A Comparison of Graphs of Concept for Reverse Engineering[*]

Nicolas Anquetil
COPPE – Universidade Federal do Rio de Janeiro
C.P. 68511, Citade Universitaria
RJ, 21945-970, Brazil
(55) (21) 590-2552 x.334
nicolas@cos.ufrj.br

Abstract

To group related things together (for example to form subsystems), researchers in Reverse Engineering are looking for algorithms that create meaningful groups.

One such algorithm, Concept Analysis, received a lot of interest recently. It creates a lattice of concepts which have some advantages over the more traditional tree of clusters from clustering algorithms.

We will argue that the main interest of Concept Analysis lies in the concepts themselves and can be disconnected from the particular structure (the lattice of concepts) in which the concepts are usually arranged. We will compare Concept Analysis to various other algorithms trying to select the most important concepts contained in a set of entities.

Our main conclusion is that although it have advantages, the lattice of concepts suffer from a major drawback that other constructs do not have: it returns much more information (concepts) than what it was given in input (a set of entities describing some software system).

[*]This work is sponsored by a grant from the Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ)

Introduction

The reverse engineering community showed recently a lot of interest in an approach called Concepts Analysis [6, 11, 14]. Researchers have compared this new approach to more traditional ones, as clustering, and found it to have qualities that clustering does not exhibit. In these works, Concepts Analysis is usually associated to a structure called a lattice of concepts (or Galois lattice).

Although we agree that Concept Analysis has some interest for Reverse Engineering, we will discuss an important problem of the lattice of concepts: It usually outputs much more information (concepts) than what it was given in input. The ratio can go from thrice, up to hundreds of times, more information output; thus overwhelming the user instead of helping him to get an abstract view of a set of data.

In this paper, we denounce this fact and propose some solutions to try to extract only the most significant concepts. We will first present the notion of concept, which is central to this discussion. We will then propose various algorithms to build graphs of concepts, including the traditional Concept Analysis resulting in a lattice of concept. Finally before concluding, we propose some experimental results to help compare the algorithms previously introduced.

1 Concepts

Concepts are groups of entities, that may be described by their properties. Wille in [17] define a concept as having three components: A name, a list of attributes describing the concept (called the *intent*) and a list of entities belonging to the concept (called the *extent*). A concept can be indifferently referred to by any of these three components. More specifically this means that a concept is uniquely identified by its intent or its extent. In other words, given the intent of a concept, one can always reconstruct its extent, and vice-versa, the two are isomorphic.

Table 1. Description of some files with the routines they refer to.

File	Description
F1.c	{sizeof, malloc, realloc, free}
F2.c	{fopen, printf, fprintf, fclose, free}
F3.c	{fopen, fscanf, printf, malloc, free}

Formally, concepts are (named) couples: `(<entity-list>,<attribute-list>)`. The meaning of such a couple is that:

- All the objects in `<entity-list>` possess all the attributes in `<attribute-list>`, or it is not possible to find an entity in `<entity-list>` and an attribute in `<attribute-list>` such that the entity does not have the attribute.

`({F2.c,F3.c},{fopen,free})` is a valid couple based on the data presented in Table 1.

However, all couples are not concepts. To that effect, a couple must respects also the following requirements: Given a couple `({E},{I})` where E is the extent (list of entities) of the couple and I is the intent (list of attributes) of the couple,

- there is no entity that has all the attributes in I and is not member of E;
- there is no attribute that belongs to all entities in E and is not member of I.

The couple `({F2.c,F3.c},{fopen,free})` is not a concept because printf also belongs to F2.c and F3.c which violates the second requirement. `({F2.c,F3.c},{fopen,printf,free})` is a concept for the data set proposed in Table 1.

The two extra requirements to be a concept, insure that there is an isomorphism between the intent and the extent of the couple.

Note that one can easily define a generalization/specialization relation between such concepts. Given two concepts $C_1 = (E_1, I_1)$ and $C_2 = (E_2, I_2)$, we will say that C_1 is more general than C_2 iff:

$$I_1 \subset I_2$$

That is to say if C_2 is described with all C_1 attributes plus some other ones, then C_2 is a specialization of C_1. Because of the isomorphism between intent and extent, we have the following property:

$$I_1 \subset I_2 \Leftrightarrow E_2 \subset E_1$$

This conforms to intuition, since it is natural that a specialization of a concept (here C_2 specializes C_1) has less instances than this concept $E_2 \subset E_1$ (but more attributes).

One can define hierarchical graphs of concepts based on this relation.

In the literature, the concepts are always associated with the lattice of concepts (or Galois lattice) which is a convenient hierarchical graph to extract all the concepts from a data set. We will see that there are other hierarchical graphs which may be of interest too.

2 Graphs of Concepts

We now present different graph structures that may be used to organize the concepts extracted from a data set into a inheritance hierarchy. Given a data set, there is a finite (maximal) set of concepts that can be extracted from it. This is the set extracted by the lattice of concepts. The different other structures present different possible subsets of this maximal set of concepts.

The name of "structure" can be misleading, since the difference resides in the method (algorithm) used to extract the concepts. The structures themselves depend directly and unequivocally on the particular set of concepts extracted. They consist in a graph of all the concepts extracted, related by the inheritance relation defined in the previous section.

We will first present the lattice of concepts which contains all concepts from a given data set. This particularity makes it a reference for the other which may be considered as containing subsets of all the concepts in the lattice.

The idea that the lattice of concepts exhibit all the possible concepts contained in a set of entities is interesting because it allows to see the it as a search space where any concept extraction method will look for important concepts. This contributes to set a common base on which to compare all the methods. For example, a method's ability of abstraction could be simply measured in the percentage of concept from the lattice of concepts it accept.

In [5] we present the conclusions of some experiments in designing such methods to select the most important concepts from a lattice of concepts. One of our conclusions was that any such method should not consider the concepts independently (e.g. "a concept is important if it has more than 5 instances and less than 7 attributes"), but rather try to compare the concepts between themselves (e.g. "a concept is important if it has strictly more attributes than all its super-concepts").

We will first present the lattice of concepts, and then two methods to limit the set of extracted concepts.

2.1 Lattices of concepts

This structure proved its utility for browsing purposes [7]. This is the one usually associated with Concept Analysis. The set of concepts of the lattice and the lattice itself or presented respectively in Figure 1 and Table 2.

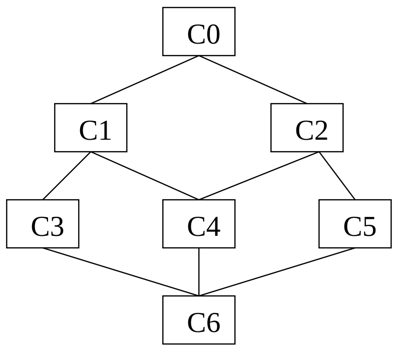

Figure 1. Lattice of concepts for the data set in Table 2.

The lattice has the interesting property that it contains all the concepts from a set of data. This, however, may also prove to be a drawback since the lattice contains usually much more concepts than the number of data it was given in input. It has absolutely no synthesis capabilities. The maximal number of concepts contained in a set of data is theoretically exponential in the number of objects in the data set (although in practice is is usually linear in the number of objects [9]. In our experiments we got from 3 to thousand of times more con-

Table 2. All concepts contained in the data set of Table 1.

Name	Extent	Intent
C0	{F1.c, F2.c, F3.c}	{free}
C1	{F1.c, F3.c}	{malloc, free}
C2	{F2.c, F3.c}	{free, fopen, printf}
C3	{F1.c}	{sizeof, malloc, realloc, free}
C4	{F3.c}	{malloc, free, fopen, printf, fscanf}
C5	{F2.c}	{free, fopen, fclose, printf, fprintf}
C6	∅	{sizeof, malloc, realloc, free, fopen, fclose, printf, fprintf, fscanf}

cepts outputs than the number of entities introduced.

2.2 Graph Based on Clustering

We have developed an alternative method to build a graph of concepts based on a hierarchical clustering method, that we will describe here However, we will not describe in length the particular hierarchical clustering algorithm used[1].

Clustering is a statistical mean that aims at gathering into coherent clusters, some set of entities. The goal to achieve is to create cluster with high internal cohesion (entities within a cluster are highly related) and low external coupling (few relations between entities in different clusters). Agglomerative hierarchical algorithms start from individual entities, gather them two by two into small clusters which are in turn gathered into larger clusters up to one final cluster containing everything. They result in a binary tree of clusters.

To get a hierarchy of concepts from this binary tree, we need to solve four problems:

- Clusters are not thought of as couples, they are usually only considered as sets of entities and have no intent.
- The tree of cluster being binary, it must represent a concept with more than two sub-concepts as an artificial hierarchy of couples, all with the same "intent" but different extents[2].
- The tree only allows simple inheritance between the couples, whereas a typical set of concepts heavily relies on multiple inheritance.
- Due to these last two problems, the couples in the tree may not respect the extra requirements for a couple to be a concept (see section §1): Because of the artificial hierarchy of couples, the sub-couples may lacks some entities in their extent. Because of the simple inheritance constraint, some couples, do not know all their sub-concepts and also lacks some entities in their extent.

In [4] we propose a solution to remedy these problems, it builds a general graph of concepts from the binary tree of clusters. This is done in three steps, the first one being presented in Table 3 (lower part) for a possible cluster set (upper part) extracted from the data in Table 1.

First we need to see clusters as couples and not only as a gathering of entities, which means we should see them as a pair intent/extent and

[1]This information may be found in [10, 15, 16]

[2]The binary tree actually contains clusters which don't have intent. They will be couples only after we solve the first problem.

Table 3. Converting clusters (left column) to concepts (right column).

Clusters	Concepts		
(=Extent)	Name	Extent	Intent
{F1.c}	C0	{F1.c}	{sizeof,malloc,realloc,free}
{F2.c}	C1	{F2.c}	{free,fopen,fclose,printf,fprintf}
{F3.c}	C2	{F3.c}	{malloc,free,fopen,printf,fscanf}
{F2.c,F3.c}	C3	{F2.c,F3.c}	{free,fopen,printf}
{F1.c,F2.c,F3.c}	C4	{F1.c,F2.c,F3.c}	{free}

not only an extent. We propose to do it simply by taking for intent of a couple, the intersection of the descriptions of all the entities in the extent of the cluster.

Note that the entities' attributes may be weighted, in that case we convert them to boolean attributes (present in the entity or not). A weight of zero indicates that the attribute is absent from the entity, otherwise it is present.

Second, we need to turn these couples into concepts, which means recomputing the extent of each couple from its intent. As a rule the couples from the previous step contain less entities in their extents that they should. The new extent is recomputed by including all entities which have all attributes of the couple's intent in their description.

We now have concepts, and there is no need to recompute intents from the new extents.

Finally, we put all these concepts into a graph structure by computing the inheritance links between the concepts.

We believe that this method has the advantage that it actually builds an abstraction of the data corpus by extracting only the most interesting concepts.

In [5], we propose an algorithm that gives similar results by mimicking hierarchical clustering inside the lattice of concepts.

2.3 Godin's Pruning Method

Godin in [8] propose his own version of a pruned lattice of concepts.

Some concepts inherit attributes from their super-concept(s) and introduce some new attributes as well. Other concepts do not introduce any new attributes, but only record a particular combination of two (or more) super-concepts through multiple inheritance. For example C4 has five attributes in Table 2, but only one is represented in Figure 2. C6 does not introduce any new attribute.

Similarly, some concepts contain all the instance of their sub-concept(s) and have instances of their own. Other concepts, only contain the union of all their sub-concepts' instances. Making an analogy with the object model, one could call these latter *abstract concepts* by opposition to the former *concrete concepts* with instances of their own. For example C2 contains two entities in Table 2, but none is represented in Figure 2.

Godin proposes to consider as non-important those abstract concepts that do not introduce any attributes of their own.

Given this new graphical representation (Figure 2), the concepts Godin proposes to eliminate are those that appear with an empty intent and empty extent (note that this is only a different representation of the same concepts, they are not actually empty).

Other pruning methods similar to this one have been proposed: [12, 13].

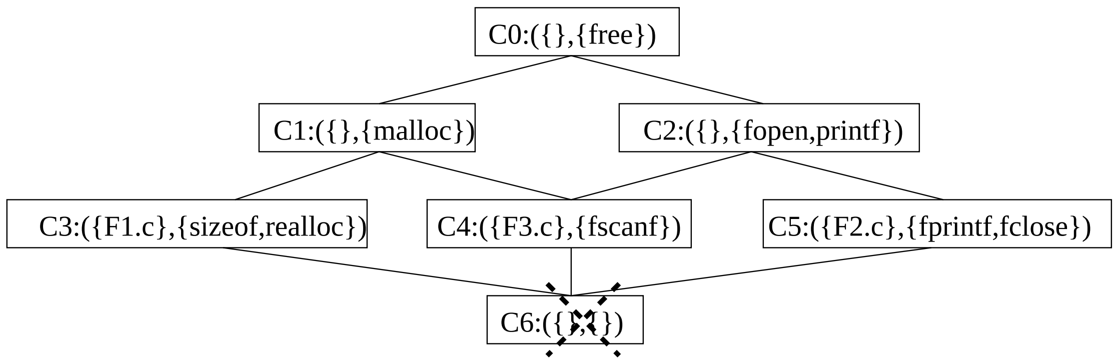

Figure 2. Illustration of Godin's pruning method. The graph is the same as in Figure 1. Compare the information represented with the actual intents and extents of the concepts in Table 2.

3 Simplifying the data set

The last two structures exhibit sub-sets of the set of concepts extracted by the lattice. They extract (a lot) less concepts that the latter. But we should also compare them on other grounds.

In [5] we compared the resistance to noise and errors in the data (see below the definition of these two terms) of the lattice of concepts and the graph based on clustering.

In the context of this experiment, we had two types of attributes, those that actually belonged to the domain and those which did not. *Noise* was defined as the presence of non-domain attribute in an entity's description. *Errors* were defined as the absence of a domain attribute in an entity's description that should have contained it, or addition of a domain attribute in an entity's description that should not have contained it.

We showed that the graph based on clustering is much more resistant to noise than the lattice of concepts (and could actually take advantage of some noise), and that the latter was more resistant to errors. One explication is that the lattice of concepts extract all the possible concepts in the data, therefore, noise produces many (erroneous) concepts, whereas clustering was intended from the beginning to deal with noise.

If we, now, consider entities which are files described by the list of routines they call (each routine is a possible attribute): We propose to say that utility routines (in C, it could be malloc, free, ...) are non-domain attributes, they are called only for implementation purposes and do not actually represent the domain. Therefore calls to these routines will be considered noise.

In the same context, errors could take the form of dead code, which would result in the entities' descriptions as calls to domain routines which should not be there. The opposite error (absence of a domain attribute in a description that should contain it, i.e. a missing routine call) would be rarer if we assume that the code is working.

This analysis points to the graph based on clustering as a better solution because it should not suffer from error in the code whereas it would provide better results when dealing with noise.

To try to improve the results of the lattice of concept, one could tried to remove the noise from the code before extracting the lattice of concepts.

3.1 Utility attributes

Van Deursen, in [6], applies the approach proposed above. His solution differs from the previous ones in that, instead of trying to simplify the set of concepts extracted, he proposes to simplify the data set before extracting concepts.

This solution has two interests:

- It can be used in conjunction with the two previous structure: First one simplifies the data set then one selects the most interesting concepts existing in this new data set. Such a combined solution should give better results.
- It introduces the issue of the description of the entities in the data set (what van Deursen tries to simplify). We will see that this issue is of the utmost importance.

Van Deursen used the lattice to find possible classes from Data Division "structures" in Cobol programs. In his work, each data is an entity and the attributes are the programs in which the data are used. Van Deursen noted that some programs are utilities that access many or all data. These introduce noise in the data set to which the lattice of concepts is extremely sensible. By removing them, he was able to clarify the data set and extract less concepts.

We identify such utility attributes by the large number of entities that possess them.

4 Some Experimental Results

We conducted a few simple experiments to compare the various methods proposed above. We will now present and comment their results.

Data are extracted from the Mosaic system. For this comparison, the kind of entity used (files, routines, structured types, etc.) has little importance. We used files. The various schema (we call them descriptive features, see [3]) used to describe these entities are[3]:

Included files: According to the files included by each entity.

Words in ident.: According to the words found in all identifiers each entity contains.

Combination: According to all routines, user defined types, global variables, files or macros referenced in each entity. Note that *Included files* is a part of *Combination*.

We will use two metrics to compare the various graphs of concept obtained:

Number of concepts: This measures the degree of abstraction a method can achieve.

Design quality: This is measured using traditional cohesion and coupling metrics. The exact formula for these two metrics is given for example in [2].

Cohesion and coupling should be computed either for a single concept (considered as a subsystem in this case) or for a partition of the entire data set. However, all methods produce graphs of concept. The numbers we will present are an average of the cohesion (or coupling) of all concepts.

One should use these results with caution[4].

Table 4 presents some information on the system studied. "Non util." are attributes which are not utilities. Following van Deursen, they were defined as attributes possessed by

[3]For more information on the descriptive features, see `http://www.site.uottawa.ca/~anquetil/Clusters/` or [1]

[4]See also [3] or [1] for a discussion of other problems associated to these two metrics.

Table 4. Some information on the data sets used. Combination, Included file and Words in ident. are three descriptive features for files. Attrib.'s "use" gives the average number of entities in which attributes appear. "# non util." gives the number of attributes which are not considered utilities (i.e. which appears in less than 20 entities.)

	Combination	Included files	Words in ident.
# entity	225		
# attribute	3059	211	3821
attrib.'s "use"	3.021	4.553	4.228
# non util.	2996	202	3781

less than a given number of entities. In our experiments, utility attributes are those which are possessed by 20 entities or more. See in the table the average "use" of the attributes; for the three descriptive features, it is below five entities possessing an attribute.

The results are given in Table 5. From these, we draw the following conclusions:

- Clearly, the Lattice of concepts may contain a great deal of concepts. In most cases, it is doubtful that a user can have any use of such a quantity of information (e.g. tens of thousand of concepts from only 225 entities).

  The pruning method proposed by Godin is more reasonable and the method based on clustering is the one with less concepts overall. This is why we say it has better abstraction capability.

- Cohesion seems very good for the lattice of concepts, however this good result is favored by the singleton concepts (which contain only one entity). Cohesion is not define for these concepts and we arbitrarily set it to 0. For the lattice of concepts, where the singleton concepts or a small portion of the total, they have few influence on the average, but for the two other methods they contribute greatly to reduce (worsen) the average cohesion. The second cohesion, excluding singleton concepts shows a much better result for the two other methods than for the lattice of concepts.

- Coupling does not change much for all experiments and is always worse (higher) for the lattice of concepts.

- Van Deursen's method to eliminate noise from the data does work. However, it does not seem to have a decisive impact on the results (for example the lattice of concepts still have many more concepts than the two other).

  Results on average cohesion seem unpredictable; there seems to be a slight increase of the cohesion without utility attributes; and no significant modification of the average coupling.

  The improvement introduced by this method does not seem as interesting as what selection in the set of concepts can provide. However, we did not try to fine-tune the threshold value used to define utility attributes ($\geq$ 20 entities possessing a utility attribute). This could improve further the results.

- The choice of descriptive feature has more impact on the lattice of concepts. For example, "Combination" is not very good (too many concepts) because each descriptive feature in the combination will produce its own set of many simplistic concepts which are probably of little interest.

5 Conclusion

We have presented and discussed various methods to extract concepts from a set of data.

Table 5. Comparison of three graphs of concepts according to different metrics, for three descriptive features.

	#ccpts	Cohesion (normal)	#singl.	Cohesion (no singl.)	Coupl.
	Combination				
Lattice of Concepts	**19971**	**0.235**	**204**	**0.237**	**0.058**
same, no util. attrib.	3787	0.256	372	0.284	0.056
Godin's pruning	**1048**	**0.237**	**197**	**0.246**	**0.049**
same, no util. attrib.	981	0.199	295	0.284	0.050
Based on Clustering	**364**	**0.058**	**189**	**0.320**	**0.049**
same, no util. attrib.	330	0.148	186	0.338	0.050
	Included files				
Lattice of Concepts	**791**	**0.200**	**125**	**0.238**	**0.054**
same, no util. attrib.	476	0.193	114	0.254	0.054
Godin's pruning	**164**	**0.076**	**111**	**0.234**	**0.048**
same, no util. attrib.	255	0.119	110	0.208	0.048
Based on Clustering	**235**	**0.125**	**113**	**0.241**	**0.049**
same, no util. attrib.	194	0.111	104	0.240	0.049
	Words in ident.				
Lattice of Concepts	**114285**	**0.172**	**415**	**0.172**	**0.053**
same, no util. attrib.	31455	0.193	372	0.181	0.051
Godin's pruning	**1893**	**0.160**	**233**	**0.183**	**0.046**
same, no util. attrib.	1851	0.119	232	0.184	0.046
Based on Clustering	**392**	**0.147**	**215**	**0.326**	**0.046**
same, no util. attrib.	380	0.111	213	0.333	0.046

One of these methods, the lattice of concepts, has recently received a lot of attention. Our main point is that although this is an interesting construct, it also have a very significant drawback for reverse engineering because it has no abstraction capability. It extracts absolutely all concepts contained in the set of data and commonly returns tens of concepts for every entity it is given in input.

Concepts (and Concept Analysis) are usually presented and studied in the context of this particular structure, but we rather propose to use them with other constructs that would make a selection in the set of all possible concepts. A new interest of the lattice of concepts would be to serve as a comparison base for all other methods.

References

[1] N. Anquetil and T. C. Lethbridge. A comparative study of clustering algorithms and abstract representations for software remodularization. submited for publication.

[2] N. Anquetil and T. C. Lethbridge. Extracting Concepts from File Names; a New File Clustering Criterion. In *International Conference on Software Engineering, ICSE'98*, pages 84–93. IEEE, IEEE Comp. Soc. Press, Apr. 1998.

[3] N. Anquetil and T. C. Lethbridge. Experiments with clustering as a software remodularization method. In *Working Conference on Reverse Engineering*, pages 235–255. IEEE, IEEE Comp. Soc. Press, Oct. 1999.

[4] N. Anquetil and J. Vaucher. Acquisition et classification de concepts pour la réutilisation. In *4ème Colloque International en informatique cognitive des organisations / 4th International Conference on Cognitive and Computer Sciences for Organizations*, pages 463–472, 1255, Carré Phillips Bureau 602, Montréal (Québec) Canada H3B 3G1, 1993. ICO, GIRICO.

[5] N. Anquetil and J. Vaucher. Extracting Hierarchical graphs of concepts from an object set : Comparison of two methods. In *Knowledge Acquisition Workshop, ICCS'94*, 1994.

[6] T. K. Arie van Deursen. Identifying object using cluster and concept analysis. In *21st International Conference on Software Engineering, ICSE'99*, pages 246–55. ACM, ACM press, may 1999.

[7] R. Godin. *L'utilisation de treillis pour l'accès aux systèmes d'information.* PhD thesis, Université de Montréal, 1986.

[8] R. Godin and H. Mili. Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. *ACM SIGplan Notices*, 28(10):394–410, 1993. OOPSLA'93 Proceedings.

[9] R. Godin, R. Missaoui, and H. Alaoui. Incremental Algorithms for Updating the Galois Lattice of a Binary Relation. Technical Report 155, Université du Québec à Montréal, septembre 1991.

[10] J. Hartigan. *Clustering Algorithms.* Mc Graw-Hill, 1975.

[11] C. Lindig and G. Snelting. Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis. In *19th International Conference on Software Engineering, ICSE'97*, pages 349–59. ACM SIGSoft, ACM Press, May 1997.

[12] G. Mineau, J. Gecsei, and R. Godin. Structuring Knowledge Bases Using Automatic Learning. In *Proceedings of the sixth International Conference on Data Engineering*, pages 274–280, 1990.

[13] G. Oosthuizen, C. Bekker, and C. Avenant. Managing Classes in Very Large Class Repositories. In *Tools'92 proceedings*, pages 625–633, 1992.

[14] M. Siff and T. Reps. Identifying modules via concept analysis. In M. J. Harrold and G. Visaggio, editors, *International Concept on Software Maintenance, ICSM'97*, pages 170–79. IEEE, IEEE Comp. Soc. Press, oct. 1997.

[15] P. H. Sneath and R. R. Sokal. *Numerical Taxonomy.* Series of books in biology. W.H. Freeman and Company, San Francisco, 1973.

[16] T. A. Wiggerts. Using clustering algorithms in legacy systems remodularization. In *Working Conference on Reverse Engineering*, pages 33–43. IEEE, IEEE Comp. Soc. Press, Oct. 1997.

[17] R. Wille. Concept Lattices and Conceptual Knowledge Systems. In F. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 493–516. 1992.