

Part III

Knowledge Acquisition

Chapter 6

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, \dots, P_n be the first n pseudo-intents of \mathbb{K} with respect to the lexic order. If \mathbb{K} is extended by an object g the object intent g' of which respects the implications $P_i \rightarrow P_i'', i \in \{1, \dots, n\}$, then P_1, P_2, \dots, P_n are also the lexically 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 $P \rightarrow P''$ 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.

6.1 The exploration algorithm

Have another look at the stem base computed in Figure 5.5 (p. 86). 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

$$\perp := M \quad \text{provided} \quad M' = \emptyset.$$

With this abbreviation the stem base for the triangles is given in Figure 6.1.

{obtuse angled, right angled}	\rightarrow	\perp
{acute angled, right angled}	\rightarrow	\perp
{acute angled, obtuse angled}	\rightarrow	\perp
{equilateral}	\rightarrow	{isocetes, acute angled}

Figure 6.1: 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.

6.1.1 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.

<p>Algorithm ATTRIBUTE EXPLORATION</p> <p>Input: A formal context (G, M, I), M finite, and a subcontext $(E, M, J := I \cap E \times M)$.</p> <p>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.</p> <p>begin</p> <p style="padding-left: 2em;">$\mathcal{L} := \emptyset;$</p> <p style="padding-left: 2em;">$A := \emptyset;$</p> <p style="padding-left: 2em;">while $A \neq M$ do</p> <p style="padding-left: 4em;">begin</p> <p style="padding-left: 6em;">unregistered := true;</p> <p style="padding-left: 6em;">while $A \neq A^{JJ}$ and unregistered do</p> <p style="padding-left: 8em;">if $A^{JJ} = A^{II}$ then</p> <p style="padding-left: 10em;">begin</p> <p style="padding-left: 12em;">$\mathcal{L} := \mathcal{L} \cup \{A \rightarrow A^{II}\};$</p> <p style="padding-left: 12em;">unregistered := false;</p> <p style="padding-left: 10em;">end</p> <p style="padding-left: 8em;">else extend E by some object $g \in G$ with $g \in A^I \setminus A^J;$</p> <p style="padding-left: 6em;">$A := \text{NEXT_}\mathcal{L}^\bullet\text{-CLOSURE}(A);$</p> <p style="padding-left: 4em;">end;</p> <p style="padding-left: 2em;">end.</p>
--

Figure 6.2: The attribute exploration algorithm.

6.1.2 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...>

6.1.3 Examples

We apply the technique to the data from Figure 2.1 (p. 21). The formal context for this diagram is given in Figure 6.3.

Do the given examples cover all possible cases? In order to find out, we apply Attribute Exploration. The formal context in Figure 6.3 is the *initial context of examples*, that is, the context (E, M, J) of the algorithm in Figure 6.2. 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 6.3 and extend it when necessary. The steps of the exploration algorithm are given in Figure 6.4.

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 $\{ce\}$. This set is not closed in the example context, its closure is $\{ce, pa, cv, cs\}$. In other words, the implication $ce \rightarrow pa, cv, cs$ 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 $\{cs\}$ and again, this set is not closed in the context of examples. Its closure is $\{cs, pa\}$. We observe that this is necessarily so for all possible examples and include the implication $cs \rightarrow pa$ in \mathcal{L} .

	di	ov	pa	cv	cs	ce
	disjoint	overlap	parallel	common vertex	common segment	common edge
	×					
		×				
				×		
	×		×			
		×	×			
			×		×	
			×	×		
			×	×	×	×
		×	×	×	×	×

Figure 6.3: 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 $pa, cv, cs \rightarrow ce$ 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 $\{ov, cv\}$, the closure of which in the context of examples is $\{ov, pa, cv, cs, ce\}$. This suggests the implication $ov, cv \rightarrow pa, cs, ce$, 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 $\{ov, cv\}$ is closed, and we may continue.

The next situation that requires some action is when we meet the quasi closed set $\{ov, pa, cs\}$, 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 $\{ov, pa, cs\}$ 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 6.4. The context of examples now has 11 objects, because we have added the two examples in Figure 6.5. Three of these examples are, however, dispensable, because they 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 6.6. 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	
{ce}	no	yes	new imp	$ce \rightarrow pa, cv, cs$
{cs}	no	yes	new imp	$cs \rightarrow pa$
{cv}	yes		next q	
{pa}	yes		next q	
{pa, cs}	yes		next q	
{pa, cv}	yes		next q	
{pa, cv, cs}	no	yes	new imp	$pa, cv, cs \rightarrow ce$
{pa, cv, cs, ce}	yes		next q	
{ov}	yes		next q	
{ov, cv}	no	no	new obj	
{ov, cv}	yes		next q	
{ov, pa}	yes		next q	
{ov, pa, cs}	no	no	new obj	
{ov, pa, cs}	yes			
{ov, pa, cv}	no	yes	new imp	$ov, pa, cv \rightarrow cs, ce$
{ov, pa, cv, cs, ce}	yes		next q	
{di}	yes		next q	
{di, cv}	no	yes	new imp	$di, cv \rightarrow \perp$
{di, pa, cs}	no	yes	new imp	$di, pa, cs \rightarrow \perp$
{di, ov}	no	yes	new imp	$di, ov \rightarrow \perp$
M	yes		end.	

Figure 6.4: Steps in the exploration algorithm

	di	ov	pa	cv	cs	ce
	disjoint	overlap	parallel	common vertex	common segment	common edge
		×		×		
		×	×		×	

Figure 6.5: Additional objects for Figure 6.3

6.1.4 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 5.3.4 (p. 83) how to shorten implications. Moreover, not all attribute combinations are possible for counter examples,

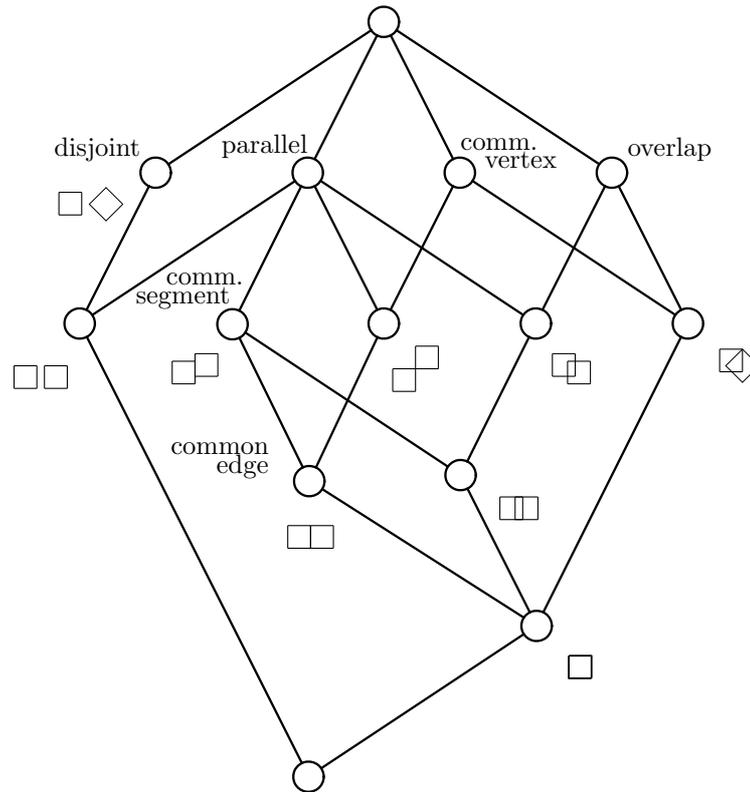


Figure 6.6: The implications of this concept lattice are valid in general.

because each counter example of course has to respect the implications already given.

Thus as a service for the interactive user, we might offer informations 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.