Representing Concept Lattices with Euler Diagrams ⋆

Uta Priss

Fakultät Informatik, Ostfalia University, Wolfenbüttel, Germany
www.upriss.org.uk

Abstract. The Hasse diagrams of Formal Concept Analysis (FCA) concept lattices have the disadvantages that users need to be trained in reading the diagrams and diagrams of larger lattices tend to be too cluttered to be comprehensible. This paper therefore discusses how to reduce lattices and then represent them with a specific type of Euler diagram instead of Hasse diagrams. A semi-automated process of reducing concept lattices is described and supported by algorithms.

1 Introduction

Formal Concept Analysis (Ganter and Wille, 1999) provides a mathematical method for data analysis which includes graphical representations in form of Hasse[1] diagrams of concept lattices[2]. Unfortunately diagrams for concept lattices with more than a small number of concepts contain many edges rendering them difficult to visually parse. Without being trained in reading diagrams, people often misread Hasse diagrams, for example, by mistaking the edges to be vectors[3]. Small Euler diagrams, however, can usually be read by people without any significant amount of training[4]. But Euler diagrams require a small size and careful layout algorithms in order to be comprehensible. Therefore two questions arise: first, how can formal contexts be reduced or partitioned so that their diagrams are sufficiently small to be easier to represent and, second, in order to avoid training users in reading diagrams, how can concept lattices be represented as Euler diagrams?

We are arguing in this paper that an optimal layout of Hasse or Euler diagrams cannot be provided via a purely deterministic algorithm because it may also depend on the content of the data. For example, some objects or attributes may be similar, belong to a shared domain or enhance or contradict each other and should be represented in a manner that reflects such relationships. These other relationships amongst objects or attributes can be considered *background knowledge* which might change with each application and need not necessarily be mathematically precise. Therefore, instead of a

⋆ Published in Dürrschnabel et al. Proc of ICFCA'23, LNCS 13934, Springer-Verlag

[1] The diagrams should not be named after Hasse because he did not invent them, but the name is widely established in the literature.

[2] An introduction to FCA is not included in this paper. Standard FCA terminology is used: *intension*, *extension*, a *cross* table for the formal context, and so on. An *object concept* is the lowest concept which has the object in its extension. *Attribute concepts* are defined dually.

[3] Based on the author's personal teaching experience with introductory mathematics courses.

[4] Again based on the author's teaching experience. "Small" in this paper means < 20 concepts.

single algorithm that produces optimal layouts, a semi-automated process is favourable that uses heuristics and interacts with users by asking them questions about the data and by giving them choices between layout options. Such a process is similar to "conceptual exploration" (Ganter and Obiedkov 2016) but conceptual exploration tends to modify the data contained in formal contexts whereas this paper assumes that only the diagrams are modified but not the original data.

Most existing research papers that discuss FCA diagram layouts or data reduction are not relevant for this paper because their lattices are still larger than in this paper, use fuzzy or probabilistic reduction methods or focus on navigation through a lattice (e.g. Vogt and Wille (1995), Cole, Eklund and Stumme (2003) or Alam, Le and Napoli (2016)). There is a large body of research on visualisations with Euler diagrams (e.g. Chapman et al. 2014), but not relating to concept lattices. The aim of this paper is to improve visualisations of conceptual hierarchies that are small, non-fuzzy but with background knowledge, for example visualisations of relationships in lexical databases (Priss and Old 2010) and of conceptual representations used for teaching. A goal is to improve existing software[5] by providing fairly simple generic algorithms for operations as outlined in the Appendix. Mathematically, such algorithms need not be complicated. Complexity considerations are not relevant for small data sets and not discussed in this paper. A challenge, however, is to respect user preferences and preserve implicit information within the data according to background knowledge. Furthermore, generating Euler diagrams is more challenging than generating Hasse diagrams as explained in the next section.

Digital tools for mathematics education often rely on "multiple linked representations (MLR)" which connect different visualisations so that changes in one part affect changes in other parts (Ainsworth 1999). In the case of FCA, an MLR display can show a formal context, its lattice and its implication base simultaneously. Changes in the context will then, for example, instantly change the lattice and the implication base[6]. Pedagogically, MLRs are useful because according to educational research, mathematical thinking often involves simultaneously imagined representations and fast shifting between them. For example, Venn and Euler diagrams are suitable means for teaching introductory set theory. But the fact that the empty set is a subset of every other set cannot be shown with a Venn or Euler diagram. It can only be concluded from the definition of "subset" by logical reasoning. Thus students need to learn that some knowledge can be represented diagrammatically (such as the subset relationship in Euler diagrams) whereas other knowledge must be represented textually using logical formulas.

Applying the idea of MLRs to FCA, one can argue that it is not necessary to represent all information contained in a formal context as a diagram of a concept lattice but instead that some information can be presented diagrammatically and some textually using implications or other logical statements. Therefore a strategy for reducing concept lattices proposed in this paper is to offload some information from diagrams into a textual representation. In some cases "a picture is worth a 1000 words". But in other cases, textual representations are simpler. For example, compared to drawing a Boolean

[5] https://github.com/upriss/educaJS

[6] An example of such an MLR of context and lattice can be found at https://upriss.github.io/educaJS/binaryRelations/fcaExample.html

lattice, a statement that all attributes from a set occur in any possible combination may be simpler. The decision as to which information should be presented diagrammatically or textually is often not based on mathematics but instead on background knowledge.

The next section introduces the specific type of Euler diagram that is suggested in this paper. Section 3 compiles a list of reduction methods that are applicable to non-fuzzy, non-probabilistic data. Section 4 demonstrates how in particular a reduction method that utilises concept extensions for partitioning objects is promising for generating Euler diagrams. Section 5 provides a short conclusion. An Appendix contains algorithms for the reduction methods.

2 Rounded Rectangular Euler Diagrams

This section argues that rounded rectangular Euler diagrams are ideally suited for representing small concept lattices and other set inclusion hierarchies. Euler diagrams are a form of graphical representation of set theory that is similar to Venn diagrams but leaves off any zones that are known to be empty (Figure 1). Most authors define the notion of "Euler diagram" in a semi-formal manner (Rodgers 2014) because a formal definition that captures all aspects of an Euler diagram is complex. Furthermore, mainly a certain subset of *well-formed* diagrams is of interest but so far no algebraic characterisation of that subset has been found. Euler diagrams consist of closed curves with labels representing sets. The smallest undivided areas in an Euler diagram are called *minimal regions* (Rodgers 2014). Regions are defined as sets (or unions) of minimal regions. *Zones* are maximal regions that are within a set of curves and outwith the remaining curves. In this paper, Euler diagrams are required to be well-formed in the sense that at most two curves intersect in any point and that each zone must not be split into several regions. Thus it is not necessary to distinguish between zones and minimal regions. The left half of Figure 1 shows a well-formed Euler diagram with 6 zones (including the outer zone).

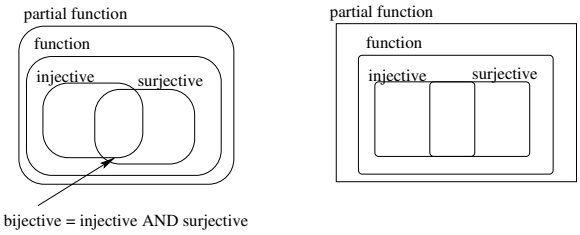

Fig. 1. Rectangular Euler diagrams with and without rounded corners and distortion

It is generally accepted that well-formed diagrams are easier to read than non-well-formed diagrams (Rodgers 2014) but not all subsets of a powerset can be represented as a well-formed diagram. In order to represent more, *shading* can be applied to zones that are meant to be empty. With shading any subset of a powerset can be drawn as

a well-formed Venn or Euler diagram. But shading several non-adjacent zones also renders diagrams difficult to read and thus should be used sparingly. There is some indication that being a lattice, or at least complete with respect to intersections, increases the probability of being representable as a well-formed Euler diagram but that is neither a necessary nor a sufficient condition (Priss 2020). Tree-like set hierarchies can always be represented as well-formed Euler diagrams. For lattices, it helps if one is allowed to omit the bottom concept, thus rendering the structure slightly more tree-like.

A further common condition for being well-formed is that curves must not be concurrent which means disallowing curves to meet in more than one adjacent point. This condition is actually dropped in this paper because it allows more subsets of a powerset to be represented. For example, a set of 3 elements has a powerset of 8 elements which has $2^8 = 256$ possible different subsets. If one considers only those where the outer zone is not empty (i.e. 128 subsets), only about 35 of these can be drawn as well-formed Euler diagrams without shading if each set is represented as a circle. If sets are represented as rectangles and curves are allowed to be concurrent, then we counted about 100 subsets that are representable without shading. The left and middle images in Figure 2 show two examples that are representable in the manner suggested in this paper, but not representable using circles without shading. Boolean lattices with more than 4 sets cannot be represented as rectangular Euler diagrams, but are also difficult to read as Hasse diagrams. A fifth set can be added as a concave curve, as shown with the dashed line in Figure 2 (right image). Because not all formal contexts with more than 4 elements are Boolean, it is still possible to represent some concept lattices with more than 4 attributes with rectangular Euler diagrams.

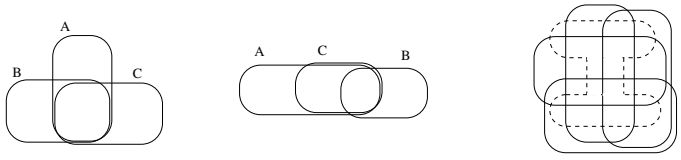

Fig. 2. Euler diagrams that cannot be "well-drawn", two dimensional $\{A, B, C, ABC\}$ (left) and one dimensional $\{A, B, AC, ABC\}$ (middle). Boolean lattice with 5 elements on the right (three dimensional).

Blake et al. (2014) conduct a usability study and conclude that circular Euler diagrams are much more readable than rectangular ones. The Euler diagrams in this paper, however, are not strictly rectangular, but instead *rounded rectangular Euler diagrams with distortion*, abbreviated as *RD-Euler diagrams*. In this case "rounded" refers to the corners. As demonstrated on the right hand side of Figure 1, if the corners are not rounded, then all sets and their intersections have the same shape and are difficult to distinguish. In circular diagrams only the sets are circles, whereas the intersections are composed of circle segments. Using RD diagrams (i.e. with purposeful imperfections) has the effect of changing the shape of the intersections and rendering them easier to discern. An alternative would be to use different colours for different sets but that potentially interferes with shading. Because of the distortion it is possible to see the shape

of each set in Figure 2. The small intersections between the concurrent edges are meant to not exist or in other words be empty. It is assumed that users will consider them drawing errors and ignore them (but a usability study should determine which size and shape of distortion is still acceptable to users and at what point shading is required).

Figure 2 also shows that in some cases the sets and their intersections can be arranged in a linear sequence as *one dimensional diagrams*. Petersen (2010) describes the conditions under which concept lattices can be linearised in that manner. Chapman et al. (2014) observe that one dimensional linear diagrams (containing lines instead of curves) have a higher usability for certain tasks than Euler diagrams because users read them faster and more accurately even though instead of shading such diagrams use repetition of attributes. Two dimensional diagrams can be embedded into direct products of one dimensional diagrams. Thus for small formal contexts a simple algorithm consists of repeatedly splitting the sets of attributes in half and then checking for linearity in order to compute a representation as an RD-Euler diagram. We are arguing that two dimensional diagrams are not significantly more complicated than one dimensional ones because only two directions (horizontal and vertical) need to be observed and should thus have similar advantages as observed by Chapman et al. (2014). Three dimensional diagrams (Figure 2 on the right) are difficult to read unless the sets of the third dimension are small and convex.

Apart from being able to express more subsets of a powerset than circular Euler diagrams, another advantage is that RD-Euler diagrams are easier to shade and label when drawn with software. In order to shade intersections one needs to compute their exact geometrical shape which is easier for rectangles than for circles. Furthermore it is more difficult to attach labels to circles. For rectangles, labels can be written adjacent to the edge of a curve either exactly horizontally or exactly rotated by 90 degrees. Intersections cannot be labelled in that manner and do require a line or arrow as in the left half of Figure 1 for either circular or RD-Euler diagrams.

3 Reducing Concept Lattices

This section provides a brief summary of methods for reducing concept lattices by offloading information from diagrams and instead storing it as logical expressions. The resulting RD-Euler diagrams then focus on information that is easier to represent graphically whereas the expressions focus on information that is more suited to textual representation. As mentioned in the introduction, the topic of reducing lattices has been discussed in the literature but usually with a focus on special properties or using fuzzy/probabilistic methods. The methods discussed in this section are more general, simple and mostly well-known. But the practical, joint use of diagrams and textual information has not yet received much attention. The assumptions about the data of a given formal context for this paper are:

crisp, non-fuzzy data: probability or other approximations are not relevant for this paper.

background knowledge may be available providing further relations, groupings, domains and so on for the objects and attributes.

three valued logic applies to most formal contexts because a missing cross can either mean that an attribute does not apply or that it is unknown whether it applies.

negatable attributes use binary logic in that objects either definitely do or do not have such attributes according to background knowledge.

supplemental concepts are concepts that can never be object concepts according to background knowledge even if more objects are added to a formal context (Priss 2020).

A formal context can be split into parts that are implications, diagrams and cross tables as long as it is clear how the original context can be recomputed from the different parts. A step-wise reduction should start by storing the original set of implications, because reduction can change implications. At any reduction step, if implications are lost, they must be extracted and stored as well. As a final step, only the implications that are not visible from the diagrams should be displayed textually. How to structure implications so that they are most effective for human users is not trivial and still an open research question that has been discussed elsewhere (Ganter 2019). Using general logical expressions instead of implications complicates matters but only if logical properties are to be computed, not if information is just displayed. Furthermore, as discussed in the introduction, it is assumed that the methods described here are executed in a semi-automated manner. Users will be asked questions about the data and then be presented with diagrams and texts using different reduction methods. Users can then choose which displays they prefer and whether to stop or try further methods.

The following listing summarises reduction methods, many of which are well known in some form, but according to our knowledge the approach of offloading information from each reduction step into a textual expression has not yet been studied much. The first four methods are discussed by Priss (2021). They can be easily implemented via an algorithm that searches through a formal context as shown in the Appendix. In our opinion the last method is particularly interesting and has not yet received as much attention in the literature. As a rule, contexts with fewer crosses result in less complicated diagrams. Thus a guiding strategy is to remove attributes and crosses or to split contexts.

SYNONYM-reduction: if several attributes have the same extensions then all but one can be removed. Expressions should be added using ":=".

AND-reduction: an attribute whose attribute concept is a meet of other concepts can be removed. An expression using "AND" should be added (e.g. "bijective" in Figure 1).

OR-reduction: an attribute whose attribute concept is a supplemental concept can be removed if its extension is a union of extensions of other attribute concepts. An expression using "OR" should be added. Caution: if several OR-reducible attributes exist, they cannot be removed simultaneously because removal of one may affect the others.

NOT-reduction: if an attribute is a negation of another attribute and both are negatable, the one with more crosses can be removed. An expression using "NOT" should be added.

NEGATION-reduction: if an attribute is negatable and its column in a context is filled more than half with crosses, then it can be replaced by its negation. An expression using "NOT" should be added.

FACTORISATION-reduction: if a context can be factorised using binary matrix multiplication so that the set of factors is smaller than the original set of attributes or smaller than a chosen value, then the set of factors can replace the set of original attributes (Belohlavek and Vychodil 2010). An expression using "AND" should be stored for each factor. Caution: factors cut the lattice vertically in half. Thus the height of the lattice is reduced but the horizontal structure at the level of the cut remains the same. Also, background knowledge may determine whether or not factors are a reasonable choice for a domain.

HORIZONTAL SPLIT: applicable if removing the top and the bottom concepts (which must both be supplemental) partitions the Hasse diagram into several separate graphs. The only implications that involve attributes from different parts use the bottom concept and may not even be of interest according to background knowledge. If the different parts have nothing in common, then no expressions need to be stored.

LOWER HORIZONTAL SPLIT: if removing the bottom concept and some supplemental concepts other than the top concept partitions the Hasse diagram into several separate graphs so that all object concepts are below the removed concepts. (This is different from factorisation because only object concepts matter for this type of splitting.)

PARTITIONING attributes: this is commonly used in FCA, for example, for nested line diagrams (Ganter and Wille, 1999). Caution: it changes the set of implications and there is no easy manner in which to store the lost information as expressions.

PARTITIONING objects: no implications are lost but each partition may have additional implications which do not hold in the original context.

CONCEPTUAL-PARTITIONING: same as the previous one but the set of objects is partitioned so that each partition is an extension of a concept or an extension of a concept of a negated attribute. The additional implications that are generated can be expressed using quantification over extensions: "FOR ALL ... that belong to extension ... : ...".

AND-reduction does not change a concept lattice. The methods of OR-and NOT-reduction and horizontal split are very effective at reducing a diagram, but the conditions for applying these are not met by many formal contexts. Factorisation depends on background knowledge and might not reduce the complexity of a diagram significantly. Negation reduction is very promising but requires a binary logic for its attribute. Therefore, partitioning appears to be the most interesting reduction method. As far as we are aware, the Darmstadt FCA group around Rudolf Wille favoured partitioning attributes over partitioning objects. As explained below, at least with respect to implications, partitioning objects is a much more promising choice. In our opinion, while nested line diagrams have proven to be useful for some applications, the diagrams are not easy to visually comprehend as a whole. If users zoom into one part of a nested diagram, they need to memorise where they are and what the relevant outer attributes are.

For conceptual partitioning, the set of objects is partitioned so that each partition can be expressed using statements involving attributes. For example, if the extensions of attributes a and b partition the set of objects, then the two partitions can be referred to as "objects having a" and "objects having b". If attributes are negatable, then one can also partition into "objects having a" and "objects not having a" or into "objects having a", "objects having b" and "objects having neither a nor b". Most likely negation should be used sparingly and not nested in order to avoid creating very complicated statements. For the same reason, not too many partitions should be created and it is ideal if the attributes chosen provide some obvious categorisation according to background knowledge (for example, "people in their 20s", "people in their 30s", and so on). Horizontal split is a type of conceptual partitioning because the extensions of the immediate lower neighbours of the top concept partition the set of objects. Lower horizontal split is a type of conceptual partitioning if the objects of each part exactly correspond to an extension of a concept. Figure 3 in the next section shows an example.

With regard to evaluating the effectiveness of reduction methods, a few simple measures can be defined:

M_{cpt}: total number of concepts in a lattice
M_{obj}, M_{att}, M_{oa}: numbers of object and attribute concepts and concepts that are simultaneously object and attribute concept, respectively
M_{spl}: number of supplemental concepts
M_{lbl}: number of labelled concepts, $M_{lbl} := M_{obj} + M_{att} - M_{oa}$

For the sake of avoiding unnecessary concepts, an optimally reduced lattice should have $M_{cpt} \approx M_{lbl}$ if information about supplemental concepts is not available and $M_{cpt} \approx M_{cpt} - M_{spl}$ otherwise. In general, supplemental concepts should be minimised because they do not add any useful information about objects that is not already contained in other concepts. But concepts that are neither object nor attribute concepts and not supplemental should not be removed during reduction because their extensions are important sets according to background knowledge. In order to avoid losing such concepts during reduction, a strategy is to add objects so that $M_{cpt} - M_{spl} = M_{lbl}$. Unfortunately, whether a concept is supplemental can only rarely be deduced from its neighbouring concepts: if a concept a is OR-reducible, then all concepts between a and an attribute concept below it, will be supplemental. In general, if a concept is supplemental, then its lower neighbours should be checked because they could also be supplemental. Sometimes background knowledge provides simple rules about supplemental concepts. For example, if it is known that every object has at least 3 attributes, then any concept at the top of the lattice with fewer than 3 attributes must be supplemental. As demonstrated in the examples below for conceptual partitioning, it can be important to check whether the immediate upper neighbours of object concepts and the joins of object concepts are supplemental because they might be removed during conceptual partitioning.

4 Two Examples of constructing Euler diagrams

In this section, the Euler diagrams of two well known examples of lattices are drawn after applying conceptual partitioning. The examples are the body of waters lattice that is on the cover of Ganter and Wille (1999) and the lattice of binary relations (in the same book, after page 85). Ganter and Wille present these lattices as additive or nested line diagrams which highlights Boolean sublattices and symmetries. But based on Chapman et al. (2014), it can be expected that one or two dimensional Euler diagrams will be easier to read for most people.

Figure 3 shows a Hasse diagram for the body of waters example on the left. All of the concepts in the upper half are supplemental (drawn as empty circles) because objects are meant to have 4 attributes. Only object “channel” has 3 attributes because it has not been decided whether it is natural or artificial. In this case, supplemental concepts can be determined by asking users the single question as to whether “channel” is an exception and otherwise objects are expected to have 4 attributes. A goal of a reduction should then be to remove the supplemental concepts. In this example, NOT-reduction might appear promising. But because the set of attributes is a set of 4 antonymic pairs, it is not clear which pair to choose. If NOT-reduction is applied to all 4 pairs, then only the attributes “temporary”, “running”, “artificial” and “maritime” would be kept. The lattice would consist of very few concepts and most of the supplemental concepts would

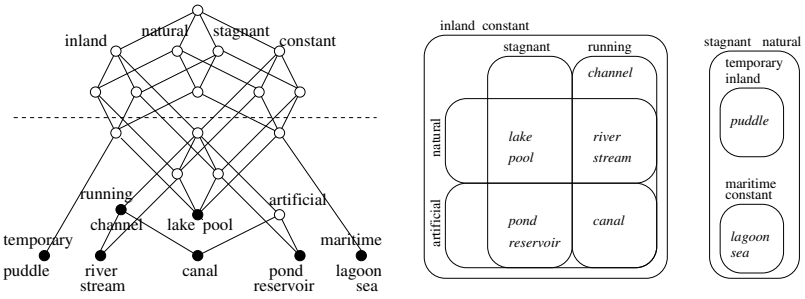

Fig. 3. Body of waters Hasse diagram (left), Euler diagrams "inland AND constant" (middle) and "NOT inland OR NOT constant" (right)

disappear but the lattice would be almost an antichain and not contain much information about the relationships amongst the objects anymore. Therefore after users have seen the results of NOT-reduction, they should be asked whether they would also like to see the result of other reduction methods. In this case a lower horizontal split is possible (the dashed line in Figure 3) and results in conceptual partitioning using the negatable extension "{inland, constant}". The right half of Figure 3 shows the resulting partitions "inland AND constant" and "NOT inland OR NOT constant". Implications can still be read from the partitioned Euler diagrams but they need to be quantified. For example: for all objects that are "NOT inland OR NOT constant", inland implies stagnant.

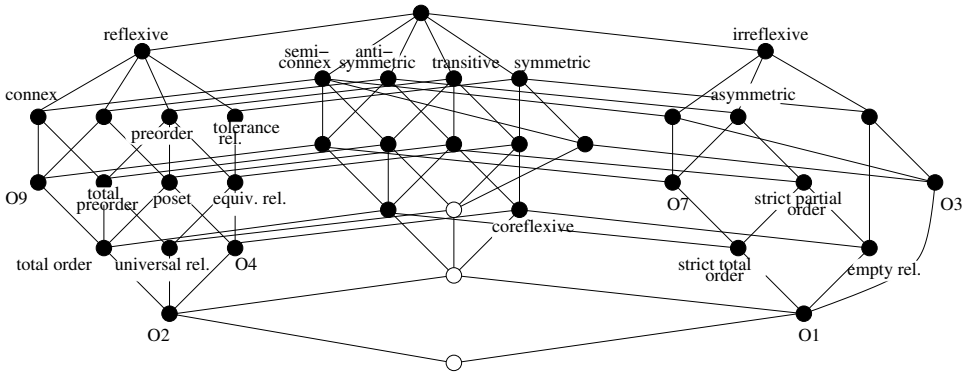

Fig. 4. A concept lattice of types of binary relations

The second example shows a lattice of properties of binary relations (Ganter and Wille 1999). The details of the content of the lattice are not relevant for this paper. A Hasse diagram of the original concept lattice is shown in Figure 4. In this example, there are only very few supplemental concepts for which Wikipedia can be consulted as a source of background knowledge: if a type of a relation has a name in Wikipedia, then it should not be supplemental and its name can be added as an object. For example,

a reflexive and transitive binary relation is called “preorder” in Wikipedia. Although not all supplemental concepts can be decided in that manner, adding objects already has an impact as discussed below. A few of the original objects from Ganter and Wille (1999) do not have special names but must be kept because their object concepts are irreducible. For example, O9 is a connex and antisymmetric relation, but there is no special name for a connex and antisymmetric relation.

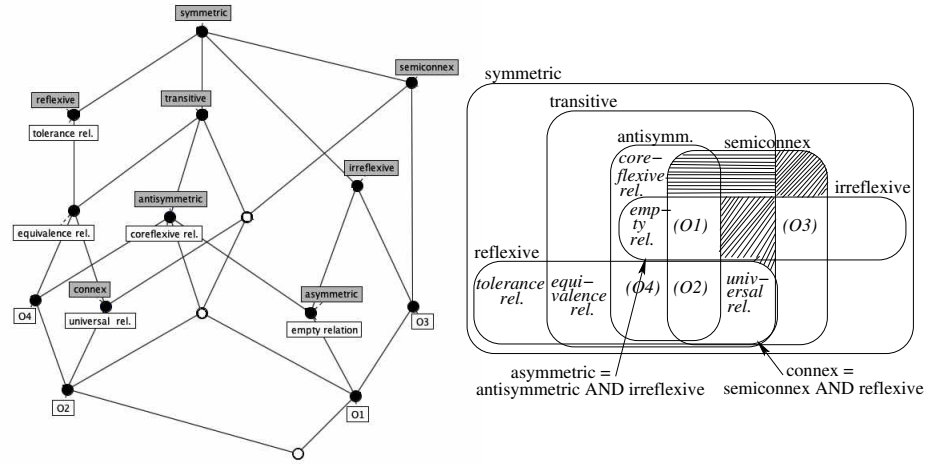

Fig. 5. Hasse and Euler diagram for a concept lattice of symmetric relations

A first obvious choice for reduction would be to use negation on “reflexive” or “irreflexive” or to partition the objects according to whether they are reflexive or irreflexive. But after adding the object “coreflexive” during the search for supplemental concepts, the attributes “reflexive” and “irreflexive” are no longer negatable and no longer partition the set of objects. A lower horizontal split is not possible for this example. A next step is to offer users conceptual partitioning for any of the other attributes whose attribute concepts are lower neighbours of the top concept after checking with users that these are indeed negatable. Based on background knowledge it can be detected that the word “order” in the strings of the object names correlates with “NOT symmetric”. Thus partitioning into “symmetric” and “NOT symmetric” is a promising choice. The resulting Hasse and Euler diagrams are shown in Figures 5 and 6. Figure 5 employs two kinds of shading: horizontal lines for supplemental concepts and diagonal lines for zones that are not concepts of the lattice. The bottom concept has been omitted. If the condition of being a lattice is dropped, then the top 3 shaded zones can be removed leaving a diagram with only 1 shaded zone. Although the Euler diagrams are not showing all of the symmetries of the lattice, they are easy to read. For example, in order to determine the properties of a total order in the Euler diagram of Figure 6 one only needs to look up (reflexive and semiconnex) and right (transitive and antisymmetric). Optionally below the diagram, it is furthermore stated that connex = reflexive AND semiconnex. Thus connex applies as well.

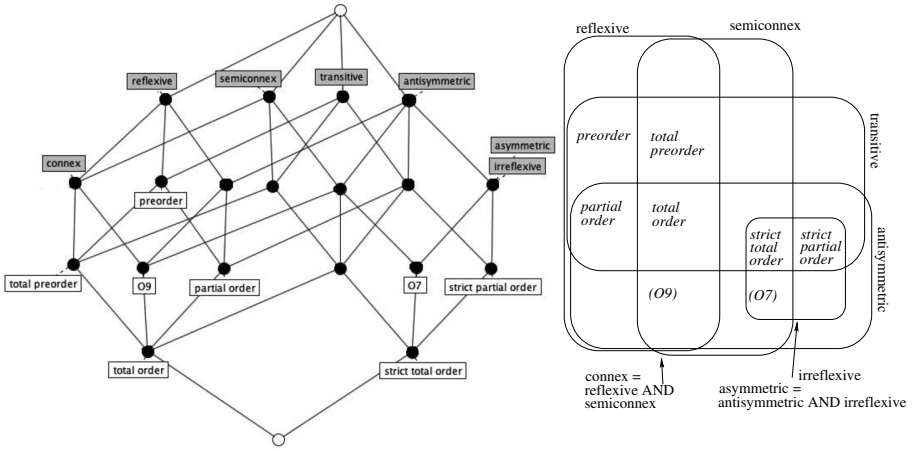

Fig. 6. Hasse and Euler diagram for a concept lattice of relations that are not symmetric

5 Conclusion

This paper discusses means for generating RD-Euler diagrams for representing concept lattices by reducing the information contained in the diagrams, potentially separating the information into several diagrams and implications or other logical expressions. While some algorithms have already been implemented (and are shown in the Appendix), an important future step will be to develop an algorithm for the precise layout of nicely formatted Euler diagrams. A semi-automated algorithm which produces some initial layouts to be further manipulated by users would already be a significant achievement because currently such software appears to be lacking. But a discussion of such algorithms will be left to another paper.

Conducting a user study comparing Hasse and Euler diagrams would also be of interest but would be difficult. Such a study is only sensible if the users have received some training in reading Hasse diagrams. If the study was conducted within the FCA community, then the users might be biased towards Hasse diagrams and furthermore they would have some mathematical background and thus possibly higher spatial reasoning abilities than the average population. FCA knowledge has an impact on reading Hasse diagrams as well. Anybody who observes the Boolean lattice over “total order” in Figure 6 will instantly read its attributes from the diagram. Anybody who has not been trained in FCA will have to follow all edges that are leading up from “total order”. This will be more error prone than reading an Euler diagram if one considers Chapman et al.’s (2014) results. Thus future user studies evaluating numbers of errors in reading RD-Euler diagrams and comparing them to other Euler diagrams might be more insightful than comparing Hasse and Euler diagrams. Since software for producing Hasse diagrams exists, it would be easy to simply add such software to Euler diagram software and give users a choice whether to display Hasse or Euler diagrams.

References

1. Ainsworth, S. (1999). The functions of multiple representations. Computers & Education, 33, 2-3, p. 131-152.
2. Alam, M.; Le, T.N.N.; Napoli, A. (2016) Steps towards interactive formal concept analysis with latviz. Proc. of 5th Intern. Workshop "What can FCA do for Artificial Intelligence"? CEUR Workshop Series, available at https://ceur-ws.org/Vol-1703/.
3. Belohlavek, R.; Vychodil, V. (2010). Discovery of optimal factors in binary data via a novel method of matrix decomposition. J. Comput. Syst. Sci., 76, 1, p. 320.
4. Blake, A.; Stapleton, G,; Rodgers, P.; Cheek, L.; Howse, J. (2014). The Impact of Shape on the Perception of Euler Diagrams. In Dwyer et al., Proceedings of Diagrams 2014, LNCS 8578, Springer, p. 123-137.
5. Chapman, P.; Stapleton, G.; Rodgers, P.; Micallef, L.; Blake, A. (2014). Visualizing sets: An empirical comparison of diagram types. In Dwyer et al., Proc. of Diagrams 2014, Springer, p. 146-160.
6. Cole, R.; Eklund, P.; Stumme, G. (2003). Document retrieval for e-mail search and discovery using formal concept analysis. Applied artificial intelligence, 17, 3, p. 257-280.
7. Ganter, B.; Wille, R. (1999). Formal Concept Analysis. Mathematical Foundations. Berlin-Heidelberg-New York: Springer.
8. Ganter, B. (2019). "Properties of Finite Lattices" by S. Reeg and W. Weiß, Revisited. In Cristea et al. (eds) Proc. of ICFCA 2019. LNCS 11511, Springer, p. 99-109.
9. Ganter, B.; Obiedkov, S. (2016). Conceptual exploration. Heidelberg: Springer.
10. Petersen, W. (2010). Linear Coding of Non-linear Hierarchies: Revitalization of an Ancient Classification Method. In Advances in Data Analysis, Data Handling and Business Intelligence, Springer, p. 307-316.
11. Priss, U.; Old, L. J. (2010). Concept Neighbourhoods in Lexical Databases. In Kwuida; Sertkaya (eds.), Proc. of ICFCA'10, Springer, LNCS 5986, p. 283-295.
12. Priss, U. (2020). Set Visualisations with Euler and Hasse Diagrams. In Graph Structures for Knowledge Representation and Reasoning: 6th International Workshop, Springer, LNCS.
13. Priss, U. (2021). Modelling Conceptual Schemata with Formal Concept Analysis. In Proc. of FCA4AI'21.
14. Rodgers, P. (2014). A survey of Euler diagrams. Journal of Visual Languages & Computing, 25, 3, p. 134-155.
15. Vogt, F., Wille, R. (1995). TOSCANA - a graphical tool for analyzing and exploring data. In Tamassia & Tollis (eds.): Graph Drawing. LNCS 894. Springer, p. 226-233.

Appendix: Algorithms

This appendix provides a few of the discussed algorithms. Factorisation algorithms are omitted because they are described by Belohlavek and Vychodil (2010). The algorithms are to be used with existing FCA software. Therefore it can be assumed that the following exist:

- lists O, A, C of objects, attributes, concepts
- list L of logical statements (e.g. implications)
- lists O_C and A_C for the object/attribute concepts
- 2-dimensional Boolean array J of size $|O| \times |A|$ for the formal context
- 2-dimensional Boolean array N of size $|C| \times |C|$ containing the immediate upper neighbours for each concept (i.e. the edges of the Hasse diagram)

The following should be computed:

- 2-dimensional array P of size $|A| \times |A|$ computed by pairwise comparison of the column bitvectors of the attributes. Contains: "e" if the two attributes are equal, "s" for "smaller not equal" (i.e. for every 1 of the first there must be a 1 in the second), "g" for "greater not equal", "n" for negation and "0" otherwise.
- procedure *delete(a)* removes an attribute a from A and J and recomputes everything else as required
- procedure *neigh(c)* returns the set of upper neighbours of concept c according to N
- procedure *meet(a)* returns a (shortest) list of attributes with binary intersection a or NONE.
- procedure *join(a)* returns a (shortest) list of attributes with binary union a or NONE.

The algorithm SynAndOrNot computes attributes that are reducible with Synonym-, AND-, OR- or NOT-reduction. The algorithm Negate computes NEGATION-reduction. Horizontal splits are computed with the final algorithm. For a lower horizontal split, the algorithm HorizontalSplit is used but is stopped as soon as the immediate upper neighbours of all object concepts have been processed. Then the top concepts are removed from each partition and it is checked whether the partitions remain unchanged if all lower neighbours (apart from the bottom concept) are added to each partition.

An Algorithm for Conceptual Partitioning is not provided in detail. Several different steps and strategies can be used to determine which concept extensions are most suitable for partitioning the objects:

- determine which concepts are supplemental and whether objects should be added to non-supplemental concepts
- check whether any other reduction methods are applicable
- run a LOWER HORIZONTAL SPLIT algorithm and determine whether each set of objects corresponds to a concept extension
- any negatable attribute always partitions the set of objects, therefore identify negatable attributes that partition the set of objects approximately in half (can involve single attributes, but also 2 or 3 attributes)
- in order to compare all different possibilities of partitioning, determine which one reduces the number of supplemental concepts
- show different versions to users and ask them which they prefer

Algorithm 1 Algorithm SynAndOrNot

```
1:  for i = 0; i < length(A)-1 do
2:      if P(A[i],A[i+1]) == "e" then
3:          ask user to choose one of the two
4:          a1 = user's choice to be deleted
5:          delete(a1)
6:          append "A[i] = A[i+1]" to L
7:      else if N(A_C(A[i])) contains more than one 1 then
8:          (A1, ..., An) = meet(A[i])
9:          delete(A[i])
10:         append "A[i] = A1 AND ... AND An" to L
11:     else if P(A[i],A[i+1]) == "g" and A[i] not in O_C then
12:         (A1, ..., An) = join(A[i])
13:         if not NULL then
14:             ask user whether concept is supplemental
15:             if yes then
16:                 delete(A[i])
17:                 append "A[i] = A1 OR ... OR An" to L
18:     else if P(A[i],A[i+1]) == "n" then
19:         if J(A[i]) contains more than length(O)/2 1s then
20:             a1 = A[i]
21:         else
22:             a1 = A[i+1]
23:         delete(a1)
24:         append "A[i] = NOT A[i+1]" to L
25:     else
26:         next
```

Algorithm 2 Algorithm Negate

```
1:  for i = 0; i < length(A) do
2:      if J(A[i]) contains more than length(O)/2 1s then
3:          ask user whether A[i] is negatable
4:          if yes then
5:              replace A[i] with NOT(A[i])
6:              change A and J accordingly
7:              recompute everything as required
```

Algorithm 3 Algorithm HorizontalSplit

```
1: initialise Set1 as a list of sets
2: for item in neigh(bottom_concept) do
3:     Set1[item] = neigh(item)
4: Cpts = union(neigh(bottom_concept))
5: to_process = Cpts
6: processed = {bottom_concept, top_concept}
7: while length(processed) < length(C) do
8:     for c2 in to_process do
9:         for item1 in Set1 do
10:            if c2 not in Set1[item1] then
11:                next
12:            for item2 in Set1 do
13:                if item1 == item2 then
14:                    next
15:                if c2 in Set1[item2] then
16:                    Set1[item1].add(Set1[item2])
17:                    Set1[item2].add(Set1[item1])
18:    for item1 in Set1 do
19:        for c3 in intersect(Set1[item1],to_process) do
20:            Set1[item1].add(neigh(c3))
21:            Cpts = union(Cpts,neigh(c3))
22:    processed = union(processed,to_process)
23:    to_process = Cpts - processed
24: for item in Set1 do
25:    if length(Set1[item]) < length(C)-2 then
26:        print(Set1[item] - top/bottom_concept is a partition)
```