mid \{x_1, \ldots, x_{k-2}, x_{k-1}\}, \{x_1, \ldots, x_{k-2}, x_k\} \in \mathcal{X}\} \quad ;$$

and

$$\mathcal{C} := \left\{X \in \widetilde{\mathcal{C}} \mid \forall x \in X \colon X \setminus \{x\} \in \mathcal{X}\right\} \quad .$$

Then $\mathcal{C} = \{X \in \mathfrak{P}_k(M) \mid X \text{ is candidate set for } \downarrow\mathcal{X}\}$.

Proof The definition of C is equivalent to $C := \{x \in \widetilde{C} \mid X \text{ is candidate set for } \downarrow\!\mathcal{X}\}$. Hence it remains to show that all candidate sets are included in $\widetilde{C}$. Let X be a candidate set, and let $X = \{x_1, \ldots, x_k\}$ with $x_1 < \cdots < x_k$. Since X is a candidate set, all its proper subsets are in $\downarrow\!\mathcal{X}$ — especially the two sets $\{x_1, \ldots, x_{k-2}, x_{k-1}\}$ and $\{x_1, \ldots, x_{k-2}, x_k\}$. Since they have cardinality k, they are also in $\mathcal{X}$. Hence $X \in \mathcal{I}$ by definition of $\widetilde{C}$. □

This generation procedure was first used in the Apriori algorithm [3] for the specific case of frequent itemsets.

Unlike in the Apriori algorithm, in our application the pruning of a set cannot be determined by its properties alone, but properties of its subsets (i. e., their weights) have to be taken into account as well. This causes an additional step in the generation function (→ Algorithm 2, step 5) compared to the version presented in [3]. Based on this additional step, at each iteration the non-key sets among the candidate sets are pruned (→ Algorithm 1, step 8) by using the second part of the following proposition.

Proposition 20 *Let $X \subseteq M$.*

1. *Let $m \in X$. Then $X \in [X \setminus \{m\}]$ if and only if $s(X) = s(X \setminus \{m\})$.*
2. *X is a key set if and only if $s(X) \neq \min\{s(X \setminus \{m\}) \mid m \in X\}$.*

Proof *1.* The "if" part follows from Definition 23 (*iii*), the "only if" part from Definition 23 (*ii*).

2. From 1. we deduce that X is a key set if and only if $s(X) \neq s(X \setminus \{m\})$, for all $m \in X$. Since s is a monotonous decreasing function, this is equivalent to 2. □

A candidate set X is hence pruned when $s(X) = \min\{s(X \setminus \{m\}) \mid m \in X\}$ holds.

Deriving weights from already known weights

If we reach a k-set which is known not to be a key set, then we already passed along at least one of the key sets in its equivalence class in an earlier iteration. Hence we already know its weight. Using the following proposition, we determine this weight by using only weights already computed.

Proposition 21 *If X is not a key set, then*

$$s(X) = \min\{s(K) \mid K \in \mathcal{K}, K \subseteq X\} \ .$$

Proof "$\geq$": Let K be a key set with $K\theta X$ and $K \subseteq X$. Then $s(X) = s(K) \geq \min\{s(K) \mid K \in \mathcal{K}, K \subseteq X\}$.

"$\leq$": Suppose that there exists $K \in \mathcal{K}$ with $K \subseteq X$ and $s(K) < s(X)$. Then $K \not\subseteq X$ by Definition 23 (i). Contradiction. □

Hence it is sufficient to compute the weights of the candidate sets only (by calling a function depending on the specific application → Algorithm 1, step 7). All other weights can be derived from those weights.

Now we are able to put all pieces together and to turn them into an algorithm.

3.4.4 The Titanic Algorithm

The pseudo-code is given in Algorithm 1. A list of notations is provided in Table 3.2.

Algorithm 1 Titanic

```
1) WEIGH($\{\emptyset\}$);
2) $\mathcal{K}_0 \leftarrow \{\emptyset\}$;
3) $k \leftarrow 1$;
4) forall $m \in M$ do $\{m\}.p\_s \leftarrow \emptyset.s$;
5) $\mathcal{C} \leftarrow \{\{m\} \mid m \in M\}$;
6) loop begin
7)    WEIGH($\mathcal{C}$);
8)    forall $X \in \mathcal{K}_{k-1}$ do $X$.closure $\leftarrow$ CLOSURE($X$);
9)    $\mathcal{K}_k \leftarrow \{X \in \mathcal{C} \mid X.s \neq X.p\_s\}$;
10)   if $\mathcal{K}_k = \emptyset$ then exit loop ;
11)   $k++$;
12)   $\mathcal{C} \leftarrow$ TITANIC-GEN($\mathcal{K}_{k-1}$);
13) end loop ;
14) return $\bigcup_{i=0}^{k-1}\{X.\text{closure} \mid X \in \mathcal{K}_i\}$.
```

The algorithm starts with determining the weight of the empty set (step 1) and stating that it is always a key set (step 2). Then all 1-sets are candidate sets by definition (steps 4+5).

In later iterations, the candidate k-sets are determined by the function Titanic-Gen (step 12 ⇝ Algorithm 2) which is (except step 5) a straight-forward implementation of Proposition 19. The result of step 5 of Algorithm 2 will be used in step 9 of Algorithm 1 for pruning the non-key sets according to Proposition 20(2).

Once the candidate k-sets are determined, the function Weigh($\mathcal{X}$) is called to compute, for each $X \in \mathcal{X}$, the weight of X and stores it in the variable $X.s$ (step 7).

Remark. In the case of concept lattices, Weigh determines the weights (i. e., the supports) of all $X \in \mathcal{X}$ with *a single pass* of the context (see Section 3.4.5).

Table 3.2: Notations used in TITANIC

k	is the counter which indicates the current iteration. In the kth iteration, all key k-sets are determined.
$\mathcal{K}_k$	contains after the kth iteration all key k-sets K together with their weight $K.s$ and their closure K.closure.
$\mathcal{C}$	stores the candidate k-sets C together with a counter $C.p\_s$ which stores the minimum of the weights of all $(k-1)$-subsets of C. The counter is used in step 9 to prune all non-key sets.

Algorithm 2 TITANIC-GEN

Input: $\mathcal{K}_{k-1}$, the set of key $(k-1)$-sets K with their weight $K.s$.

Output: $\mathcal{C}$, the set of candidate k-sets C
with the values $C.p\_s := \min\{s(C \setminus \{m\} \mid m \in C\}$.

The variables p_s assigned to the sets $\{m_1, \dots, m_k\}$ which are generated in step 1 are initialized by $\{m_1, \dots, m_k\}.p\_s \leftarrow s_{\max}$.

```
1) $\mathcal{C} \leftarrow \{\{m_1 < m_2 < \dots < m_k\} \mid \{m_1, \dots, m_{k-2}, m_{k-1}\}, \{m_1, \dots, m_{k-2}, m_k\} \in \mathcal{K}_{k-1}\}$;
2) **forall** $X \in \mathcal{C}$ **do begin**
3)   **forall** $(k-1)$-subsets $S$ of $X$ **do begin**
4)     **if** $S \notin \mathcal{K}_{k-1}$ **then begin** $\mathcal{C} \leftarrow \mathcal{C} \setminus \{X\}$; **exit forall** ; **end**;
5)     $X.p\_s \leftarrow \min(X.p\_s, S.s)$;
6)   **end**;
7) **end**;
8) **return** $\mathcal{C}$.
```

This is the reason why we call the function WEIGH for a set of sets instead of calling it for each set separately. In general, computing the weights of different sets simultaneously may or may not be more efficient than doing it separately, depending on the application.

For those sets which remained from the pruning (step 9) in the previous pass (and which are now known to be key sets), their closures are computed (step 8 $\rightsquigarrow$ Algorithm 9). The CLOSURE function (Algorithm 9) is a straight-forward implementation of Proposition 16 (steps 3–7) and Proposition 21 (step 5) plus an additional optimization (step 2).

In step 9 of Algorithm 1, all candidate k-sets which are not key sets are pruned according to Proposition 20 (2). Algorithm 1 terminates, if there are no key k-sets left (step 10). Otherwise the next iteration begins (step 11).

The correctness of the algorithm is proved by the theorems in the previous section. Examples for the algorithm are given in the next section.

3.4.5 Computing (Iceberg) Concept Lattices with TITANIC

In the sequel we will show that, for a given formal context, the support function fulfills the conditions of Definition 23 for being compatible to the closure operator $h(X) := X''$. Hence computing concept lattices is a typical application of the problem. We will also discuss how to modify the closure operator such that the problem description applies to iceberg concept lattices as well.

Algorithm 3 CLOSURE(X) for $X \in \mathcal{K}_{k-1}$

```
1) $Y \leftarrow X$;
2) **forall** $m \in X$ **do** $Y \leftarrow Y \cup (X \setminus \{m\})$.closure;
3) **forall** $m \in M \setminus Y$ **do begin**
4)    **if** $X \cup \{m\} \in \mathcal{C}$ **then** $s \leftarrow (X \cup \{m\}).s$
5)         **else** $s \leftarrow \min\{K.s \mid K \in \mathcal{K},\ K \subseteq X \cup \{m\}\}$;
6)    **if** $s = X.s$ **then** $Y \leftarrow Y \cup \{m\}$
7) **end**;
8) **return** $Y$.
```

We demonstrate the TITANIC algorithm by two examples: computing a concept lattice, and computing an iceberg concept lattice. For other applications (for instance those listed in Section 3.4.6), only the WEIGH function has to be adapted.

Computation of Concept Lattices

In the following, we will use the closure operator $B \mapsto B''$, for $B \subseteq M$. As said before, it is a closure operator on M. The related closure system (i. e., the set of all $B \subseteq M$ with $B'' = B$) is exactly the set of the intents of all concepts of the context. The structure of the concept lattice is hence already determined by this closure system. Therefore we restrict ourselves to the computation of the closure system of all concept intents in the sequel. The computation makes extensive use of the support function introduced in Definition 22. We show first that the support function fulfills the conditions of Definition 23:

Lemma 22 *Let $X, Y \subseteq M$.*

1. $X \subseteq Y \Rightarrow \text{supp}(X) \geq \text{supp}(Y)$
2. $X'' = Y'' \Rightarrow \text{supp}(X) = \text{supp}(Y)$
3. $X \subseteq Y \wedge \text{supp}(X) = \text{supp}(Y) \Rightarrow X'' = Y''$

Proof *1.* Let $X \subseteq Y$. Then $Y' \subseteq X'$ by Proposition 1, which implies

$$\text{supp}(Y) = \frac{|Y'|}{|G|} \leq \frac{|X'|}{|G|} = \text{supp}(X).$$

2. $X\,\theta\,Y \iff X'' = Y'' \iff X''' = Y''' \iff X' = Y' \Rightarrow$

$$s(X) = \frac{|X'|}{|G|} = \frac{|Y'|}{|G|} = s(Y) \ .$$

3. supp(X) = supp(Y) implies $|X'| = |Y'|$, and $X \subseteq Y$ implies $X' \supseteq Y'$. Hence $X' = Y'$, since X' and Y' are finite. It follows $X'' = Y''$. □

Corollary 23 *The support count is a weight function which is compatible with the closure operator $X \mapsto X''$.*

Thus we can use TITANIC for computing concept lattices. In this special application, we can benefit from two optimizations:

1. In Algorithm 1, we can — in the case of (iceberg) concept lattices — replace step 1 by

   $$1') \quad \emptyset.s \leftarrow 1$$

   since we know that supp$(\emptyset) = 1$. We avoid one call of the WEIGH function.

Algorithm 4 The WEIGH algorithm for concept lattices

```
1) forall $X \in \mathcal{X}$ do $X.s \leftarrow 0$;
2) forall $g \in G$ do
3)    forall $X \in$ SUBSETS$(g', \mathcal{X})$ do $X.s++$;
4) forall $X \in \mathcal{X}$ do $X.s \leftarrow \frac{X.s}{|G|}$;
```

	edible (e)	poisonous (p)	cap shape: convex (c)	cap shape: flat (l)	cap surface: fibrous (i)
Mushroom 1	×		×		
Mushroom 2	×		×		×
Mushroom 3	×			×	×
Mushroom 4	×			×	×
Mushroom 5	×		×		×
Mushroom 6	×		×		
Mushroom 7		×		×	×
Mushroom 8		×		×	×
Mushroom 9		×		×	×
Mushroom 10		×		×	

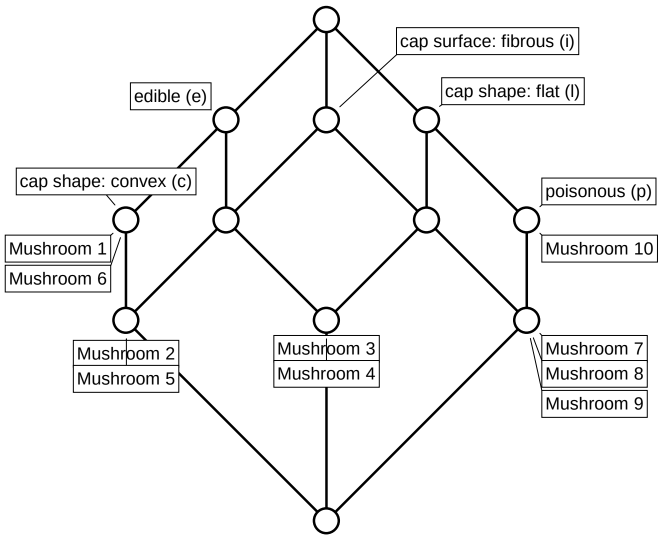


Figure 3.9: Example for the TITANIC algorithm

2. For concept lattices, WEIGH determines the weights — that is, the supports — of all $X \in \mathcal{X}$ with a single pass over the context. This is (together with the fact that only $\max\{|X| \mid X \subseteq M$ is candidate set$\}$ passes are needed) the reason for the efficiency of TITANIC. The WEIGH algorithm for concept lattices is given in Algorithm 4. SUBSETS$(Y, \mathcal{X})$ returns, for $Y \subseteq M$ and $\mathcal{X} \subseteq \mathfrak{P}(M)$, all $X \in \mathcal{X}$ with $Y \subseteq X$. It uses a tree structure with hash tables (as described in [PBTL98]) to efficiently encode $\mathcal{X}$.

Example 5 For explaining how TITANIC works, we will use the mushroom example again, but will reduce it to the first ten objects, and to the first five attributes (see Figure 3.9).

In the first pass, the algorithm deals with the empty set and all 1-sets. It returns the results for $k = 0$ and $k = 1$:

$k = 0$:

step 1		step 2
X	$X.s$	$X \in \mathcal{K}_k$?
$\emptyset$	1	yes

$k = 1$:

steps 4+5		step 7	step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e\}$	1	6/10	yes
$\{p\}$	1	4/10	yes
$\{c\}$	1	4/10	yes
$\{l\}$	1	6/10	yes
$\{i\}$	1	7/10	yes

Step 8 returns: $\emptyset$.closure $\leftarrow \emptyset$

Then the algorithm repeats the loop for $k = 2, 3$, and 4:

<u>$k = 2$:</u>

step 12		step 7	step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e,p\}$	4/10	0	yes
$\{e,c\}$	4/10	4/10	no
$\{e,l\}$	6/10	2/10	yes
$\{e,i\}$	6/10	4/10	yes
$\{p,c\}$	4/10	0	yes
$\{p,l\}$	4/10	4/10	no
$\{p,i\}$	4/10	3/10	yes
$\{c,l\}$	4/10	0	yes
$\{c,i\}$	4/10	2/10	yes
$\{l,i\}$	6/10	5/10	yes

Step 8 returns: $\{e\}$.closure $\leftarrow \{e\}$
$\{p\}$.closure $\leftarrow \{p,l\}$
$\{c\}$.closure $\leftarrow \{c,e\}$
$\{l\}$.closure $\leftarrow \{l\}$
$\{i\}$.closure $\leftarrow \{i\}$

<u>$k = 3$:</u>

step 12		step 7	step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e,l,i\}$	2/10	2/10	no
$\{p,c,i\}$	0	0	no
$\{c,l,i\}$	0	0	no

Step 8 returns: $\{e,p\}$.closure $\leftarrow \{e,p,c,l,i\}$
$\{e,l\}$.closure $\leftarrow \{e,l,i\}$
$\{e,i\}$.closure $\leftarrow \{e,i\}$
$\{p,c\}$.closure $\leftarrow \{e,p,c,l,i\}$
$\{p,i\}$.closure $\leftarrow \{p,l,i\}$
$\{c,l\}$.closure $\leftarrow \{e,p,c,l,i\}$
$\{c,i\}$.closure $\leftarrow \{e,c,i\}$
$\{l,i\}$.closure $\leftarrow \{l,i\}$

Since $\mathcal{K}_k$ is empty the loop is exited in step 10.

Finally the algorithm collects all concept intents (step 14):

$$\emptyset,\ \{e\},\ \{p,l\},\ \{c,e\},\ \{l\},\ \{i\},\ \{e,p,c,l,i\},$$
$$\{e,l,i\},\ \{e,i\},\ \{p,l,i\},\ \{e,c,i\},\ \{l,i\}$$

(which are exactly the intents of the concepts of the concept lattice in Figure 3.9). The algorithm determined the support of $5 + 10 + 3 = 18$ attribute sets in three passes of the database.

Equipping Titanic for Iceberg Concept Lattices

The structure of an iceberg concept lattice is determined by the semi-lattice of its frequent intents. If we add the set M (which is not frequent in general) to the set of frequent intents, it becomes a closure system. The next lemma presents its closure operator.

Algorithm 5 TITANIC improved for iceberg concept lattices

```
1)  $\emptyset.s \leftarrow 1$;
2)  $\mathcal{K}_0 \leftarrow \{\emptyset\}$;
3)  $k \leftarrow 1$;
4)  forall $m \in M$ do $\{m\}.p\_s \leftarrow \emptyset.s$;
5)  $\mathcal{C} \leftarrow \{\{m\} \mid m \in M\}$;
6)  loop begin
7)    WEIGH($\mathcal{C}$);
8)    forall $X \in \mathcal{K}_{k-1}$ do $X$.closure $\leftarrow$ CLOSURE($X$);
9)    $\mathcal{K}_k \leftarrow \{X \in \mathcal{C} \mid X.s \neq X.p\_s\}$;
10)   if $\{X \in \mathcal{K}_k \mid X.s \neq -1\} = \emptyset$ then exit loop ;
11)   $k++$;
12)   $\mathcal{C} \leftarrow$ TITANIC-GEN($\mathcal{K}_{k-1}$);
13) end loop ;
14) return $\bigcup_{i=0}^{k-1}\{X.\text{closure} \mid X \in \mathcal{K}_i, X.s \neq -1\}$.
```

Lemma 24 *Let* $\mathbb{K} := (G, M, I)$ *be a context, and let* minsupp $\in [0, 1]$*. The set* $\mathcal{F} := \{B \subseteq M \mid (A, B) \in \underline{\mathfrak{B}}(\mathbb{K}), \text{supp}(B) \geq \text{minsupp}\} \cup \{M\}$ *is a closure system on* M*. Its closure operator is given by* $h(X) := X''$ *if* $\text{supp}(X) \geq \text{minsupp}$ *and* $h(X) := M$ *else. The weight function* $s(X) := \text{supp}(X)$ *if* $\text{supp}(X) \geq \text{minsupp}$ *and* $s(X) := -1$ *else is compatible with the closure operator.*

Proof $\widetilde{\mathcal{F}} := \{B \subseteq M \mid \text{supp}(B) \geq \text{minsupp}\} \cup \{M\}$ is a closure system, since it is closed under arbitrary intersections. $\text{Int}(\mathbb{K}) := \{B \subseteq M \mid (A, B) \in \underline{\mathfrak{B}}(\mathbb{K})\}$ is a closure system as shown in Section 3.1.3. Hence $\mathcal{F}$ is — as intersection of the two closure systems $\widetilde{\mathcal{F}}$ and $\text{Int}(\mathbb{K})$ — also a closure system. Verifying that h is the related closure operator and that s is compatible is straightforward. □

The lemma shows that the TITANIC algorithm as presented in Section 3.4.4 can directly be applied to iceberg concept lattices. However we can benefit from the fact that weight -1 indicates that the closure of the set is the whole set M. In this case we can improve the algorithm. The improved version is discussed now.

Algorithm 5 differs from Algorithm 1 in steps 1, 10, and 14; Algorithm 6 differs from Algorithm 2 in steps 1 and 4; and Algorithm 7 is extending Algorithm 9 by step 1. We discuss these differences step by step:

- Algorithm 5, step 1: See the remark about the first optimization in Section 3.4.5.
- Algorithm 5, step 10: The loop can be exited when no *or only infrequent* key sets remain, as they are not used for generating candidate sets in the next iteration (see Algorithm 6, step 1)
- Algorithm 5, step 14: The algorithm returns only frequent intents, i. e. only closures of frequent key sets.
- Algorithm 6, step 1: Only frequent key sets are used to construct new candidate sets. See next item.
- Algorithm 6, step 4: S is a candidate set only if all $(k - 1)$-subsets of S are frequent key sets, because sets containing an infrequent key set are known not to be key sets.

Algorithm 6 TITANIC-GEN for iceberg concept lattices

Input: $\mathcal{K}_{k-1}$, the set of key $(k-1)$-sets K with their support $K.s$.

Output: $\mathcal{C}$, the set of candidate k-sets C with the values
$C.p\_s := \min\{s(C \setminus \{m\} \mid m \in C\}$.

The variables p_s assigned to the sets $\{m_1, \ldots, m_k\}$ which are generated in step 1 are initialized by $\{m_1, \ldots, m_k\}.p\_s \leftarrow 1$.

```
1) $\mathcal{C} \leftarrow \{\{m_1 < m_2 < \cdots < m_{k-1} < m_k\} \mid \{m_1, \ldots, m_{k-2}, m_{k-1}\},$
                         $\{m_1, \ldots, m_{k-2}, m_k\} \in \{K \in \mathcal{K}_{k-1} \mid K.s \neq -1\}\};$
2) **forall** $X \in \mathcal{C}$ **do begin**
3)   **forall** $(k-1)$-subsets $S$ of $X$ **do begin**
4)     **if** $S \notin \mathcal{K}_{k-1}$ or $S.s = -1$ **then begin** $\mathcal{C} \leftarrow \mathcal{C} \setminus \{X\}$; **exit forall** ; **end**;
5)     $X.p\_s \leftarrow \min(X.p\_s, S.s)$;
6)   **end**;
7) **end**;
8) **return** $\mathcal{C}$.
```

Algorithm 7 CLOSURE for iceberg concept lattices

```
1) **if** $X.s = -1$ **then return** $M$;
2) $Y \leftarrow X$;
3) **forall** $m \in X$ **do** $Y \leftarrow Y \cup (X \setminus \{m\})$.closure;
4) **forall** $m \in M \setminus Y$ **do begin**
5)   **if** $X \cup \{m\} \in \mathcal{C}$ **then** $s \leftarrow (X \cup \{m\}).s$
6)         **else** $s \leftarrow \min\{K.s \mid K \in \mathcal{K},\ K \subseteq X \cup \{m\}\}$;
7)   **if** $s = X.s$ **then** $Y \leftarrow Y \cup \{m\}$
8) **end**;
9) **return** $Y$.
```

- Algorithm 7, step 1: If the weight of a set is -1, its closure must be M by Lemma 24.

As before, the function WEIGH($\mathcal{X}$) determines, in one pass of the context, for each $X \in \mathcal{X}$ the support of X and stores it in the variable $X.s$. If $s(X) <$ minsupp, then WEIGH returns $X.s \leftarrow -1$.

Example 6 Although TITANIC only needs three passes of the database to compute the iceberg lattice in Figure 3.4 (and four passes for the one in Figure 3.6), we decided not to use it as example for explaining the mechanism of TITANIC for iceberg lattices. The reason is, that at the first pass the algorithm has to handle 80 candidate itemsets of size one. Of course, this is no problem in praxis, but is too large for demonstration purposes. Therefore we reuse the context in Figure 3.9, and show the computation of its iceberg concept lattice for minsupp = 30 %.

In the first pass, the algorithm deals with the empty set and all 1-sets. It returns the results for $k = 0$ and $k = 1$. As no infrequent sets are considered here, the results are exactly the same as in Example 5:

<u>$k = 0$:</u>

step 1		step 2
X	$X.s$	$X \in \mathcal{K}_k$?
$\emptyset$	1	yes

<u>$k = 1$:</u>

steps 4+5		step 7	step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e\}$	1	6/10	yes
$\{p\}$	1	4/10	yes
$\{c\}$	1	4/10	yes
$\{l\}$	1	6/10	yes
$\{i\}$	1	7/10	yes

Step 8 returns: $\emptyset$.closure $\leftarrow \emptyset$

Then the algorithm repeats the loop for $k = 2$. Here, the first infrequent sets are reached:
<u>$k = 2$:</u>

step 12		step 7	step 9
X	$X.p\_s$	$X.s$	$X \in \mathcal{K}_k$?
$\{e,p\}$	4/10	-1	yes
$\{e,c\}$	4/10	4/10	no
$\{e,l\}$	6/10	-1	yes
$\{e,i\}$	6/10	4/10	yes
$\{p,c\}$	4/10	-1	yes
$\{p,l\}$	4/10	4/10	no
$\{p,i\}$	4/10	3/10	yes
$\{c,l\}$	4/10	-1	yes
$\{c,i\}$	4/10	-1	yes
$\{l,i\}$	6/10	5/10	yes

Step 8 returns: $\{e\}$.closure $\leftarrow \{e\}$
$\{p\}$.closure $\leftarrow \{p,l\}$
$\{c\}$.closure $\leftarrow \{c,e\}$
$\{l\}$.closure $\leftarrow \{l\}$
$\{i\}$.closure $\leftarrow \{i\}$

Remark. As the weight of the key sets $\{e,p\}$, $\{e,l\}$, $\{c,l\}$, and $\{c,i\}$ is -1, we know that these sets are infrequent (with respect to our minimum support threshold of 30 %). In the corresponding closure system, they will hence generate the whole set M. These infrequent key sets are important if we want to provide a basis for association rules. See [STBPL01] for details. If our aim is conceptual clustering, we can neglect these infrequent key sets and can improve the performance of the algorithm by modifying step 9 in Algorithm 5 to

$$9') \qquad \mathcal{K}_k \leftarrow \{X \in \mathcal{C} \mid X.s \neq X.p\_s \text{ and } X.s \neq 1\} \ .$$

This would yield 'no instead of 'yes' in the last column for the five sets mentioned above.

<u>$k = 3$:</u>

Step 12 returns the empty set (because of the condition $K.s \neq -1$ in step 1 of Algorithm 2). Hence there is nothing to weigh in step 7. Step 9 sets $\mathcal{K}_3$ equal to the empty set; and in step 10, the loop is exited.

Step 8 returns: $\{e,p\}$.closure $\leftarrow M$
$\{e,l\}$.closure $\leftarrow M$
$\{e,i\}$.closure $\leftarrow \{e,i\}$
$\{p,c\}$.closure $\leftarrow M$
$\{p,i\}$.closure $\leftarrow \{p,l,i\}$
$\{c,l\}$.closure $\leftarrow M$
$\{c,i\}$.closure $\leftarrow M$
$\{l,i\}$.closure $\leftarrow \{l,i\}$

Finally the algorithm collects all frequent concept intents (step 14):

$$\emptyset,\ \{e\},\ \{p,l\},\ \{c,e\},\ \{l\},\ \{i\},\ \{e,i\},\ \{p,l,i\},\ \{l,i\}$$

The resulting concept iceberg lattice is shown in Figure 3.10.

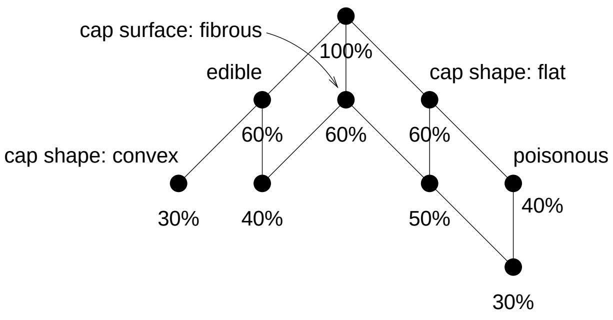


Figure 3.10: Iceberg concept lattice for the context in Figure 3.9 for minsupp = 30 %

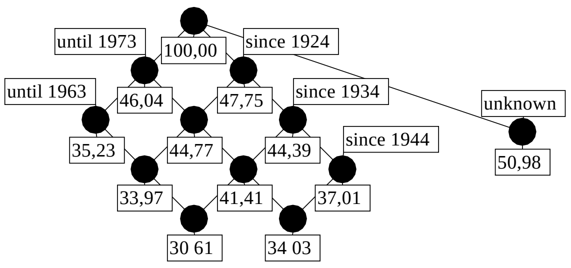


Figure 3.11: Iceberg concept lattice for customers clustered by their year of birth.

3.4.6 Some Typical Applications

In Section 3.4.1, we have already discussed the use of (iceberg) concept lattices for knowledge discovery and conceptual clustering. Here we present two examples, in which iceberg concept lattices have been applied:

Database marketing. The purpose of database marketing is the study of customers and their buying behavior in order to create and validate marketing strategies. In [19], the use of iceberg concept lattices for database marketing in a Swiss department store is discussed in more detail. In that scenario, the object set G consists of all customers of the warehouse paying by credit card, and the attribute set M consists of attributes describing the customers (e. g., 'lives in Western Switzerland') and their buying behavior (e. g., 'has spent more than 1000 Swiss francs in the last year'). For a given set X of attributes, the weight function returns the number of customers fulfilling all attributes in X. By decreasing the minimum support, one can study the customer clusters in more and more detail. In Figure 3.11, for instance, the customers of the warehouse are clustered according to their year of birth. The minimum support threshold is set to 0.3, i. e., all concepts whose extents do not comprise at least 30 % of all customers, are pruned.

Another situation where a weight function arises naturally in the computation of a closure system is the following. This scenario is more difficult to state in terms of a formal context:

Discovery of functional dependencies. One important task of logical database tuning is the discovery of minimal functional dependencies from database relations [HKPT99, LPL00]. This is equivalent to computing a closure system on the set M of all database attributes. The closed sets are just those which are closed under all functional dependencies which hold in the database. Titanic can be applied for this computation, using as weight of a given attribute set X the minimal number of rows which have to be deleted from the database such that X is closed under all functional dependencies which are valid for the remaining rows. This weight function is derived from the g_3 measure introduced in [KM95]. For this application, all 'min' in this section have to be replaced by 'max' (refer to Remark 2).

3.4.7 Complexity

There are several algorithms known for computing concept lattices: [MiS89], [15], [GM94], [NR99], [32], [31], and [PHM00]. The most efficient algorithm for practical applications to the best of our knowledge is Ganter's Next-Closure algorithm [15]; the algorithm with the best worst-case complexity is the In this section, we will compare Titanic with Next Closure (Section 3.3) and an algorithm from Nourine and Raynaud [NR99], which substantiates in an efficient way the approach described in Section 2.3.1.

As shown in Section 3.3.4, the problem of computing concept lattices has exponential worst-case complexity. However, for practical purposes, it is interesting to examine the situation in more detail. In the sequel, we assume that $|M| \leq |G|$.

The Next Closure algorithm computes the concepts sequentially. In Section 3.3.5 is shown that the complexity for computing one concept is in $O(|G| \cdot |M|^2)$, so that the overall complexity could be stated as $O(|\underline{\mathfrak{B}}(\mathbb{K})| \cdot (|G| \cdot |M|^2))$. For each concept, the context has to be accessed. If we consider additionally the access time db of the formal context (which can be significantly large when the context is too large to be stored in main memory!), we obtain $O(|\underline{\mathfrak{B}}(\mathbb{K})| \cdot (db + |G| \cdot |M|^2))$.

The algorithm of Nourine and Raynaud also computes the concepts sequentially. For each concept, the algorithm needs time $O((|M| + |G|) \cdot |G|)$, thus improving Ganter's worst-case complexity. Both algorithms need to access the context for each concept to be computed: If we add the access time db of the formal context to Nourine and Raynaud's algorithm, it is in $O(|\underline{\mathfrak{B}}(\mathbb{K})| \cdot (db + (|M| + |G|) \cdot |G|))$. On the other hand, Nourine and Raynaud's algorithm needs exponential space, since the whole lattice must be stored during run-time; while Next-Closure needs the context only, and has thus linear space complexity.

Both algorithms have different benefits. While Next-Closure needs only linear space, Nourine and Raynaud's algorithm provides the best worst-case complexity known so far. On the other hand, Next-Closure can be easily adapted to efficiently compute iceberg lattices, while the structure of Nourine and Raynaud's algorithm prohibits this. Furthermore, for the latter algorithm, the need to access the results computed so far makes it impractical for very large databases (contexts). Therefore, we will compare Titanic in the experimental evaluation with Ganter's Next-Closure algorithm only.

From a complexity point of view, Titanic is in between those two algorithms. Its worst-case space complexity is reached, when all $\left\lfloor \frac{|M|}{2} \right\rfloor$-sets are candidate sets. Then all these

$$\binom{|M|}{\left\lfloor \frac{|M|}{2} \right\rfloor} = \frac{|M| \cdot \dots \cdot \left(\left\lfloor \frac{|M|}{2} \right\rfloor + 1 \right)}{\left\lceil \frac{|M|}{2} \right\rceil \cdot \dots \cdot 1}$$

sets have to be stored. This is the widest level of the powerset of $|M|$, and its width grows exponentially relatively to $|M|$.

Titanic's time complexity can be determined as follows: The algorithm accesses the context as often as the size L of the largest candidate set is. This size is bounded by $|M|$, the height of the powerset of M. At each access, the algorithm considers a number of candidate sets. Let N be the maximal number of candidate sets considered at one of the accesses of the context. Then the time complexity is $O(L \cdot (db + N \cdot |G| \cdot |M|))$. By using the upper limits for L and N, we obtain

$$O\left(|M| \cdot \left(db + \binom{|M|}{\left\lfloor \frac{|M|}{2} \right\rfloor} \cdot |G| \cdot |M|\right)\right) \quad .$$

We see that the number of accesses of the context is at most $|M|$ (rather than $2^{|M|}$ as for the other two algorithms), which is especially important, when the context is so large that it doesn't fit into main memory. In that case, *db* can be a significant (or even the dominant) time factor.

The results show Titanic's worst-case complexity. In praxis the values for L and N are usually much lower. Especially for N (which contributes the exponentiality), the upper limit is, in the average case for computing iceberg concept lattices, the number of 2-itemsets, which is at most

$$\binom{|M|}{2} = \frac{|M| \cdot (|M| - 1)}{2} \quad .$$

3.5 Exercises

1. Give a proof of Lemma 10.
2. Give a proof of Theorem 11.
3. Let M be a set. Show that the set $\mathfrak{C} := \{\mathcal{C} \subseteq \mathfrak{P}(M) \mid (C) \text{ is a closure system on } M\}$ is a closure system on $\mathfrak{P}(M)$.

3.6 Bibliographic Notes

———(to be written)———

There are several algorithms known for computing concept lattices: [MiS89], [15], [GM94], [YLBC96], [NR99], [32], [31], and [PHM00]. The most efficient algorithm for practical applications to the best of our knowledge is Ganter's Next-Closure algorithm [15]; the algorithm with the best worst-case complexity is the one from Nourine and Raynaud presented in [NR99]. The latter one substantiates in an efficient way the approach proposed by R. Wille [42], which we presented in Section 2.3.1.

The way of computing concept lattices with Titanic follows a data mining viewpoint by using a level-wise approach [3, MT97]. Titanic was presented in [1], where also an experimental evaluation of its performance is discussed. Conceptual Clustering techniques were introduced first in [Mi80], see also [WMJ00]. FCA was considered as a framework for conceptual clustering from the early 1990ies on [StrW93, CR93, MG95].

FCA has been used as a formal framework for implication and association rules discovery and reduction [32, STBPL01] and for improving the response times of algorithms for mining association rules [31, 32]. The interaction of FCA and KDD in general has been discussed in [37] and [19].

Chapter 4

Implications

Have another look at the concept lattice shown in Figure 2.1 (p. 21). The six attributes describe how two unit squares can be placed with respect to each other. Each of the ten objects is a pair of unit squares, representing a possible placement. These ten pairs are representatives for an infinite set of possible positions that such pairs of squares may have. It is not stated, but perhaps expected by the reader, that these ten examples cover all possible combinations of the given attributes.

Such a situation occurs often: attributes are given, but objects are not known, or too many to handle them completely. We then have to study the possible attribute combinations, the *attribute logic* of the respective situation.

Let M be some set. We shall call the elements of M *attributes*, so as if we consider a formal context (G, M, I). However we do not assume that such a context is given or explicitly known.

Definition 28 An **implication between attributes** in M is a pair of subsets of M, denoted by $A \to B$. The set A is the **premise** of the implication $A \to B$, and B is its **conclusion**.

A subset $T \subseteq M$ **respects** an implication $A \to B$ if $A \not\subseteq T$ or $B \subseteq T$. We then also say that T is a **model** of the implication $A \to B$, and denote this by $T \models A \to B$. T **respects a set** $\mathcal{L}$ of implications if T respects every single implication in $\mathcal{L}$. The implication $A \to B$ **holds** in a set $\{T_1, T_2, \ldots\}$ of subsets if each of these subsets respects $A \to B$. With

$$\mathrm{Imp}\{T_1, T_2, \ldots\}$$

we denote the set of all implications that hold in $\{T_1, T_2, \ldots\}$. $\Diamond$

4.1 Implications of a formal context

Now let us consider the special case of implications of a formal context.

Definition 29 $A \to B$ **holds in a context** (G, M, I) if every object intent respects $A \to B$, that is, if each object that has all the attributes in A also has all the attributes in B. We then also say that $A \to B$ *is an implication of* (G, M, I). $\Diamond$

Proposition 25 *An implication $A \to B$ holds in (G, M, I) if and only if $B \subseteq A''$, which is equivalent to $A' \subseteq B'$. It then automatically holds in the set of all concept intents as well.*

An implication $A \to B$ holds in (G, M, I) if and only if each of the implications

$$A \to m, \qquad m \in B,$$

holds ($A \to m$ is short for $A \to \{m\}$). We can read this off from a concept lattice diagram in the following manner: $A \to m$ holds if the infimum of the attribute concepts corresponding to the attributes in A is less or equal than the attribute concept for m, formally if

$$\bigwedge\{\mu a \mid a \in A\} \leq \mu m.$$

$A \to B$ holds in (G, M, I) if

$$\bigwedge\{\mu a \mid a \in A\} \leq \bigwedge\{\mu b \mid b \in B\}.$$

4.2 Semantic and syntactical implication inference

As we will see, it is not necessary to store all implications of a formal context. We will discuss how implications can be derived from already known implications. First we discuss which kind of inference we want to model. This is given by the so-called *semantical inference*. Then we discuss a calculus (*syntactic inference*, and argue that the calculus is correct and complete with respect to our semantics.

4.2.1 When does an implication follow from other implications (semantically)?

Proposition 26 *If $\mathcal{L}$ is a set of implications in M, then*

$$\operatorname{Mod}\mathcal{L} := \{T \subseteq M \mid T \textit{ respects } \mathcal{L}\}$$

is a closure system on M. If $\mathcal{L}$ is the set of all implications of a context, then $\operatorname{Mod}\mathcal{L}$ *is the system of all concept intents.*

The respective closure operator

$$X \mapsto \mathcal{L}(X)$$

can be described as follows: For a set $X \subseteq M$, let

$$X^{\mathcal{L}} := X \cup \bigcup\{B \mid A \to B \in \mathcal{L}, A \subseteq X\}.$$

Form the sets $X^{\mathcal{L}}$, $X^{\mathcal{L}\mathcal{L}}$, $X^{\mathcal{L}\mathcal{L}\mathcal{L}}$, ... until[1] a set $\mathcal{L}(X) := X^{\mathcal{L}\cdots\mathcal{L}}$ is obtained with $\mathcal{L}(X)^{\mathcal{L}} = \mathcal{L}(X)$. Later on we shall discuss how to do this computation efficiently.

It is not difficult to construct, for any given set $\mathcal{L}$ of implications in M, a formal context such that $\operatorname{Mod}\mathcal{L}$ is the set of concept intents of this formal context. In fact, $(\operatorname{Mod}\mathcal{L}, M, \ni)$ will do.

Definition 30 An implication $A \to B$ **follows (semantically)** from a set $\mathcal{L}$ of implications in M if each subset of M respecting $\mathcal{L}$ also respects $A \to B$. A family of implications is called **closed** if every implication following from $\mathcal{L}$ is already contained in $\mathcal{L}$. A set $\mathcal{L}$ of implications of (G, M, I) is called **complete**, if every implication that holds in (G, M, I) follows from $\mathcal{L}$. ◊

In other words: An implication $A \to B$ follows semantically from $\mathcal{L}$ if it holds in every model of $\mathcal{L}$.

[1] If M is infinite, this may require infinitely many iterations.

Closure systems as extents

As an exercise in abstraction we demonstrate that the situation we have just discussed can neatly be formulated in formal context language. For a given set M, we may define a formal context

$$(\mathfrak{P}(M), \{A \to B \mid A, B \subseteq M\}, \models).$$

The attributes of this formal context are all implications in M, the objects are all subsets of M, and $T \models A \to B$ says that the subset T is a models of the implication $A \to B$. The derivation operators of this context are the operators Imp and Mod. The concept intents are precisely the complete sets of implications in M, the corresponding extents are precisely the closure systems on M. Therefore, the concept lattice of this formal context is isomorphic to the lattice of all closure systems on M, and dually isomorphic to the lattice of all closed implication sets on M.

We give an example for $M := \{a, b, c\}$, but display, for simplicity, only the *reduced* context. We omit some of the set brackets.

	$\emptyset \to a$	$\emptyset \to b$	$\emptyset \to c$	$a \to b$	$a \to c$	$b \to a$	$b \to c$	$c \to a$	$c \to b$	$a, b \to c$	$a, c \to b$	$b, c \to a$
$\emptyset$	.	.	.	×	×	×	×	×	×	×	×	×
$\{a\}$	×	.	.	.	.	×	×	×	×	×	×	×
$\{b\}$	.	×	.	×	×	.	.	×	×	×	×	×
$\{a, b\}$	×	×	.	×	.	×	.	×	×	.	×	×
$\{c\}$	.	.	×	×	×	×	×	.	.	×	×	×
$\{a, c\}$	×	.	×	.	×	×	×	×	.	×	.	×
$\{b, c\}$	.	×	×	.	×	.	×	.	×	×	×	.

Note that the extents of this formal context form a closure system, while each extent itself is a closure system. What we get is the *closure system of all closure systems on M*. Even more brain-twisting seems the question which implications hold in this context. The attributes themselves are implications. So we get implications between sets of implications. We shall come back to that later.

4.2.2 When does an implication follow from other implications (syntactically)?

The semantic definition of implication inference has a syntactic counterpart. We can give sound and complete inference rules (known as **Armstrong rules** [?]) and an efficient algorithm for inference testing.

Proposition 27 *A set $\mathcal{L}$ of implications in M is closed if and only if the following conditions are satisfied for all $W, X, Y, Z \subseteq M$:*

1. *$X \to X \in \mathcal{L}$,*
2. *If $X \to Y \in \mathcal{L}$, then $X \cup Z \to Y \in \mathcal{L}$,*
3. *If $X \to Y \in \mathcal{L}$ and $Y \cup Z \to W \in \mathcal{L}$, then $X \cup Z \to W \in \mathcal{L}$.*

Readers with a background in Computational Logic may prefer a different notation of these Armstrong rules:

$$\frac{}{X \to X}, \qquad \frac{X \to Y}{X \cup Z \to Y}, \qquad \frac{X \to Y, \quad Y \cup Z \to W}{X \cup Z \to W}.$$

The proposition says that a set of implications is the set of all implications of some context if and only if it is closed with respect to these rules. In other words, an implication follows from other implications if and only if it can be derived from these by successive applications of these rules. In particular, semantic and syntactic inference are the same.

However, these rules do not always suggest the best proof strategy. Instead, we may note the following:

Proposition 28 *An implication $X \to Y$ follows from a list $\mathcal{L}$ of implications if and only if $Y \subseteq \mathcal{L}(X)$.*

Proof $\mathcal{L}(X)$ is a model of $\mathcal{L}$ containing X. If $Y \not\subseteq \mathcal{L}(X)$, then $\mathcal{L}(X)$ is no model of $X \to Y$. Then, $X \to Y$ does not follow from $\mathcal{L}$. Conversely, any model of $\mathcal{L}$ that contains X must also contain $X^{\mathcal{L}}$ and, by induction, also contain $\mathcal{L}(X)$, which proves the Proposition. □

The Closure Algorithm

We give an algorithm that efficiently computes the closure $\mathcal{L}(X)$ for any given set X. Such algorithms are used in the theory of relational data bases for the study of functional dependencies. Later on, we shall show that this can be understood as a special instance of our approach.

```
Algorithm CLOSURE
Input:  A list $\mathcal{L} =: [\mathcal{L}[1], \dots, \mathcal{L}[n]]$ of implications in $M$
          and a set $X \subseteq M$.
Output:  The closure $\mathcal{L}(X)$ of $X$.
begin
  for all $x \in M$ do
  begin
    avoid[$x$] := $\{1, \dots, n\}$;
    for $i := 1$ to $n$ do with $A \to B := \mathcal{L}[i]$
      if $x \in A$ then avoid[$x$] :=avoid[$x$] $\setminus \{i\}$;
  end;
  used_imps :=$\emptyset$;
  old_closure :=$\{-1\}$;     (* some element not in $M$ *);
  new_closure := $X$;
  while new_closure $\neq$ old_closure do
  begin
    old_closure := new_closure;
    $T := M \setminus$new_closure;
    usable_imps := $\bigcap_{x \in T}$avoid[$x$];
    use_now_imps := usable_imps $\setminus$ used_imps;
    used_imps := usable_imps;
    for all $i \in$ use_now_imps with $A \to B := \mathcal{L}[i]$ do
      new_closure := new_closure $\cup B$;
  end;
  $\mathcal{L}(x)$ :=new_closure;
end.
```

Figure 4.1: Algorithm CLOSURE.

We can give a rough complexity estimation of the algorithm in Figure 4.1. Except for manipulations of addresses, the main effort is to apply the implications. Each implication is applied at most once, and each application requires a simple set operation. Therefore the time required by the Linclosure algorithm is essentially *linear* in the size of the input $\mathcal{L}$.

Linear complexity of implication inference

Summarizing these considerations we learn that implication inference is easy: to check if an implication $X \to Y$ follows from a list $\mathcal{L}$ of implications, it suffices to check if $Y \subseteq \mathcal{L}(X)$ (by Proposition 28), and this can be done in time linear in the size of the input.

In other words: implications are easy to use, much easier than many other logical constructs. This may be a reason why implications are popular, and perhaps be part of an explanation why our simple theory of formal concepts is so useful.

4.3 The Stem Base

The number of implications that hold in a given situation can be very large. For example, if there is only one closed set, M, then *every* implication holds. If M has n elements, then these are some 2^{2n} implications. But this is ridiculous, because all these implications can be inferred from a single one, namely from $\emptyset \to M$.

We see from this trivial example that the set of *all* implications of a given formal context may be highly redundant. It is a natural question to ask for a small *implicational base*, from which everything else follows. More precisely we ask, for any given formal context (G, M, I), for a list $\mathcal{L}$ of implications that is

- sound (i.e., each implication in $\mathcal{L}$ holds in (G, M, I)),
- complete (i.e., each implication that holds in (G, M, I) follows from $\mathcal{L}$), and
- non redundant (i.e., no implication in $\mathcal{L}$ follows from other implications in $\mathcal{L}$).

It is easy to see that (for finite M) such sets $\mathcal{L}$ exist. We may start with some sound and complete set of implications, for example, with the set of all implications that hold in (G, M, I). We then can successively remove redundant implications from this set, until we obtain a sound, complete, non redundant set.

But this is an unrealistic procedure. We therefore look for a better way to construct an implicational base. Duquenne and Guigues [**?**] have shown that there is a natural choice, the *stem base*.

4.3.1 A recursive definition

The following recursive definition is rather irritating on the first glance. We define a pseudo closed set to be a set which is not closed, but contains the closure of every pseudo-closed proper subset. More precisely,

Definition 31 Let $X \mapsto X''$ be a closure operator on the finite set M. We call a subset $P \subseteq M$ **pseudo-closed**, if (and only if)

- $P \neq P''$, and
- if $Q \subset P$ is a pseudo-closed proper subset of P, then $Q'' \subseteq P$.

$\Diamond$

This *is* a valid recursive definition. It is not circular, because the pseudo-closed set Q must have fewer elements than P, because it is a proper subset.[2]

Reformulating this definition for formal contexts, we obtain

Definition 32 Let (G, M, I) be a formal context with M finite. A subset $P \subseteq M$ is a **pseudo intent** of (G, M, I) iff

- P is not a concept intent, and
- if $Q \subset P$ is a pseudo-closed proper subset of P, then there is some object $g \in G$ such that $Q \subseteq g'$ but $P \not\subseteq g'$.

$\Diamond$

4.3.2 The stem base

Theorem 29 *Let M be finite and let $X \mapsto X''$ be some closure operator on M. The set of implications*

$$\{P \to P'' \mid P \text{ pseudo-closed } \}$$

is sound, complete, and non redundant.

The **Proof** is so simple that we can give it here. Obviously all implications of the form $X \to X''$ hold in the given closure system, therefore the set is sound. Suppose it were not complete. Then there is some implication $X \to Y$ that holds, but does not follow from the set $\mathcal{L}$ of implications given in the theorem. In particular, $Y \subseteq X''$ (because the implication holds) but $Y \not\subseteq \mathcal{L}(X)$ (because the implication does not follow). Then $\mathcal{L}(X) \neq X''$, and thus $\mathcal{L}(X)$ is not closed. Now let $Q \subset \mathcal{L}(X)$ be any pseudo-closed proper subset of $\mathcal{L}(X)$. Since the implication $Q \to Q''$ belongs to $\mathcal{L}$, Q'' must be in $\mathcal{L}(X)$.

Therefore, $\mathcal{L}(X)$ is not closed but contains the closure of every pseudo closed proper subset. By definition then $\mathcal{L}(X)$ must be pseudo-closed. But that gives a contradiction, because then the implication $\mathcal{L}(X) \to \mathcal{L}(X)''$ is in $\mathcal{L}$, which implies $\mathcal{L}(X)'' \subseteq \mathcal{L}(X)$ and consequently $\mathcal{L}(X) = \mathcal{L}(X)''$.

To see that the list $\mathcal{L}$ is non redundant, remove one of the implications, say, $P \to P''$. With respect to the smaller list $\mathcal{L}^-$ then P is closed (because P respects all implications $Q \to Q''$ in $\mathcal{L}^-$), and thus $\mathcal{L}^-(P) = P$. By Proposition 28, $P \to P''$ does not follow from $\mathcal{L}^-$. $\square$

The implication set in the theorem deserves a name. It is sometimes called the *Duquenne–Guigues–base.* We simply call it the **stem base** of the closure operator (or the *stem base of a given formal context*, if the closure operator is given that way). In practice one uses a slightly different version of the stem base, namely

$$\{P \to P'' \setminus P \mid P \text{ pseudo-closed } \}.$$

The stem base is not the only implicational base, but it plays a special rôle. For example, no implicational base can consist of fewer implications, as the next proposition shows:

Proposition 30 *Every sound and complete set of implications contains, for every pseudo closed set P, an implication $A \to B$ with $A'' = P''$.*

[2]'Recursive' is meant here with respect to set inclusion. Compare with the following recursive definition: A natural number is *prime* iff it is greater than 1 and not divisible by any smaller prime number.

4.3.3 (∗) Some open questions about pseudo-closed sets.

Pseudo closed sets are not well understood. It is not even known how many there are. More precisely: It is easy to give an example of a closure system with many pseudo-closed sets. Take, for a given base set M with $|M| = 2n > 2$, as closed sets all subsets with at most n elements, plus M itself. The pseudo-closed sets then are precisely the subsets with $n + 1$ elements. There are $\binom{2n}{n}$ of them, a number that is exponential in n. Thus the number of pseudo-closed sets can be exponential in comparison with the size of the base set.

But any formal context describing this closure system must have at least $\binom{2n}{n}$ objects, and thus itself be of a size exponential in n. In other words, the number of pseudo-closed sets in this example is large in comparison with the base set, but small in comparison with the formal context. We do not know if there are (infinite series of) contexts with *many* pseudo-closed sets, meaning that the number of pseudo-closed sets is exponential in the size of the formal contexts.

Another unresolved problem is how difficult it is to decide if a given set is pseudo-closed.

Is the following problem in $\mathcal{NP}$?

INSTANCE: A formal context (G, M, I) and a set $P \subseteq M$.

TASK: Decide if P is pseudo-closed.

Figure 4.2: An open problem on pseudo-closed sets

4.3.4 Making stem base implications shorter

We have stated above that the stem base is non redundant. This means that no implication in the stem base is dispensable. It is however possible that an implication in the base may be shortened, for example because it has a redundant premise. Such are occasionally counter–intuitive, in particular for mathematicians, who are trained to base their proofs on minimal resources. Similarly, the conclusion of an implication may be redundant. It may suffice to prove only part of it, and then to use other implications to obtain the rest.

It seems to be straightforward what to do: if an attribute is redundant in a premise or a conclusion, just omit it. But things get more complicated if two or more attributes are redundant. If each of m and n is redundant, it does not mean that they may be omitted *simultaneously*. In general, there is no guarantee that a set A contains a *unique* minimal generating set $S \subseteq A$ satisfying $A'' = S''$. There may be many such sets, and they may be difficult to find. For example, to find such a set with minimum possible cardinality is an $\mathcal{NP}$–hard problem. This can be derived from the following observations:

(to be completed . . .)

4.4 Computing the Stem Base

As before, consider a closure operator $X \mapsto X''$ on a finite set M. We start with a harmless definition:

Definition 33 A set $Q \subseteq M$ is **•-closed** if it contains the closure of every •-closed set that is properly contained in Q.

Formally, Q is $\bullet$-closed iff for each $\bullet$-closed set $Q_0 \subset Q$ with $Q_0 \neq Q$ we have $Q_0'' \subseteq Q$. $\Diamond$

This is a simple (but convenient) renaming, as the next proposition shows.

Proposition 31 *A set is $\bullet$-closed iff it is either closed or pseudo-closed.*

Observe that if Q contains the closure of every $\bullet$-closed subset, then Q must be closed.

4.4.1 $\bullet$-closed sets form a closure system

The first crucial step towards finding pseudo-closed sets is this:

Proposition 32 *The intersection of $\bullet$-closed sets is $\bullet$-closed.*

Proof Let Q_t, $t \in T$, be $\bullet$-closed and let $Q := \bigcap_{t\in T} Q_t$. We have to show that Q is $\bullet$-closed. Actually, we show more: Either $Q = Q_t$ for some $t \in T$, or Q is closed. Assume $Q \neq Q_t$ for all t, and let Q_0 $(0 \notin T)$ be some $\bullet$-closed set contained in Q. Then Q_0 is properly contained in each Q_t, $t \in T$, because Q is. Since each Q_t is $\bullet$-closed, the closure of Q_0 must also be contained in Q. Therefore Q is closed. $\square$

In other words: the $\bullet$-closed sets form a closure system. We have described an algorithm to compute, for a given closure operator, all closed sets. We can apply this algorithm for computing all $\bullet$-closed sets, provided that we can access the corresponding closure operator. This is easy. We prepare the result with a proposition that is an immediate consequence of Definition 33.

Proposition 33 *Q is $\bullet$-closed iff Q satisfies the following condition:*

If $P \subset Q$, $P \neq Q$, is pseudo-closed, then $P'' \subseteq Q$.

4.4.2 The closure operator

Proposition 33 shows how to find the quasi closure of an arbitrary set $S \subseteq M$: As long as the condition in the proposition is violated, we (are forced to) extend the set S, until we finally reach a fixed point.

Let $\mathcal{L}$ be the stem base[3]. Define, for $X \subseteq M$,

$$X^{\mathcal{L}^\bullet} := X \cup \bigcup\{P'' \mid P \to P'' \in \mathcal{L}, P \subset X, P \neq X\},$$

iterate by forming

$$X^{\mathcal{L}^\bullet\mathcal{L}^\bullet}, X^{\mathcal{L}^\bullet\mathcal{L}^\bullet\mathcal{L}^\bullet}, \ldots$$

until a set

$$\mathcal{L}^\bullet(X) := X^{\mathcal{L}^\bullet\mathcal{L}^\bullet\ldots\mathcal{L}^\bullet}$$

is obtained that satisfies

$$\mathcal{L}^\bullet(X) = \mathcal{L}^\bullet(X)^{\mathcal{L}^\bullet}.$$

Proposition 34 *$\mathcal{L}^\bullet(X)$ is the smallest $\bullet$-closed set containing X.*

Note that in order to find the quasi closure, we use only pseudo-closed sets which are contained in the closure and therefore in particular are lectically smaller than the quasi closure. Thus, the same result is obtained if $\mathcal{L}$ is a subset of the stem base, containing those implications $P \to P''$ for which P is pseudo-closed and lectically smaller than $\mathcal{L}^\bullet(X)$.

[3]The reader wonder why we use the stem base to construct the stem base. As we shall see soon, this works, due to the recursive definition of the stem base.

```
Algorithm $\mathcal{L}^\bullet$–Closure
Input:  A list $\mathcal{L} =: [\mathcal{L}[1], \ldots, \mathcal{L}[n]]$ of implications in $M$
          and a set $X \subseteq M$.
Output:  The closure $\mathcal{L}(X)$ of $X$.
begin
  for all $x \in M$ do
  begin
    avoid$[x] := \{1, \ldots, n\}$;
    for $i := 1$ to $n$ do with $A \to B := \mathcal{L}[i]$
      if $x \in A$ then avoid$[x] :=$avoid$[x] \setminus \{i\}$;
  end;
  used_imps :=$\emptyset$;
  old_closure :=$\{-1\}$;     (* some element not in $M$ *);
  new_closure := $X$;
  while new_closure $\neq$ old_closure do
  begin
    old_closure := new_closure;
    $T := M \setminus$ new_closure;
    usable_imps := $\bigcap_{x \in T}$avoid$[x]$ $\cap \bigcup_{x \in \text{new\_closure}}$avoid$[x]$;
    use_now_imps := usable_imps $\setminus$ used_imps;
    used_imps := usable_imps;
    for all $i \in$ use_now_imps with $A \to B := \mathcal{L}[i]$ do
      new_closure := new_closure $\cup B$;
  end;
  $\mathcal{L}(x)$ :=new_closure;
end.
```

Figure 4.3: Computing the $\mathcal{L}^\bullet$–Closure.

4.4.3 An algorithm for computing the stem base

Now it is easy to give an algorithm to compute all pseudo-closed sets for a given closure operator. We use the Next Closure algorithm applied to the closure system of •-closed sets. For short, we shall refer to this as the **next quasi closure** after a given set A, NEXT_$\mathcal{L}^\bullet$_CLOSURE(A). This produces all •-closed sets in lectic order. We record only those which are not closed. This yields a list of all pseudo-closed sets.

Since the •-closed sets are generated in lectic order, we have, at each step, the full information about the lectically smaller pseudo-closed sets. We have seen that this suffices to compute the "quasi closure" operator. The algorithm in Figure 4.4 uses a dynamic list $\mathcal{L}$. Whenever a pseudo-closed set P is found, the corresponding implication $P \to P''$ is included in the list. Since the pseudo-closed sets and found in lectic order, this makes sure that at any step we have sufficient information to compute the quasi closure.

4.4.4 An example

We compute the stem base for the context of triangles given on page 27. The steps are shown in figure 4.5. The first column contains all quasi closed sets, in lectic order. The pseudo-closed sets are precisely those which are not closed (see middle column). Each pseudo-closed set gives rise to an entry in the stem base (last column, short form). This stem base is given again, in slightly modified form, in Figure 5.1 (p. 100).

```
Algorithm STEM BASE
Input:   A closure operator $X \mapsto X''$ on a finite set $M$,
         for example given by a formal context $(G, M, I)$.
Output:  The stem base $\mathcal{L}$

begin
  $\mathcal{L} := \emptyset$;
  $A := \emptyset$;
  while $A \neq M$ do
  begin
    if $A \neq A''$ then $\mathcal{L} := \mathcal{L} \cup \{A \to A''\}$;
    $A :=$ NEXT_$\mathcal{L}^\bullet$_CLOSURE($A$);
  end;
end.
```

Figure 4.4: Computing the stem base for a given closure operator.

Since the closure operator is given in terms of a formal context, we may speak of **quasi intent**s and **pseudo intent**s instead of •-closed sets or pseudo-closed sets. We see that the algorithm generates all quasi intents to find the stem base. In other words, to compute all pseudo intents we also compute all intents, possibly exponentially many. This looks like a rather inefficient method. Unfortunately, we do not know of a better strategy. It is an open problem to find a better algorithm for the stem base. In practice, the algorithm is not fast, but nevertheless very useful.

•-closed set	closed ?	stem base implication
$\emptyset$	yes	
$\{e\}$	yes	
$\{d\}$	yes	
$\{d, e\}$	no	$\{d, e\} \to \{a, b, c\}$
$\{c\}$	yes	
$\{c, e\}$	no	$\{c, e\} \to \{a, b, d\}$
$\{c, d\}$	no	$\{c, d\} \to \{a, b, e\}$
$\{b\}$	yes	
$\{b, e\}$	yes	
$\{b, d\}$	yes	
$\{b, c\}$	yes	
$\{a\}$	no	$\{a\} \to \{b, c\}$
$\{a, b, c\}$	yes	
$\{a, b, c, d, e\}$	yes	

Figure 4.5: Steps in the stem base algorithm

4.5 Database Dependencies

How can we apply the theory of implications in the case of many-valued contexts? The attribute implications in the derived context offer one approach, however elementary. Basically, it describes implications between the individual attribute values, at least as long as we keep to plain scaling.

In colloquial language we use the term **dependency** of many-valued attributes as exemplified by the following sentence

> "The price of a real-estate depends on situation and size".

This is meant to express a simultaneous dependency of attribute values, perhaps even a gradual one, in the sense of "the larger the more expensive".

There are different notions of dependency for many-valued attributes, which correspond to the different possibilities of scaling. For an integration into a general theoretic framework, please refer to the corresponding literature.

We now describe the case of *functional dependency*, the (stronger) one of *ordinal dependency* and will indicate generalizations. For reasons of simplicity, we will first concentrate on complete many-valued contexts.

4.5.1 Functional dependencies

Definition 34 If $X \subseteq M$ and $Y \subseteq M$ are sets of attributes of a complete many-valued context (G, M, W, I), then we say that Y is **functionally dependent** on X if the following holds for every pair of objects $g, h \in G$:

$$(\forall_{m \in X}\ m(g) = m(h)) \Rightarrow (\forall_{n \in Y}\ n(g) = n(h)).$$

$\Diamond$

That is to say, if two objects have the same values with respect to all attributes from X the same must be true for the attributes from Y. This notion of dependency is often used in the theory of relational databases. The term "functional" can be explained as follows: Y is functionally dependent on X if and only if there is a map $f : W^X \to W^Y$ with

$$f(m(g) \mid m \in X) = (n(g) \mid n \in Y) \qquad \text{for all } g \in G.$$

In the case of ordinal dependency, we consider an ordinal context, i.e., we have for each attribute $m \in M$ an order $\leq_m$ on the set $m(G)$ of the values of m. (We obtain the special case of functional dependency if we take the equality relation for each of those orders.)

4.5.2 Ordinal dependencies

Definition 35 Let (G, M, W, I) be a complete many-valued context and let $\leq_m$ be an order relation on the set $m(G)$ of the values of m for every attribute $m \in M$. If $X \subseteq M$ and $Y \subseteq M$ are sets of attributes, we call Y **ordinally dependent** on X if the following holds for each pair of objects $g, h \in G$:

$$(\forall_{m \in X}\ m(g) \leq_m m(h)) \Rightarrow (\forall_{n \in Y}\ n(g) \leq_n n(h)).$$

$\Diamond$

Irrespective of which orders $\leq_m$ we have chosen, ordinal dependency always implies functional dependency, since from $m(g) = m(h)$ it follows that $m(g) \leq_m m(h)$ as well as $m(h) \leq_m m(g)$, and vice versa. Thus, one would expect that ordinal dependency is a kind of "order-preserving functional dependency". Intuitively, this is quite correct, but it is difficult to formulate, since the condition of being order-preserving is only required for the tuples $(m(g) \mid m \in X)$ that appear in the context. Not every map of this kind can be extended to form an order-preserving map of W^X to W^Y.

The ordinal dependencies (and as a special case within them the functional dependencies) of many-valued contexts can be expressed elegantly by implications of appropriate one-valued contexts. By means of the rule

$$(g,h) I_{\mathbb{O}}\, m \quad :\Longleftrightarrow\ m(g) \leq_m m(h),$$

we define a one-valued context

$$\mathbb{K}_{\mathbb{O}} := (G \times G, M, I_{\mathbb{O}})$$

for a complete many-valued context (G,M,W,I) with orders $\leq_m$ on the values. For the functional dependencies the context can be simplified further: It is possible to take advantage of the symmetry of the equality relation and to define

$$\mathbb{K}_{\mathbb{N}} := (\mathfrak{P}_2(G), M, I_{\mathbb{N}})$$

by

$$\{g,h\} I_{\mathbb{N}}\, m \quad :\Longleftrightarrow\ m(g) = m(h).$$

Then,

$$\mathfrak{P}_2(G) := \{\{g,h\} \mid g,h \in G, g \neq h\}.$$

The contexts defined in this way exactly fit the above-mentioned definitions of the dependencies and it is easy to prove the following proposition:

Corollary 35 *In (G,M,W,I) the attribute set Y is functionally dependent on X if and only if the implication $X \to Y$ holds in the context $\mathbb{K}_{\mathbb{N}}$. In (G,M,W,I) the attribute set Y is ordinally dependent on X if and only if the implication $X \to Y$ holds in the context $\mathbb{K}_{\mathbb{O}}$.* □

Hereby we have traced back the theory of functional and ordinal dependencies completely to the theory of implications. In particular, the algorithm mentioned in the previous section can also be used for the creation of a basis for the functional or ordinal dependencies, respectively.

The translation works even if the many-valued context (G,M,W,I) is not complete. In this connection, first of all we observe that Y is ordinally dependent on X if and only if this is true for every single attribute in Y, i.e., if $\{n\}$ is ordinally dependent on X for every $n \in Y$. This means that it is sufficient to state in which cases a single attribute is dependent on an attribute set. ... ———(to be written)———

4.6 Association rules

— This section is still in a preliminary form —

One of the core tasks of *Knowledge Discovery in Databases* (*KDD*) is the mining of association rules (conditional implications). *Association rules* are statements of the type '67 % of the customers buying cereals and sugar also buy milk (where 7% of all customers buy all three items)'. The task of mining association rules is to determine all rules whose *confidences* (67 % in the example) and *supports* (7 % in the example) are above user-defined thresholds. Since the problem was stated [2], various approaches have been proposed for an increased efficiency of rule discovery in very large databases [3, 8, 12, 31, 32]. However, fully taking advantage of exhibited rules means providing capabilities to handle them. The problem is especially critical when collected data is highly correlated or dense, like in statistical databases [12]. For instance, when applied to a census dataset of 10,000 objects, each of which characterized by values of 73 attributes, experiments result in more then 2,000,000 rules with support and confidence greater than or equal 90%. Thus the question arises: How can long lists of association rules be reduced in size?

Formal Concept Analysis allows to significantly reduce the number of rules without losing any information. We extract only a subset of all association rules, called *basis*, from which all other rules can be derived.

We use results of Duquenne and Guigues ([13], cf. also [16]) and Luxenburger [23, 24]. The former have studied bases (i. e., minimal non-redundant sets of rules from which all other rules can be derived) for association rules with 100 % confidence, and the latter association rules with less than 100 % confidence, but neither of them considered the support of the rules. We adopt their results to association rules (where both the support and the confidence are considered) and provide algorithms for computing the new bases by using *iceberg concept lattices* [40]. We follow an approach in two steps. In the first step, we compute the iceberg concept lattice for the given parameters. It consists of all FCA concepts whose extents exceed the user-defined minimum support. In the second step, we derive the bases for the association rules. In this paper, we focus on the second step. For the first step, we refer to the Pascal [7] and Titanic [39] algorithms.

This two-step approach has two advantages compared to the classical two-step approach [3] (which computes all frequent itemsets as intermediate result, and not only those which are intents of frequent FCA concepts):

1. It allows to determine bases for non-redundant association rules and thus to prune redundancy.

2. It speeds up the computation, especially for strongly correlated data or when the minimum support is low.

4.6.1 Formal Concept Analysis and the Association Rule Framework

In this section, we use (in the current version of this script) special notations. Therefore, we recall the basic definitions. ———(to be written)———

Definition 36 A *formal context* is a triple $\mathbb{K} := (G, M, R)$ where G and M are sets and $R \subseteq G \times M$ is a binary relation. A *data mining context* (or *dataset*) is a formal context where G and M are finite sets. Its elements are called *objects* and *items*, respectively. $(o, i) \in R$ is read as "object o is related to item i".

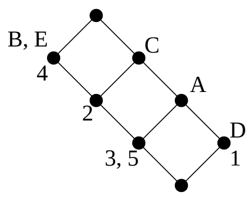
For $O \subseteq G$, we define $f(O) := \{i \in M \mid \forall o \in O\colon (o, i) \in R\}$; and for $I \subseteq M$, we define dually $g(I) := \{o \in G \mid \forall i \in I\colon (o, i) \in R\}$. A *formal concept* is a pair $(O, I) \in \mathfrak{P}(G) \times \mathfrak{P}(M)$ with $f(O) = I$ and $g(I) = O$. O is called *extent* and I is called *intent* of the concept. The set of all concepts of a formal context $\mathbb{K}$ together with the partial order $(O_1, I_1) \leq (O_2, I_2) :\Longleftrightarrow O_1 \subseteq O_2$ ($\Longleftrightarrow I_2 \subseteq I_1$) is a complete lattice, called *concept lattice* of $\mathbb{K}$.

In this setting, we call each subset of M also *itemset*, and each intent I also *closed itemset* (since it satisfies the equation $I = f(g(I))$). For two closed itemsets I_1 and I_2, we note $I_1 \prec I_2$ if $I_1 \subset I_2$ and if there does not exist a closed itemset I_3 with $I_1 \subset I_3 \subset I_2$.[4] $\diamondsuit$

In the following, we will use the composed function $h := f \circ g\colon \mathfrak{P}(M) \to \mathfrak{P}(M)$ which is a closure operator on M (i. e., it is extensive, monotonous, and idempotent). The related closure system (i. e., the set of all $I \subseteq M$ with $h(I) = I$) is exactly the set of the intents of all concepts of the context.

[4]We write $X \subset Y$ if and only if $X \subseteq Y$ and $X \neq Y$.

	A	B	C	D	E
1	×		×	×	
2		×	×		×
3	×	×	×		×
4		×			×
5	×	×	×		×



Exact rule	Supp	Approximate rule	Supp	Conf	Approximate rule	Supp	Conf	Approximate rule	Supp	Conf
ABC → E	0.4	BCE → A	0.4	2/3	C → ABE	0.4	2/4	C → BE	0.4	3/4
ABE → C	0.4	AC → BE	0.4	2/3	E → ABC	0.4	2/4	E → BC	0.6	3/4
ACE → B	0.4	BE → AC	0.4	2/4	A → BC	0.4	2/3	A → B	0.4	2/3
AB → CE	0.4	CE → AB	0.4	2/3	B → AC	0.4	2/4	B → A	0.4	2/4
AE → BC	0.4	AC → B	0.4	2/3	C → AB	0.4	2/4	C → A	0.6	3/4
AB → C	0.4	BC → A	0.4	2/3	A → BE	0.4	2/3	A → E	0.4	2/3
AB → E	0.4	BE → A	0.4	2/4	B → AE	0.4	2/4	E → A	0.4	2/4
AE → B	0.4	AC → E	0.4	2/3	E → AB	0.4	2/4	C → B	0.6	3/4
AE → C	0.4	CE → A	0.4	2/3	A → CE	0.4	2/3	C → E	0.6	3/4
BC → E	0.6	BE → C	0.6	3/4	C → AE	0.4	2/4	E → C	0.6	3/4
CE → B	0.6	A → BCE	0.4	2/3	E → AC	0.4	2/4	∅ → E	0.8	4/5
A → C	0.6	B → ACE	0.4	2/4	B → CE	0.6	3/4	∅ → BE	0.8	4/5
B → E	0.8	∅ → C	0.8	4/5	∅ → AC	0.6	3/5	∅ → BC	0.6	3/5
E → B	0.8	∅ → A	0.6	3/5	∅ → B	0.8	4/5	∅ → CE	0.6	3/5
								∅ → BCE	0.6	3/5

Figure 4.6: The example data mining context $\mathbb{K}$ and its concept lattice. The table shows all association rules that hold in $\mathbb{K}$ for *minsupp* $= 0.4$ and *minconf* $= 1/2$.

Definition 37 Let $I \subseteq M$, and let *minsupp*, *minconf* $\in [0,1]$. The *support count* of the itemset I in $\mathbb{K}$ is $supp(I) := \frac{|g(I)|}{|G|}$. I is said to be *frequent* if supp$(I) \geq$*minsupp.* The set of all frequent itemsets of a context is denoted FI.

An *association rule* is a pair of itemsets I_1 and I_2, denoted $I_1 \to I_2$, where $I_2 \neq \emptyset$. I_1 and I_2 are called *antecedent* and *consequent* of the rule, respectively. The *support* and *confidence* of an association rule $r := I_1 \to I_2$ are defined as follows: $\text{supp}(r) := \frac{|g(I_1 \cup I_2)|}{|G|}$, $\text{conf}(r) := \frac{supp(I_1 \cup I_2)}{supp(I_1)}$. If *conf(r)*=1, then r is called *exact association rule* (or *implication*), otherwise r is called *approximate association rule.*

An association rule r *holds* in the context if supp$(r) \geq$ minsupp and conf$(r) \geq$ minconf. The set of all association rules holding in $\mathbb{K}$ for given *minsupp* and *minconf* is denoted AR. ◊

Remark. The definition of association rules often includes the additional condition $I_1 \cap I_2 = \emptyset$. This condition helps pruning rules which are obviously redundant, as $I_1 \to I_2$ and $I_1 \to I_2 \setminus I_1$ have same support and same confidence. In this paper, we omit the condition, in order to simplify definitions. When discussing the algorithms, however, we will use the condition since it saves memory.

The association rule framework has first been formulated in terms of Formal Concept Analysis independently in [29], [38], and [43]. [29] provided also the first algorithm (named Close) based on this approach.

Example 7 An example data mining context $\mathbb{K}$ consisting of five objects (identified by their OID) and five items is given in Figure 4.6 together with its concept lattice. The association rules holding for *minsupp* $= 0.4$ and *minconf* $= 1/2$ are shown in the lower table.

In the *line diagram*, the name of an object g is always attached to the node representing the smallest concept with g in its extent; dually, the name of an attribute m is always attached to the node representing the largest concept with m in its intent. This allows us to read the context relation from the diagram because an object g has an attribute m if and only if there is an ascending path from the node labeled by g to the node labeled by m. The extent of a concept consists of all objects whose labels are below in the diagram, and the intent consists of all attributes attached to concepts above in the hierarchy. For example, the concept labeled by 'A' has $\{1, 3, 5\}$ as extent, and $\{\mathrm{A}, \mathrm{C}\}$ as intent.

An example for an exact rule (implication) which holds in the context is $\{\mathrm{A}, \mathrm{B}\} \to \{\mathrm{C}, \mathrm{E}\}$. It can also be read directly in the line diagram: the largest concept having both A and B in its intent is the one labeled by 3 and 5, and it is below or equal to (here the latter is the case) the largest concept having both C and E in its intent. This implication can be derived from two simpler implications, namely $\{\mathrm{A}\} \to \{\mathrm{C}\}$ and $\{\mathrm{B}\} \to \{\mathrm{E}\}$. The aim of the frequent Duquenne-Guigues-basis which we introduce in the next section is to provide only a minimal, non-redundant set of implications to the user. That basis will include the two simpler implications.

At the end of this section, we give some simple facts about association rules. We will refer to them later as derivation rules.

Lemma 36 *Rules 1 and 2 hold for* $\phi \in \{\mathrm{conf}, \mathrm{supp}\}$.

1. $\phi(X \to Y) = \phi(X \to Y \setminus Z)$, *for all* $Z \subseteq X \subseteq M$, $Y \subseteq M$.

2. $\phi(h(X) \to h(Y)) = \phi(X \to Y)$, *for all* $X, Y \subseteq M$.

3. $\mathrm{conf}(X \to Y) = p \wedge \mathrm{conf}(Y \to Z) = q \Rightarrow \mathrm{conf}(X \to Z) = p \cdot q$, *for all frequent concept intents* $X \subset Y \subset Z$.

3'. $\mathrm{supp}(X \to Z) = \mathrm{supp}(Y \to Z)$, *for all* $X, Y \subseteq Z$.

4. $\mathrm{conf}(X \to X) = 1$, *for all* $X \subseteq M$.

Proof The proofs for the confidence are given in [24].

1. $\mathrm{supp}(X \to Y) = \mathrm{supp}(X \to Y \setminus Z)$ follows from $X \cup Y = X \cup (Y \setminus Z)$ and the definition of the support count.

2. $\mathrm{supp}(h(X) \to h(Y)) = \mathrm{supp}(X \to Y)$ follows from $g(h(X) \cup h(Y)) = g(h(X)) \cap g(h(Y)) = g(f(g(X))) \cap g(f(g(Y))) = g(X) \cap g(Y) = g(X \cup Y)$ by using the facts $g(f(g(X))) = g(X)$ and $g(X \cup Y) = g(X) \cap g(Y)$ provided in [16].

3'. $\mathrm{supp}(X \to Z) = \frac{|g(X \cup Z)|}{|G|} = \frac{|g(Z)|}{|G|} = \frac{|g(Y \cup Z)|}{|G|} = \mathrm{supp}(Y \to Z)$

$\square$

4.6.2 Bases for Association Rules

In this section, we recall the definition of *iceberg concept lattices* and show that one can derive all frequent itemsets and association rules from them. Then we characterize the *Duquenne-Guigues basis for exact association rules* and the *Luxenburger basis for approximate association rules* and show that all other association rules can be derived from these two bases.

Definition 38 A concept (O, I) is called *frequent concept* if $\mathrm{supp}(I)\,(= \frac{|O|}{|G|}) \geq$ *minsupp*. The set of all frequent concepts is called *iceberg concept lattice*. An itemset I is called *frequent intent* (or *frequent closed itemset*) if it is intent of a frequent concept (i. e., its support is at least *minsupp*). The set of all frequent closed itemsets in $\mathbb{K}$ is denoted FC. $\Diamond$

Frequent closed itemset	support
∅	1.0
{C}	0.8
{AC}	0.6
{BE}	0.8
{BCE}	0.6
{ABCE}	0.4

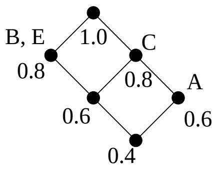


Figure 4.7: Frequent closed itemsets extracted from $\mathbb{K}$ for *minsupp* = 0.4.

Example 8 The frequent closed itemsets in the context $\mathbb{K}$ for *minsupp*=0.4 are presented in Figure 4.7 together with the semi-lattice of all frequent concepts. Both the table and the diagram provide the same information. Note that, in general, the set of frequent concepts is not a lattice, but only a semi-lattice (consider e. g. *minsupp*= 0.5 in the example).

Lemma 37 ([32]) *i) The support of an itemset I is equal to the support of the smallest closed itemset containing I, i. e., $supp(I) = supp(h(I))$.*

ii) The set of maximal frequent itemsets $\{I \in FI \mid \nexists I' \in FI \colon I \subset I'\}$ is identical to the set of maximal frequent closed itemsets $\{I \in FC \mid \nexists I' \in FC \colon I \subset I'\}$.

The next theorem shows that the set of frequent closed itemsets with their support is a small collection of frequent itemsets from which all frequent itemsets with their support and all association rules can be derived. I. e., it is a condensed representation in the sense of Mannila and Toivonen [25]. This theorem follows from Lemma 37.

Theorem 38 *All frequent itemsets and their support, as well as all association rules holding in the dataset, their support, and their confidence can be derived from the set FC of frequent closed itemsets with their support.*

Duquenne-Guigues Basis for Exact Association Rules

Next we present the Duquenne-Guigues basis for exact association rules. It is based on the following closure operator.

Theorem 39 *The set $FI \cup \{M\}$ is a closure system on M, and its related closure operator $\overline{\cdot}$ is given by $\overline{I} := h(I)$ if $supp(I) \geq minsupp$ and $\overline{I} := M$ else.*

Proof The set of all frequent itemsets together with M is a closure system, as well as the set of all concept intents. Hence $FI \cup \{M\}$ is, as intersection of those two closure systems, also a closure system. The proof of the fact that $\overline{\cdot}$ is the corresponding closure operator is straightforward. □

Our basis adopts the results of [13] to the association rule framework, where additionally the support of the rules has to be considered.

Definition 39 An itemset $I \subseteq M$ in $\mathbb{K}$ is a $\overline{\cdot}$*–pseudo-closed itemset* (or *pseudo-closed itemset* for short) [5] if $\overline{I} \neq I$ and for all pseudo-closed itemsets J with $J \subset I$, we have $\overline{J} \subset I$. The set of all frequent pseudo-closed itemsets in $\mathbb{K}$ is denoted *FP*, the set of all infrequent pseudo-closed itemsets is denoted *IP*. In the (unlikely) case that all itemsets are frequent except the whole set M, we let $IP := \{M\}$ (in order to distinguish this situation from the one where all itemsets are frequent).

The *Duquenne-Guigues basis for exact association rules* (or *frequent Duquenne-Guigues basis*) is defined as the tuple $FDG := (\mathcal{L}, IP)$ with $\mathcal{L} := \{I_1 \to h(I_1) \mid I_1 \in FP\}$ and IP as defined above. ◊

[5] We do not consider pseudo-closed itemsets with respect to other closure operators than $\overline{\cdot}$ (especially not with respect to h) in this paper.

Theorem 40 *From the Duquenne-Guigues basis for exact association rules one can derive all exact association rules holding in the dataset by applying the following rules. Rules ii) to iv) can be applied to $\mathcal{L}$ as long as they do not contradict (i).*

i) If there exists $I \in IP$ with $I \subseteq I_1 \cup I_2$, then $I_1 \to I_2$ does not hold (because its support is too low).

ii) $X \to X$ holds.

iii) If $X \to Z$ holds, then also $X \cup Y \to Z$.

iv) If $X \to Y$ and $Y \cup Z \to W$ hold, then also $X \cup Z \to W$.

Proof We only sketch the proof here, which applies results of [13] (see also [16]). One has to check that $\mathcal{L} \cup \{I{\to}M \mid I \in IP\}$ is the Duquenne-Guigues-basis (in the traditional sense, cf. to [13, 16]) of the closure system $FC \cup \{M\}$. Rule (i) reflects the implications of the form $I \to M$. □

The Duquenne-Guigues basis for exact association rules is not only minimal with respect to set inclusion, but also minimal with respect to the number of rules in $\mathcal{L}$ plus the number of elements in IP, since there can be no complete set with fewer rules than there are frequent pseudo-closed itemsets [13, 16]. Observe that, although it is possible to derive all exact association rules from the Duquenne-Guigues basis, it is not possible in general to determine their support.[6]

Example 9 The set of frequent pseudo-closed itemsets of $\mathbb{K}$ for *minsupp*=0.4 and *minconf*=1/2 is $FP = \{\{A\}, \{B\}, \{E\}\}$, the set of infrequent pseudo-closed itemsets is $IP = \{\{D\}\}$. The Duquenne-Guigues basis is presented in Figure 4.8.

Luxenburger Basis for Approximate Association Rules

In [23, 24], M. Luxenburger discusses bases for partial implications. A *partial implication* is an association rule where the support is not considered. He observed that it is sufficient to consider rules between concept intents only, since $\text{conf}(X \to Y) = \text{conf}(h(X) \to h(Y))$. However, his derivation process does not only consist of deduction rules which can be applied in a straightforward manner, but it requires to solve a system of linear equations.

In the KDD process, however, we have to consider the trade-off between the amount of information presented to the user, and the degree of its explicitness. The appearance of the system of linear equations indicates that Luxenburger's results are in favor for a minimal amount of information presented, and against a higher degree of explicitness. As one of the requirements to KDD is that the results should be "ultimately understandable" [14], we want to emphasize more on the explicitness of the results. Therefore we restrict now the expressiveness of the derivation process. This forces the association rules presented to the user to be more explicit.[7]

In the sequel, we consider the derivation rules given in Lemma 36. We present a basis for the approximate association rules for these derivation rules.

Definition 40 The *Luxenburger basis for approximate association rules* is given by $LB := \{(r, \mathit{supp}(r), \mathit{conf}(r)) \mid r = I_1 \to I_2,\ I_1, I_2 \in FC,\ I_1 \prec I_2,\ \mathit{conf}(r) \geq \mathit{minconf},\ \mathit{supp}(I_2) \geq \mathit{minsupp}\}$. ◊

[6]Even if the support for all rules in the basis is known. With the knowledge about all frequent closed itemsets and their support however, this is possible (see Theorem 38).

[7]Note that in the KDD setting the user will never actually perform longer series of inference steps.

Luxenburger basis		
Approximate rule	Support	Confidence
BCE → A	0.4	2/3
AC → BE	0.4	2/3
BE → C	0.6	3/4
C → BE	0.6	3/4
C → A	0.6	3/4
∅ → BE	0.8	4/5
∅ → C	0.8	4/5

Duquenne-Guigues basis	
$\mathcal{L}$	(Support)
A → C	0.6
B → E	0.8
E → B	0.8
$IP = \{\{D\}\}$	

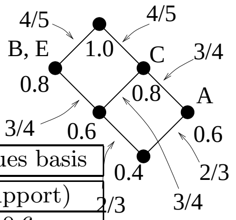


Figure 4.8: Duquenne-Guigues and Luxenburger bases for *minsupp*=0.4 and *minconf*=1/2.

Theorem 41 *From the Luxenburger basis LB for approximate association rules one can derive all association rules holding in the dataset together with their support and their confidence by using the rules given in Lemma 36. Furthermore, LB is minimal (with respect to set inclusion) with this property.*

Proof In order to determine if an association rule $r := I \to J$ holds in a context (and for determining its support and its confidence) one can consider the rule $I' \to J'$ with $I' := h(I)$ and $J' := h(I \cup J)$ which has (by Rules 1 & 2) the same support and the same confidence. If $I' = J'$, then $conf(r) = 1$ and $supp(r) = \text{supp}(I')$. If $I' \neq J'$, then exists a path of approximate rules, i. e., there are frequent closed itemsets $I_1, \ldots, I_n$ with $I_i \to I_{i+1} \in LB$ and $I' = I_1$ and $I_n = J'$. Support and confidence of r can now be determined by $supp(r) = supp(I_n)$ (Rule 3') and $conf(r) = \Pi_{i=1}^{n-1} conf(I_i \to I_{i+1})$ (Rule 3).

Now we show the minimality of LB. Let $r := I \to J \in LB$. We show that the confidence of r cannot be derived from $LB \setminus \{r\}$ by applying the rules of Lemma 2. Rule 1 cannot be applied forward since J already contains I. It cannot be applied backward because of $I \prec J$. Rule 2 cannot be applied forward since $I = h(I)$ and $J = h(J)$. It cannot be applied backward as LB contains only rules with closed antecedent and closed consequent. Rule 3 cannot be applied since there is no $K \subset M$ with $I \to K \in LB \setminus \{r\}$ and $K \to J \in LB \setminus \{r\}$ (because of $I \prec J$). Rule 4 cannot be applied since $I \neq J$. □

Remark. A basis in the sense of [24] is a maximal spanning tree of our basis (when considered as *undirected* graph) containing at most one rule with M as conclusion.[8]

Example 10 The Luxenburger basis for approximate association rules of $\mathbb{K}$ for *minsupp*=0.4 and *minconf*=1/2 is also presented in Figure 4.8. It provides the same information as the list in Figure 4.6, but in a more condensed form. The Luxenburger basis is visualized in the line diagram in Figure 4.8: From its definition it is clear, that each approximate rule in the basis corresponds to (at most)[9] one edge in the diagram. The edge is labeled by the confidence of the rule (as a fraction), and its lower vertice is labeled by its support (as a rational). Implications (exact rules) can be read in the diagram in the standard way described in Section 4.6.1.

As example for the proof of Theorem 41, let us check if $\{B\} \to \{A\}$ holds in the context for *minsupp*=0.4 and *minconf*=1/2. We have $I := \{B\}$ and $J := \{A\}$.

[8]The second condition is negligible in KDD, as it follows directly from minsupp$> 0\,\%$.

[9]In general, there may be edges which do not represent any rule in the Luxenburger basis. Consider for instance *minconf*$= 7/10$. In this case, the two lowest edges would not stand for a valid approximate rule, and would hence not be labelled.

The smallest frequent closed itemset containing B is $I' := \{B, E\}$ and the smallest one containing A and B is $J' := \{A, B, C, E\}$. In the diagram, I' and J' are always represented by the largest concepts which are below all attributes in I and $I \cup J$, resp. Between the two concepts we find the path $I_1 := I'$, $I_2 := \{B, C, E\}$, and $I_3 := J'$. Hence $\mathit{supp}(B \to A) = \mathit{supp}(J') = 0.4 \geq \mathit{minsupp}$ and $\mathit{conf}(B \to A) = \mathit{conf}(I_1 \to I_2) \cdot \mathit{conf}(I_2 \to I_3) = 3/4 \cdot 2/3 = 2/4 \geq \mathit{minconf}$, which means that the rule holds.

4.6.3 Algorithms for Computing the Bases

The algorithms presented in this paper assume that the iceberg concept lattice is already computed. There are several algorithms for computing iceberg concept lattices: the algorithm Close for strongly correlated data [32], the algorithm A-Close for weakly correlated data [31], the algorithms CLOSET [33], ChARM [44], and TITANIC [39, 40]. The algorithm PASCAL [7] computes all (closed and non-closed) frequent itemsets, but can be upgraded to determine also their closures with almost no additional computation time by using the fact that, for $I \subseteq M$,

$$h(I) = I \cup \{m \in M \setminus I \mid \mathit{supp}(I) = \mathit{supp}(I \cup \{m\})\} \ .$$

When the iceberg concept lattice is computed, then the Duquenne–Guigues basis and finally the Luxenburger basis are computed.

Generating the Duquenne-Guigues basis for Exact Association Rules with Gen-FDG

In this section, we present an algorithm that determines the Duquenne–Guigues basis using the iceberg concept lattice. This algorithm (which has not been presented before) implements Definition 39. As it needs to know the closure of frequent itemsets, it is best applied after an algorithm like PASCAL with the modification mentioned above, ChARM, or CLOSET.

The pseudo-code is given in Algorithm 8. The algorithm takes as input the sets FI_i, $1 \leq i \leq k$, containing the frequent itemsets and their support, and the sets FC_i, $0 \leq i \leq k$, containing the frequent closed itemsets and their support. It first computes the frequent pseudo-closed itemsets iteratively (steps 2 to 17). In steps 2 and 3, the empty set is examined. (It must be either a closed or a pseudo-closed itemset by definition.) The loop from step 4 to 17 is a direct implementation of Definition 39 for the frequent pseudo-closed itemsets. The frequent pseudo-closed i-itemsets, their closure and their support are stored in FP_i. They are used to generate the set $\mathcal{L}$ of implications of the Duquenne-Guigues basis for exact association rules DG (step 18).

The set of infrequent pseudo-closed itemsets is determined in steps 19 to 21 using the function $\mathcal{L}^*$-closure (Algorithm 9). This function uses the fact that, for a given closure system, the set of all closed or pseudo-closed sets forms again a closure system [15]. Hence one can generate all closed sets and pseudo-closed sets iteratively by using the corresponding closure operator $\mathcal{L}^*\text{-closure}(Z) := \bigcup_{i=0}^{\infty} Z_i$ with $Z_0 := Z$ and $Z_{i+1} := Z_i \cup \bigcup\{Y | X \to Y \in \mathcal{L}, X \subset Z_i\}$ [15]. The set $\mathcal{L}$ of implications has the form $\mathcal{L} = \{X_1 \to Y_1, \ldots, X_n \to Y_n\}$.

Generating the Luxenburger Basis for Approximate Association Rules with Gen-LB

The pseudo-code generating the Luxenburger basis for approximate association rules is presented in Algorithm 10. The algorithm takes as input the sets FC_i, $0 \leq i \leq$

Algorithm 8 Generating the Duquenne-Guigues basis with Gen-FDG.

```
1)  $\mathcal{L} \leftarrow \{\}$;
2)  if $(FC_0 = \{\})$ then $FP_0 \leftarrow \emptyset$;
3)  else $FP_0 \leftarrow \{\}$;
4)  for $(i \leftarrow 1;\ i \leq k;\ i{+}{+})$ do begin
5)      $FP_i \leftarrow FI_i \setminus FC_i$;
6)      forall $L \in FP_i$ do begin
7)          $pseudo \leftarrow true$;
8)          forall $P \in FP_j$ with $j < i$ do begin
9)              if $(P \subset L)$ and $(P$.closure $\not\subseteq L)$
10)             then do begin
11)                 $pseudo \leftarrow false$;
12)                 $FP_i \leftarrow FP_i \setminus \{L\}$;
13)             endif
14)         end
15)         if $(pseudo = true)$ then $L$.closure $\leftarrow \min_{\subseteq}(\{C \in FC_{j>i} \mid L \subseteq C\})$;
16)     end
17) end
18) forall $P \in \bigcup_{i=1}^{n} FP_i$ do $\mathcal{L} \leftarrow \mathcal{L} \cup \{P \rightarrow (P.\text{closure}\setminus P)\}$;
19) $IP \leftarrow \emptyset$;
20) forall $L \in MI$ do $IP \leftarrow IP \cup \{\mathcal{L}^*\text{-closure}(I)\}$;
21) $IP \leftarrow \min_{\subseteq} IP$;
```

Algorithm 9 Function $\mathcal{L}^*$-closure reads X and returns its $\mathcal{L}^*$-closure $\mathcal{L}^*(X)$.

```
1)  $Y \leftarrow X$;
2)  for $(i \leftarrow 1; i = n; i++)$ do $i$.used$\leftarrow$ false;
3)  repeat
4)      changed $\leftarrow$ false;
5)      If Subsets$(IP, Y) \neq \emptyset$ then begin $Y \leftarrow M$; changed$\leftarrow$ true end
6)      else for $(i \leftarrow 1; i \leq n; i++)$ do
7)              if $X_i \subset Y$ then begin $Y \leftarrow Y \cup Y_i$; changed$\leftarrow$ true end
8)  until not changed;
9)  return Y
```

k, containing the frequent closed itemsets and their support. The output of the algorithm is the Luxenburger basis for approximate association rules LB.

The algorithm iteratively considers all frequent closed itemsets $L \in FC_i$ for $2 \leq i \leq k$. It determines which frequent closed itemsets $L' \in \bigcup_{j<i} FC_j$ are covered by L and generates association rules of the form $L' \rightarrow L \setminus L'$ that have sufficient confidence. During the i^{th} iteration, each itemset L in FC_i is considered (steps 3 to 13). For each set FC_j, $1 \leq j < i$, a set S_j containing all frequent closed j-itemsets in FC_j that are subsets of L is created (step 4). Then, all these subsets of L are considered in decreasing order of their sizes (steps 5 to 12). For each of these subsets $L' \in S_j$, the confidence of the approximate association rule $r := L' \rightarrow L \setminus L'$ is computed (step 7). If the confidence of r is sufficient, r is inserted into LB (step 9) and all subsets L'' of L' are removed from S_l, for $l < j$ (step 10). At the end of the algorithm, the set LB contains all rules of the Luxenburger basis for approximate association rules. The proof of the correctness of the algorithm is given in [28].

Algorithm 10 Generating the Luxenburger basis with Gen-LB.

```
1)  $LB \leftarrow \{\}$;
2)  for ($i \leftarrow 2$; $i \leq k$; $i$++) do begin
3)      forall $L \in FC_i$ do begin
4)          for ($j \leftarrow 0; J < i; j++$) do $S_j \leftarrow$ Subsets($FC_j, L$);
5)          for ($j \leftarrow i-1; J \geq 1; j--$) do begin
6)              forall $L' \in S_j$ do begin
7)                  $conf \leftarrow L$.support / $L'$.support;
8)                  if ($conf \geq minconf$)
9)                      then $LB \leftarrow LB \cup \{(L' \rightarrow (L \setminus L'), L$.support, $conf)\}$;
10)                 for ($l \leftarrow j; l \geq 1; l--$) do $S_l \leftarrow S_l \setminus$ Subsets($S_l, L'$);
11)             end
12)         end
13)     end
14) end
```

4.6.4 Experimental Results

We have preformed several experiments on synthetic and real data. The characteristics of the datasets used in the experiments are given in Table 4.1. These datasets are the T10I4D100K synthetic dataset that mimics market basket data,[10] the C20D10K and the C73D10K census datasets from the PUMS sample file,[11] and the MUSHROOMS dataset describing mushroom characteristics.[12] In all experiments, we attempted to choose significant minimum support and confidence threshold values. We varied these thresholds and, for each couple of values, we analyzed rules extracted in the bases.

Table 4.1: Datasets.

Name	Number of objects	Average size of objects	Number of items
T10I4D100K	100,000	10	1,000
MUSHROOMS	8,416	23	127
C20D10K	10,000	20	386
C73D10K	10,000	73	2,177

Number of Rules. Table 4.2 compares the size of the Duquenne-Guigues basis for exact rules with the number of all exact association rules, and the size of the Luxenburger basis for approximate rules with the number of all approximate rules. In the case of weakly correlated data (T10I4D100K), no exact rule is generated. The reason is that in such data all frequent itemsets are frequent closed itemsets. However, the Luxenburger basis is relatively small compared to the number of all rules, since only immediate neighbors with respect to the subset order (and not arbitrary pairs of sets) are considered. In the case of strongly correlated data (MUSHROOMS, C20D10K and C73D10K), the ratio between the size of the bases to the number of all rules which hold is much smaller than in the weekly correlated case, because here only few of the frequent itemsets are closed and have to be considered.

[10]http://www.almaden.ibm.com/cs/quest/syndata.html
[11]ftp://ftp2.cc.ukans.edu/pub/ippr/census/pums/pums90ks.zip
[12]ftp://ftp.ics.uci.edu/~cmerz/mldb.tar.Z

Table 4.2: Number of exact and approximate association rules compared with the number of rules in the Duquenne-Guigues and Luxenburger bases.

Dataset (Minsupp)	Exact rules	D.-G. basis	Minconf	Approximate rules	Luxenburger basis
T10I4D100K (0.5%)	0	0	90%	16,269	3,511
			70%	20,419	4,004
			50%	21,686	4,191
			30%	22,952	4,519
Mushrooms (30%)	7,476	69	90%	12,911	563
			70%	37,671	968
			50%	56,703	1,169
			30%	71,412	1,260
C20D10K (50%)	2,277	11	90%	36,012	1,379
			70%	89,601	1,948
			50%	116,791	1,948
			30%	116,791	1,948
C73D10K (90%)	52,035	15	95%	1,606,726	4,052
			90%	2,053,896	4,089
			85%	2,053,936	4,089
			80%	2,053,936	4,089

Relative Performance. Our experiments also show that in all cases the execution time of Gen-FDG and Gen-LB are insignificantly small compared to those of the computation of the iceberg concept lattice, since both algorithms need not access the database. We can conclude that without additional computation time (compared to other approaches, like e. g. Apriori) our approach not only computes all frequent closed itemsets but also the two bases described in Section 4.6.1.

4.7 Bibliographic Notes

Association rule mining with Formal Concept Analysis is a relatively new approach. Other approaches addressing the problem of reducing large sets of association rules provide users with mechanisms for filtering rules, for instance by user defined templates [5, 22], Boolean [27, 36] or SQL-like [26] operators or by introducing further measures of "usefulness" [9]; or they attempt to minimize the number of extracted rules a priori by using information about taxonomies [18, 20, 35] or by applying statistical measures like Pearson's correlation or the χ^2-test [11]. All these approaches have in common that they lose some information.

In [6], we have presented another pair of bases which are different from those presented here. They provide rules with minimal antecedents and maximal consequents. Compared to the results presented here, they have the disadvantage of a higher total number of rules. For the approximate rules, M. Zaki has presented similar results in [45].

———(to be written)———

Part III

Knowledge Acquisition

Chapter 5

Attribute exploration

An imaginary example shall serve as an introduction to the problem: imagine a manufacturer of computer hardware, whose different products can be combined in various ways but not arbitrarily. In order to obtain a conceptual structuring of the (reasonable) configurations, we would have to examine a context the objects of which are the combinations and the attributes of which are the components. If a list of these combinations is not available, we have to draw it up. This can be done on the basis of our knowledge about the existing possibilities of combining the elements.

In this case, the starting point of concept analysis is not an explicitly stated context. Rather, we infer the context and at the same time the concept system from the **attribute logic**, i.e., from the rules concerning the combination of attributes.

This method does not only suggest itself in the example discussed above. It often becomes necessary to classify a large number of objects with respect to a relatively small number of attributes, and it is frequently useless or impracticable to write down the whole context and to apply the procedures for the determination of the concept system which were described in the previous section. In such cases, the concept lattices can be inferred from the implications between the attributes.

The question is thus: how can we determine the concept intents by means of the implications? We have seen that in order to do so, we do not need all the implications, but that a small subset of them is sufficient. So far we have only explained how these implications can be obtained from an available context. By means of the tools now on hand, however, we can also develop a method of generating sets of implications which are free of redundancies, even if the context is not or only partly available. This procedure, which is called **attribute exploration**, has proved successful in many applications. In practice, we use a computer which administers the sets of implications and is able to compute which information is still lacking. The implications are then determined *interactively*, i.e., in cooperation with the user.

The algorithm for the determination of the pseudo-intents permits a modification resulting in an interactive program: it is possible to modify the context by adding new objects, even while the generation of the list $\mathcal{L}$ of the implications is in progress. If the intents of these objects respect all implications determined so far, the computation for the new context can be continued with the results so far obtained. This is the content of the following proposition:

Corollary 42 *Let $\mathbb{K}$ be a context and let $P_1, P_2, \ldots, P_n$ be the first n pseudo-intents of $\mathbb{K}$ with respect to the lectic order. If $\mathbb{K}$ is extended by an object g the object intent g' of which respects the implications $P_i \to P_i''$, $i \in \{1, \ldots, n\}$, then $P_1, P_2, \ldots, P_n$ are also the lectically first n pseudo-intents of the extended context.*

This can be proved for example by induction on n. □

Therefore, if we have found a new pseudo-intent P, we can stop the algorithm and ask, whether the implication $P \to P''$ should be added to $\mathcal{L}$. The user can answer this question in the affirmative or add a counter-example, which must not contradict the implications he has confirmed so far. In the extreme case, the procedure can be started with a context the object set of which is empty. In this case, the user will have to enter all counter-examples, thereby creating a concept system with a given "attribute logic".

Instead of describing this program in detail, we shall demonstrate its functioning by means of an example.

5.1 The exploration algorithm

Have another look at the stem base computed in Figure 4.5 (p. 84). Some of the implications state that certain attribute combinations imply *all* other attributes. This occurs often when a certain attribute combination is *contradictory*, i.e., when these attributes do not hold together for any object. It is sometimes more suggestive to use the symbol $\perp$ for the set M of all attributes, but only if there is no object g with $g' = M$. Thus we define

$$\perp := M \qquad \text{provided} \quad M' = \emptyset.$$

With this abbreviation the stem base for the triangles is given in Figure 5.1.

{obtuse angled, right angled}	$\to$	$\perp$
{acute angled, right angled}	$\to$	$\perp$
{acute angled, obtuse angled}	$\to$	$\perp$
{equilateral}	$\to$	{isosceles, acute angled}

Figure 5.1: The stem base for the triangles.

The implications in this set are obviously true, because *they hold for all triangles*, not only for the objects of the triangles–context on page 27. This is an important information, because it shows that the concept lattice will not change if more examples are added. The triangles in the formal context are representative for the entire theory. The stem base generates the implicational theory of these five attributes for *all* triangles. Each implication which holds for all triangles follows from the stem base. For each implication that does not hold in general, there is already a counter example among the seven triangles of the formal context.

5.1.1 The idea

The exploration algorithm is based on this idea. We want to explore the possible combinations of a given attribute set. The objects under consideration however are many, or difficult to enumerate. Some examples (possibly zero) are known, they make up the context of examples. The stem base of this formal context is computed and we ask if the implications in this stem base hold in general. If so, then no further examples are necessary and the context of examples is representative for the entire implicational theory. Otherwise, there are counter examples to stem base implications (outside the context of examples). The context of counter examples then is extended and the computation is repeated.

```
Algorithm Attribute exploration
Input:   A formal context $(G, M, I)$, $M$ finite,
          and a subcontext $(E, M, J := I \cap E \times M)$.
Output:  The stem base $\mathcal{L}$ of $(G, M, I)$ and a possibly enlarged
          subcontext $(E, M, J := E \times M)$ with the same stem base.
begin
  $\mathcal{L} := \emptyset$;
  $A := \emptyset$;
  while $A \neq M$ do
  begin
    unregistered := true;
    while $A \neq A^{JJ}$ and unregistered do
      if $A^{JJ} = A^{II}$ then
      begin
        $\mathcal{L} := \mathcal{L} \cup \{A \to A^{II}\}$;
        unregistered := false;
      end
        else extend $E$ by some object $g \in G$ with $g \in A^I \setminus A^J$;
    $A :=$ next_$\mathcal{L}^\bullet$_closure($A$);
  end;
end.
```

Figure 5.2: The attribute exploration algorithm.

5.1.2 The algorithm

One of the oldest implementations of this algorithm is P. Burmeister's program ConImp, freely available under `http://www.mathematik.tu-darmstadt.de....`

5.1.3 Examples

We apply the technique to the data from Figure 2.1 (p. 21). The formal context for this diagram is given in Figure 5.3.

Do the given examples cover all possible cases? In order to find out, we apply Attribute Exploration. The formal context in Figure 5.3 is the *initial context of examples*, that is, the context (E, M, J) of the algorithm in Figure 5.2. The formal context (G, M, I) is infinite: its object set G is the set of *all* possible combinations of two squares (of equal size). This infinite context is the one we are interested in, and we would like to compute its stem base. Since we cannot access this infinite formal context directly, we start with the finite subcontext given in Figure 5.3 and extend it when necessary. The steps of the exploration algorithm are given in Figure 5.4.

At start, our list $\mathcal{L}$ of implications is empty. The first quasi closed set to consider is the empty set $\emptyset$. This set is closed in the context of examples, and is therefore ignored. We compute the next quasi closed set, using the (still empty) list $\mathcal{L}$. The result is $\{\mathrm{ce}\}$. This set is not closed in the example context, its closure is $\{\mathrm{ce}, \mathrm{pa}, \mathrm{cv}, \mathrm{cs}\}$. In other words, the implication $\mathrm{ce} \to \mathrm{pa}, \mathrm{cv}, \mathrm{cs}$ holds for all examples considered so far. Obviously, this implication holds in general, because two squares that share a common edge must always be parallel, must share a vertex and a line segment. We therefore include this implication in $\mathcal{L}$.

The next quasi closed set is $\{\mathrm{cs}\}$ and again, this set is not closed in the context of examples. Its closure is $\{\mathrm{cs}, \mathrm{pa}\}$. We observe that this is necessarily so for all possible examples and include the implication $\mathrm{cs} \to \mathrm{pa}$ in $\mathcal{L}$.

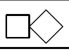
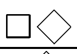
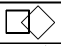
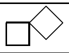
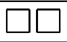
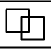
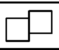
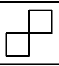
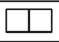
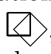
	di	ov	pa	cv	cs	ce
	disjoint	overlap	parallel	common vertex	common segment	common edge
						
	×					
		×				
				×		
	×		×			
		×	×			
			×		×	
			×	×		
			×	×	×	×
		×	×	×	×	×

Figure 5.3: The formal context for Figure 2.1

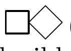
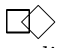
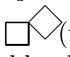
The next four quasi closed sets are all closed. They are ignored. Then we discover that we should include the implication pa, cv, cs → ce in the list $\mathcal{L}$. This implication holds in general, because two squares of equal size that have a common vertex and a common line segment must share an edge. It is somewhat irritating that the premise of this implication contains the assumption pa, which is unnecessary because it follows from cs. This is a typical feature of pseudo closed sets, they tend to be "large".

After two more closed sets we obtain the quasi closed set $\{\text{ov}, \text{cv}\}$, the closure of which in the context of examples is $\{\text{ov}, \text{pa}, \text{cv}, \text{cs}, \text{ce}\}$. This suggests the implication ov, cv → pa, cs, ce, which does *not* hold in general. Here is a counter example: , with attributes ov, cv only. It seems that this possibility had been overlooked when the data was collected. We extend the formal context of examples by this new object and continue our work with this extended context. Now, the quasi closed set $\{\text{ov}, \text{cv}\}$ is closed, and we may continue.

The next situation that requires some action is when we meet the quasi closed set $\{\text{ov}, \text{pa}, \text{cs}\}$, the closure of which in the (extended) context of examples contains cv. Again we find that this is not an instance of a generally valid implication, because we can give another new example: . In the extended context, the set $\{\text{ov}, \text{pa}, \text{cs}\}$ is closed.

The next quasi closed set leads to a new entry in $\mathcal{L}$: If two squares (of equal size) overlap, are parallel, and have a common vertex, then they must be equal.

Continuing with the algorithm, we find some more generally valid implications, all of which express that two squares cannot simultaneously be disjoint and have a vertex, a segment, or an edge in common. Adding these to $\mathcal{L}$, the algorithm terminates.

The stem base is the set of the seven implications given in the last column of figure 5.4. The context of examples now has 11 objects, because we have added the two examples in Figure 5.5. Three of these examples are, however, dispensable, because the are reducible and therefore can be omitted without effect on the implications. These are  (which was reducible from the beginning),  and (which became reducible when the new examples were added). These dispensable objects are omitted in the concept lattice in displayed in Figure 5.6. This lattice is our final result: its implicational theory is the same as the general implicational theory, i.e., the theory for *all* possible placements of two squares of equal size. In other words: this concept lattice has the same stem base as the infinite context (G, M, I).

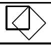
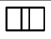
Quasi intent A	closed? $A = A^{JJ}$?	general? $A^{JJ} = A^{II}$?	action	new object or implication
$\emptyset$	yes		next q	
$\{\mathrm{ce}\}$	no	yes	new imp	ce$\rightarrow$ pa, cv, cs
$\{\mathrm{cs}\}$	no	yes	new imp	cs$\rightarrow$ pa
$\{\mathrm{cv}\}$	yes		next q	
$\{\mathrm{pa}\}$	yes		next q	
$\{\mathrm{pa}, \mathrm{cs}\}$	yes		next q	
$\{\mathrm{pa}, \mathrm{cv}\}$	yes		next q	
$\{\mathrm{pa}, \mathrm{cv}, \mathrm{cs}\}$	no	yes	new imp	pa, cv, cs$\rightarrow$ ce
$\{\mathrm{pa}, \mathrm{cv}, \mathrm{cs}, \mathrm{ce}\}$	yes		next q	
$\{\mathrm{ov}\}$	yes		next q	
$\{\mathrm{ov}, \mathrm{cv}\}$	no	no	new obj	
$\{\mathrm{ov}, \mathrm{cv}\}$	yes		next q	
$\{\mathrm{ov}, \mathrm{pa}\}$	yes		next q	
$\{\mathrm{ov}, \mathrm{pa}, \mathrm{cs}\}$	no	no	new obj	
$\{\mathrm{ov}, \mathrm{pa}, \mathrm{cs}\}$	yes			
$\{\mathrm{ov}, \mathrm{pa}, \mathrm{cv}\}$	no	yes	new imp	ov, pa, cv$\rightarrow$ cs, ce
$\{\mathrm{ov}, \mathrm{pa}, \mathrm{cv}, \mathrm{cs}, \mathrm{ce}\}$	yes		next q	
$\{\mathrm{di}\}$	yes		next q	
$\{\mathrm{di}, \mathrm{cv}\}$	no	yes	new imp	di, cv$\rightarrow \perp$
$\{\mathrm{di}, \mathrm{pa}, \mathrm{cs}\}$	no	yes	new imp	di, pa, cs$\rightarrow \perp$
$\{\mathrm{di}, \mathrm{ov}\}$	no	yes	new imp	di, ov$\rightarrow \perp$
M	yes		end.	

Figure 5.4: Steps in the exploration algorithm

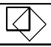
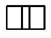
	di	ov	pa	cv	cs	ce
	disjoint	overlap	parallel	common vertex	common segment	common edge
		×		×		
		×	×		×	

Figure 5.5: Additional objects for Figure 5.3

5.1.4 Refined queries

The interactive part of the Attribute Exploration algorithm suggests an implication and asks for either a confirmation or a counter example. In a concrete application, such a question may be difficult to answer. It is possible to give more detailed information supporting the user's decision.

We have already discussed in Subsection 4.3.4 (p. 81) how to shorten implications. Moreover, not all attribute combinations are possible for counter examples,

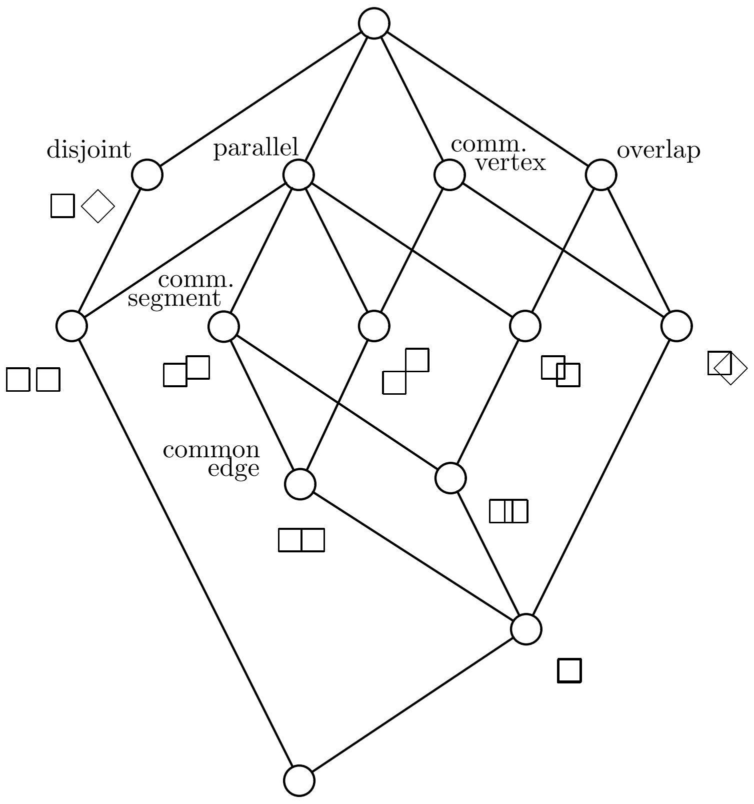


Figure 5.6: The implications of this concept lattice are valid in general.

because each counter example of course has to respect the implications already given.

Thus as a service for the interactive user, we might offer information about short premises, short conclusions and possible counter examples, whenever a question is asked. In fact, Burmeister's implementation within ConImp offers aspects of all three possibilities, and these are often helpful.

Chapter 6

Rule exploration

6.1 Predicates and relational contexts

6.1.1 n-ary relations

6.1.2 Relational structures and relational contexts

6.1.3 Power context families

6.2 Rules

6.2.1 Horn formulae

6.2.2 Complexity of Horn inference

6.2.3 Prolog

6.2.4 Relational contexts as models of Horn formulae

6.3 Generalizing attribute exploration to Horn formulae

6.3.1 Propositionalization

6.3.2 Variable permutations induce context symmetries

6.3.3 Rule exploration

6.3.4 An example

6.3.5 Hammer's complexity result on Horn bases

Chapter 7

Attribute Exploration with Background knowledge

7.1 Demand for a non-implicational background

7.1.1 The example of the drivers' exam

7.2 Implication inference with background clauses

7.2.1 Semantic inference

7.2.2 Inference rules

7.2.3 Inference is hard

7.3 Making inference feasible

7.3.1 Pseudo models

7.3.2 Cumulated clauses

7.3.3 The "basis" theorem

7.3.4 Finding pseudo models

7.4 Implication inference with background clauses: Lintime theorem

7.5 Partially given examples. The attribute exploration algorithm

7.6 Many valued attribute exploration. Logical scaling

7.7 Merging two explored contexts

Bibliography

[AIS93] R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. *Proc. SIGMOD Conf.*, 1993, 207–216

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. VLDB Conf.*, 1994, 478–499 (Expanded version in IBM Report RJ9839)

[AN68] A. Arnauld, P. Nicole: *La logique ou l'art de penser — contenant, outre les règles communes, plusieurs observations nouvelles, propres à former le jugement.* Ch. Saveux, Paris 1668

[BPTSL00] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, L. Lakhal: Mining Minimal Non-Redundant Association Rules Using Frequent Closed Itemsets. In: J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.–K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, P. J. Stuckey (eds.): *Computational Logic — CL.* Proc. 1st Intl. Conf. on CL (6th Intl. Conf. on Database Systems). LNAI **1861**, Springer, Heidelberg 2000, 972–986

[BTPSL00] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, L. Lakhal: Mining Frequent Patterns with Counting Inference. *SIGKDD Explorations* **2**(2), Special Issue on Scalable Algorithms, 2000, 71–80

[Ba98] R. J. Bayardo: Efficiently Mining Long Patterns from Databases. *Proc. SIGMOD '98*, 1998, 85–93

[BSWWZ00] K. Becker, G. Stumme, R. Wille, U. Wille, M. Zickwolff: Conceptual Information Systems Discussed Through an IT-Security Tool. In: R. Dieng, O. Corby (eds.): *Knowledge Engineering and Knowledge Management. Methods, Models, and Tools.* Proc. EKAW '00. LNAI **1937**, Springer, Heidelberg 2000, 352–365

[CR93] C. Carpineto, G. Romano: GALOIS: An Order-Theoretic Approach to Conceptual Clustering. *Machine Learning.* Proc. ICML 1993, Morgan Kaufmann Prublishers 1993, 33–40

[CS00] R. Cole, G. Stumme: CEM – A Conceptual Email Manager. In: B. Ganter, G. W. Mineau (eds.): *Conceptual Structures: Logical, Linguistic, and Computational Issues.* Proc. ICCS '00. LNAI **1867**, Springer, Heidelberg 2000, 438–452

[DDJL96] H. Dicky, C. Dony, M. Huchard, T. Libourel: On automatic class insertion with overloading. *OOPSLA 1996*, 251–267

[GR91] B. Ganter, K. Reuter: Finding all closed sets: A general approach. *Order.* Kluwer Academic Publishers, 1991, 283–290

[GW99] B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations.* Springer, Heidelberg 1999

[GMMMAC98] R. Godin, H. Mili, G. Mineau, R. Missaoui, A. Arfi, T. Chau: Design of class hierarchies based on concept (Galois) lattices. *TAPOS* **4**(2), 1998, 117–134

[GM94] R. Godin, R. Missaoui: An incremental concept formation approach for learning from databases. *TCS* **133**(2): 387–419 (1994)

[Gr94] T. Gruber: Towards principles for the design of ontologies used for knowledge sharing. *Intl. J. of Human and Computer Studies* **46**(2/3), 1997, 293–310

[HSWW00] J. Hereth, G. Stumme, U. Wille, R. Wille: Conceptual Knowledge Discovery and Data Analysis. In: B. Ganter, G. Mineau (eds.): *Conceptual Structures: Logical, Linguistic, and Computational Structures.* Proc. ICCS 2000. LNAI **1867**, Springer, Heidelberg 2000, 421–437

[HKPT99] Y. Huhtala, J. Kärkkäinen, P. Porkka, H. Toivonen: TANE: an efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal* **42**(2), 1999, 100–111

[KM95] J. Kivinen, H. Mannila: Approximate inference of functional dependencies from relations. *TCS* **149**(1), 1995, 129–149

[KS94] M. Krone, G. Snelting: On the inference of configuration structures from source code. *Proc. 16th Intl. Conference on Software Engineering*, May 1994, IEEE Comp. Soc. Press, 49–57

[Li97] Ch. Lindig: Concepts. ftp://ftp.ips.cs.tu-bs.de/pub/local/softech/misc/concepts-0.3d.tar.gz, 1997. (Open Source implementation of concept analysis in C)

[LPL00] S. Lopes, J.–M. Petit, L. Lakhal: Efficient discovery of functional dependencies and Amstrong relations. *Proc. EDBT 2000.* LNCS **1777**. Springer, Heidelberg 2000, 350–364

[Lu91] M. Luxenburger: Implications partielles dans un contexte. *Mathématiques, Informatique et Sciences Humaines* **29**(113), 1991, 35–55

[MW97] K. Mackensen, U. Wille: Qualitative Text Analysis Supported by Conceptual Data Systems. *Quality and Quantity: Internatinal Journal of Methodology* **2**(33), 1999, 135–156

[MaS00] A. Mädche, S. Staab: Mining Ontologies from Text. In: R.Dieng, O Corby (eds.): *Knowledge Engineering and Knowledge Management. Methods, Models, and Tools.* Proc. EKAW '00. LNAI **1937**, Springer, Heidelberg 2000, 189–202

[MT97] H. Mannila, H. Toivonen: Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery* **1**(3): 241-258 (1997)

[Mi80] R. S. Michalski: Knowledge acquisition through conceptual clustering: a theoretical framework and an algorithm for partitioning data into conjunctive concepts. *Policy Analysis and Information Systems* **4**(3), 1980, 219–244

[MG95] G. Mineau, G., R. Godin: Automatic Structuring of Knowledge Bases by Conceptual Clustering. *IEEE Transactions on Knowledge and Data Engineering* **7**(5), 1995, 824–829

[MiS89] M. Missikoff, M. Scholl: An algorithm for insertion into a lattice: application to type classification. *Proc. 3rd Intl. Conf. FODO 1989.* LNCS **367**, Springer, Heidelberg 1989, 64–82

[NR99] L. Nourine, O. Raynaud: A fast algorithm for building lattices. *Information Processing Letters* **71**, 1999, 199–204

[PBTL98] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Pruning Closed Itemset Lattices for Association Rules. *14èmes Journées Bases de Données Avancées (BDA'98)*, Hammamet, Tunisia, 26–30 October 1998

[PBTL99a] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Efficient mining of association rules using closed itemset lattices. *Journal of Information Systems*, **24**(1), 1999, 25–46

[PBTL99b] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Discovering frequent closed itemsets for association rules. *Proc. ICDT'99.* LNCS **1540**. Springer, Heidelberg 1999, 398–416

[PHM00] J. Pei, J. Han, R. Mao: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery 2000*, 21–30

[SSVWW93] P. Scheich, M. Skorsky, F. Vogt, C. Wachter, R. Wille: Conceptual Data Systems. In: O. Opitz, B. Lausen, R. Klar (eds.): *Information and Classification.* Springer, Berlin-Heidelberg 1993, 72–84

[SS98] I. Schmitt, G. Saake: Merging inheritance hierarchies for database integration. *Proc. 3rd IFCIS Intl. Conf. on Cooperative Information Systems*, New York City, Nework, USA, August 20-22, 1998, 122–131

[ST98] G. Snelting, F. Tip: Reengineering class hierarchies using concept analysis. *Proc. ACM SIGSOFT Symposium on the Foundations of Software Engineering*, November 1998, 99–110

[StrW93] S. Strahringer, R. Wille: Conceptual clustering via convex-ordinal structures. In: O. Opitz, B. Lausen, R. Klar (eds.): *Information and Classification.* Springer, Berlin-Heidelberg 1993, 85–98

[SM01] G. Stumme, A. Mädche: FCA–Merge: Bottom-Up Merging of Ontologies. *Proc. 17th Intl. Conf. on Artificial Intelligence (IJCAI '01).* Seattle, WA, USA, 2001, 225–230

[STBL01] G. Stumme, R. Taouil, Y. Bastide, L. Lakhal: Conceptual Clustering with Iceberg Concept Lattices. *Proc. GI–Fachgruppentreffen Maschinelles Lernen '01.* Universität Dortmund **763**, Oktober 2001

[STBPL00] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal: Fast computation of concept lattices using data mining techniques. *Proc. 7th Intl. Workshop on Knowledge Representation Meets Databases*, Berlin, 21–22. August 2000. CEUR-Workshop Proceeding. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/

[1] G. Stumme, R. Taouil, Y. Bastide, N. Pasqier, L. Lakhal: Computing Iceberg Concept Lattices with Titanic. *J. on Knowledge and Data Engineering* **42**(2), 2002, 189–222

[STBPL01] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal: Intelligent Structuring and Reducing of Association Rules with Formal Concept Analysis. In: F. Baader. G. Brewker, T. Eiter (eds.): *KI 2001: Advances in Artificial Intelligence.* Proc. KI 2001. LNAI **2174**, Springer, Heidelberg 2001, 335–350

[SWW98] G. Stumme, R. Wille, U. Wille: Conceptual Knowledge Discovery in Databases Using Formal Concept Analysis Methods. In: J. M. Żytkow, M. Quafofou (eds.): *Principles of Data Mining and Knowledge Discovery.* Proc. 2nd European Symposium on PKDD '98, LNAI **1510**, Springer, Heidelberg 1998, 450–458

[SW00] G. Stumme, R. Wille (eds.): *Begriffliche Wissensverarbeitung – Methoden und Anwendungen.* Springer, Heidelberg 2000

[TPBL00] R. Taouil, N. Pasquier, Y. Bastide, L. Lakhal: Mining Bases for Association Rules Using Closed Sets. *Proc. 16th Intl. Conf. ICDE 2000*, San Diego, CA, US, February 2000, 307

[Vo95] F. Vogt, R. Wille: TOSCANA – A graphical tool for analyzing and exploring data. LNCS **894**, Springer, Heidelberg 1995, 226–233

[WTL97] K. Waiyamai, R. Taouil, L. Lakhal: Towards an object database approach for managing concept lattices. *Proc. 16th Intl. Conf. on Conceptual Modeling*, LNCS **1331**, Springer, Heidelberg 1997, 299–312

[Wi82] R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.). *Ordered sets.* Reidel, Dordrecht–Boston 1982, 445–470

[Wi84] R. Wille: Line diagrams of hierarchical concept systems. *Int. Classif.* **11**, 1984, 77–86

[Wi92] R. Wille: Concept lattices and conceptual knowledge systems. *Computers & Mathematics with Applications* **23**, 1992, 493–515

[WMJ00] S. Wrobel, K. Morik, T. Joachims: Maschinelles Lernen und Data Mining. In: G. Görz, C.–R. Rollinger, J. Schneeberger (eds.): *Handbuch der Künstlichen Intelligenz.* 3. Auflage. Oldenbourg, München–Wien 2000, 517–597

[YLBC96] A. Yahia, L. Lakhal, J. P. Bordat, R. Cicchetti: iO2: An algorithmic method for building inheritance graphs in object database design. *Proc. 15th Intl. Conf. on Conceptual Modeling.* LNCS **1157**, Springer, Heidelberg 1996, 422–437

[2] R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. *Proc. SIGMOD Conf.*, 1993, 207–216

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. VLDB Conf.*, 1994, 478–499 (Expanded version in IBM Report RJ9839)

[4] F. Baader, R. Molitor: Building and structuring Description Logic knowledge bases using least common subsumers and Concept Analysis. In: B. Ganter, G. W. Mineau (eds.): *Conceptual Structures: Logical, Linguistic, and Computational Issues.* Proc. ICCS 2000. LNAI **1867**, Springer, Heidelberg 2000, 292–305

[5] E. Baralis and G. Psaila. Designing templates for mining association rules. *Journal of Intelligent Information Systems* 9(1), 1997, 7–32

[6] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, L. Lakhal: Mining minimal non-redundant association rules using frequent closed itemsets. In: J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.–K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, P. J. Stuckey (Eds.): *Computational Logic — CL 2000.* Proc. 1st Intl. Conf. on CL (6th Intl. Conf. on Database Systems). LNAI **1861**, Springer, Heidelberg 2000, 972–986

[7] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, L. Lakhal: Mining Frequent Patterns with Counting Inference. *SIGKDD Explorations* **2**(2), Special Issue on Scalable Algorithms, 2000, 66–75

[8] R. J. Bayardo. Efficiently mining long patterns from databases. *Proc. SIGMOD Conf.*, 1998, 85–93

[9] R. J. Bayardo, R. Agrawal, D. Gunopulos. Constraint-based rule mining in large, dense databases. *Proc. ICDE Conf.*, 1999, 188–197

[10] K. Becker, G. Stumme, R. Wille, U. Wille, M. Zickwolff: Conceptual Information Systems discussed through an IT-security tool. In: R. Dieng, O. Corby (Eds.): *Knowledge Engineering and Knowledge Management. Methods, Models, and Tools.* Proc. EKAW '00. LNAI **1937**, Springer, Heidelberg 2000, 352–365

[11] S. Brin, R. Motwani, C. Silverstein: Beyond market baskets: Generalizing association rules to correlation. *Proc. SIGMOD Conf.*, 1997, 265–276

[12] S. Brin, R. Motwani, J. D. Ullman, S. Tsur: Dynamic itemset counting and implication rules for market basket data. *Proc. SIGMOD Conf.*, 1997, 255–264

[13] V. Duquenne, J.-L. Guigues: Famille minimale d'implication informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines* 24(95), 1986, 5–18

[14] U. M. Fayyad, G. Piatetsky–Shapiro, P. Smyth, R. Uthurusamy (eds.): Advances in knowledge discovery and data mining. AAAI Press, Cambridge 1996

[15] B. Ganter, K. Reuter: Finding all closed sets: A general approach. *Order*. Kluwer Academic Publishers, 1991, 283–290

[16] B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical Foundations.* Springer, Heidelberg 1999

[17] A. Grosskopf and G. Harras: Eine TOSCANA-Anwendung für Sprechaktverben des Deutschen. In: G. Stumme and R. Wille (eds.), Begriffliche Wissensverarbeitung: Methoden und Anwendungen. Springer, Berlin-Heidelberg-New York 2000.

[18] J. Han, Y. Fu: Discovery of multiple-level association rules from large databases. *Proc. VLDB Conf.*, 1995, 420–431 1995.

[19] J. Hereth, G. Stumme, U. Wille, R. Wille: Conceptual Knowledge Discovery and Data Analysis. In: B. Ganter, G. W. Mineau (eds.): *Conceptual Structures: Logical, Linguistic, and Computational Issues.* Proc. ICCS 2000. LNAI **1867**, Springer, Heidelberg 2000, 421–437

[20] J. Hipp, A. Myka, R. Wirth, U. Güntzer: A new algorithm for faster mining of generalized association rules. LNAI **1510**, Springer, Heidelberg 1998

[21] U. Kaufmann: Begriffliche Analyse über Flugereignisse – Implementierung eines Erkundungs- und Analysesystems mit TOSCANA. Diplomarbeit, FB4, TU Darmstadt 1996.

[22] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, A. I. Verkamo: Finding interesting rules from large sets of discovered association rules. *Proc. CIKM Conf.*, 1994, 401–407

[23] M. Luxenburger: Implications partielles dans un contexte. *Mathématiques, Informatique et Sciences Humaines*, 29(113), 1991, 35–55

[24] M. Luxenburger: Partial implications. Part I of *Implikationen, Abhängigkeiten und Galois Abbildungen.* PhD thesis, TU Darmstadt. Shaker, Aachen 1993

[25] H. Mannila, H. Toivonen: Multiple uses of frequent sets and condensed representations (Extended abstract). *Proc. KDD 1996*, 189–194

[26] R. Meo, G. Psaila, S. Ceri: A new SQL-like operator for mining association rules. *Proc. VLDB Conf.*, 1996, 122–133

[27] R. T. Ng, V. S. Lakshmanan, J. Han, A. Pang: Exploratory mining and pruning optimizations of constrained association rules. *Proc. SIGMOD Conf.*, 1998, 13–24

[28] N. Pasquier: *Data Mining : algorithmes d'extraction et de réduction des règles d'association dans les bases de données.* PhD thesis. Université Blaise Pascal, Clermont-Ferrand II, 2000

[29] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Pruning closed itemset lattices for association rules. *Proc. 14ièmes Journées Bases de Données Avancées (BDA '98)*, Hammamet, Tunisie, 177–196

[30] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Closed set based discovery of small covers for association rules. *Proc. 15èmes Journées Bases de Données Avancées*, Bordeaux, France, 25–27 October 1999, 361–381

[31] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Discovering frequent closed itemsets for association rules. *Proc. ICDT Conf.*, 1999, 398–416

[32] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal: Efficient mining of association rules using closed itemset lattices. *Journal of Information Systems*, 24(1), 1999, 25–46

[33] J. Pei, J. Han, R. Mao: CLOSET: An efficient algorithm for mining frequent closed itemsets. *Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery 2000*, 21–30

[34] M. Roth-Hintz, M. Mieth, T. Wetter, S. Strahringer, B. Groh, R. Wille: Investgating SNOMED by Formal Concept Analysis. Preprint, FB4, TU Darmstadt 1997.

[35] R. Srikant, R. Agrawal: Mining generalized association rules. *Proc. VLDB Conf.*, 1995, 407–419

[36] R. Srikant, Q. Vu, R. Agrawal: Mining association rules with item constraints. *Proc. KDD Conf.*, 1997, 67–73

[37] G. Stumme, R. Wille, U. Wille: Conceptual Knowledge Discovery in Databases Using Formal Concept Analysis Methods. In: J. M. Żytkow, M. Quafofou (eds.): *Principles of Data Mining and Knowledge Discovery.* Proc. 2nd European Symposium on PKDD '98, LNAI **1510**, Springer, Heidelberg 1998, 450–458

[38] G. Stumme: *Conceptual Knowledge Discovery with Frequent Concept Lattices.* FB4-Preprint **2043**, TU Darmstadt 1999

[39] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal: Fast Computation of Concept Lattices Using Data Mining Techniques. *Proc. 7th Intl. Workshop on Knowledge Representation Meets Databases*, Berlin, 21–22. August 2000. CEUR-Workshop Proceeding. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/

[40] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, L. Lakhal: Computing Iceberg Concept Lattices with TITANIC. *J. on Knowledge and Data Engineering.* (submitted)

[41] F. Vogt, R. Wille: TOSCANA – A graphical tool for analyzing and exploring data. LNCS **894**, Springer, Heidelberg 1995, 226–233

[42] R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.). *Ordered sets.* Reidel, Dordrecht–Boston 1982, 445–470

[43] M. J. Zaki, M. Ogihara: Theoretical Foundations of Association Rules, *3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, Seattle, WA, June 1998, 7:1–7:8

[44] M. J. Zaki, C.–J. Hsiao: ChARM: An efficient algorithm for closed association rule mining. Technical Report 99–10, Computer Science Dept., Rensselaer Polytechnic Institute, October 1999

[45] M. J. Zaki: Generating non-redundant association rules. *Proc. KDD 2000.* 34–43