Algebraic Visualization of Relations Using RelView

Rudolf Berghammer[1] and Gunther Schmidt[2]

[1] Institut für Informatik, Christian-Albrechts-Universität Kiel
Olshausenstraße 40, 24098 Kiel, Germany
rub@informatik.uni-kiel.de
[2] Fakultät für Informatik, Universität der Bundeswehr München
85577 Neubiberg, Germany
gunther.schmidt@unibw.de

Abstract. Relations and graphs are widely used as modeling tools. For graphs, there exist highly elaborated graph drawing algorithms that help getting an impression on how the graph is structured. We concentrate here in an analogous way on visualizing relations represented as Boolean matrices as, for example, in the specific purpose Computer Algebra system RelView. This means rearranging the matrix appropriately, permuting rows and columns simultaneously or independently as required. In this way, many complex situations may successfully be handled in various application fields. We show how relation algebra and RelView can be combined to solve such tasks.

1 Introduction

Although graphs as well as relations are frequently used as modeling tools, the theory of graphs is more broadly known than the theory of relations. Theoretical aims of graph theory are often complexity considerations in case one is interested in asymptotic behaviour of algorithms. Many results of this type have been difficult to obtain, but frequently they also turned out to be without much practical relevance; cf. the discussion in [16] for example. On the other side, one often works with graphs of small or moderate size that model practical situations. In such cases, graph drawing is used as a supporting technique. There exist a lot of highly elaborated graph drawing algorithms and implemented tools, for general graphs as well as for specific classes, that help getting an impression on how the graph is structured. See, for example, [9] for more details (including aesthetic criteria and drawing paradigms) and a diversity of graph drawing algorithms.

Since many years relation algebra is used as a very convenient means for problem solving in mathematics, computer science, engineering and some other disciplines. A lot of practical applications of relation-based techniques can be found in the two books [7, 8] of the former EU COST Action 274 TARSKI. As demonstrated in [18] for example, graph theory and relation theory interact in many ways. But relations are geared towards an algebraic treatment, ultimately leading to the structure of a relation algebra in the sense of Tarski (see [21])

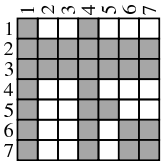
and, if required, its mechanization via appropriate Computer Algebra systems. While one may draw relations in a multitude of versions when interpreting them as graphs, we here aim at depicting them as Boolean matrices. This is often very useful for visual editing and for discovering structural properties that are not evident from a graph representation. To this end, the Boolean matrix is rearranged appropriately, permuting rows and columns simultaneously or independently as required, so as to have a more or less immediate impression of what it stands for. But there remains the question whether this is possible, whether it is justified algebraically, and how the desired form can be obtained algorithmically.

The observation concerning relations occurring in practice is that they are "not too big"; row or column numbers do often not exceed 40 or 50. Even if an algorithm turns out to be rather inefficient, it seems possible to handle that size with our actual computer equipment. A competent overview on Multi-Criteria Decision Aid of this type is given in [6] for example. It contains a considerable number of practical examples of tables that are limited in size but lend themselves to be investigated with algebraic methods with respect to several criteria. They resemble evaluations with patient material or the multi-purpose transnational water system of, e.g., Lago Maggiore.

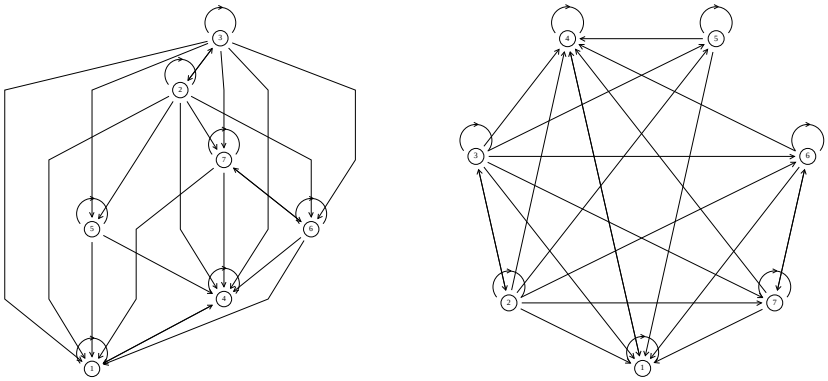
This is where our investigation started. Using some basic algorithms, we have developed a tool set of relation-algebraic expressions describing rearrangements of Boolean matrices / relations into specific forms. These could immediately be translated into the programming language of the specific purpose Computer Algebra system RelView [1, 3, 4]. While we have handled, among others, symmetric idempotent relations, matching decompositions, independent and/or covering pairs of sets, implication structure decompositions, equivalences, we restrict to presenting here all the types of orderings traditionally used by decision makers. Several pictures show how the system then can help in visualization.

2 A Motivating Example

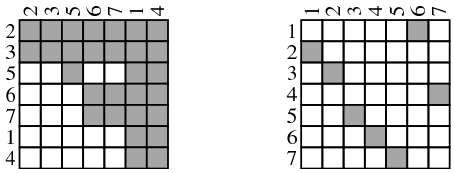
As a rather trivial initial example consider the following relation on elements $1, 2, \ldots 7$, represented with the RelView tool as a Boolean matrix.



A black square stands for the matrix entry 1 (or *true*) and a white square stands for the entry 0 (or *false*). It is easy to check that the relation R represented by the matrix is a *pre-order relation*, i.e., is *reflexive* ($\mathsf{I} \subseteq R$) and *transitive* ($RR \subseteq R$). With a graph drawing algorithm one would obtain something as shown in the following pictures. Again they are produced by RelView; the picture on the left uses the hierarchical polyline drawing algorithm of [13] and that on the right is the result of the spring-embedder algorithm of [15].



But these drawings do not give an appropriate impression of what the relation R really expresses. More intuitive is the left one of the following two RelView-matrices. It is given as an upper right triangle of rectangles of either 1's or 0's, where the four rectangle-forming parts correspond to the four sets $\{2,3\}$, $\{5\}$, $\{6,7\}$ and $\{1,4\}$ of indices of the original matrix.



Later we will show how such a rearrangement can be obtained from the original matrix. The key of our procedure will be the relation-algebraic specification of a permutation relation P; for the above example it is represented by the RelView-matrix on the right. Then the new version is obtained by multiplying R with the transpose of P from the left and with P from the right. The former rearranges the rows of R accordingly and the latter does so for the columns.

3 Relation Algebra

We write $R : X \leftrightarrow Y$ if R is a relation with domain X and range Y, i.e., a subset of the direct product $X \times Y$. If the base sets X and Y of R's *type* $X \leftrightarrow Y$ are finite and of size m and n, respectively, we may consider R as a Boolean $m \times n$ matrix. This interpretation is well suited for many purposes. As it is also used by RelView to depict relations, we will often use matrix terminology and notation, i.e., speak about rows and columns and write $R_{x,y}$ instead of $\langle x, y\rangle \in R$ or $x\,R\,y$. We assume the reader to be familiar with the basic operations on relations, viz. R^{T} (*transposition*), $\overline{R}$ (*complement*), $R \cup S$ (*union*), $R \cap S$ (*intersection*), and RS (*multiplication*), the predicate $R \subseteq S$ (*inclusion*), and the special relations O (*empty relation*), L (*universal relation*), and I (*identity relation*).

To model sets we will use *vectors*, which are relations v with $v = v\mathsf{L}$. Since for the relation modelling a set the range is irrelevant, we consider in the following

mostly vectors $v : X \leftrightarrow \mathbf{1}$ with a specific singleton set $\mathbf{1} = \{\bot\}$ as range and omit in such cases the subscript $\bot$, i.e., write v_x instead of $v_{x,\bot}$. Such a vector can be considered as a Boolean column vector, and *represents* the subset $\{x \in X \mid v_x\}$ of X. A non-empty vector v is a *point* if $vv^\mathsf{T} \subseteq \mathsf{I}$, i.e., it is *injective*. This means that it represents a singleton subset of its domain or an element from it if we identify a singleton set $\{x\}$ with the element x. In the Boolean matrix model a point $v : X \leftrightarrow \mathbf{1}$ is a Boolean column vector in which exactly one entry is 1.

When dealing with orders, one typically investigates extremal elements. In this paper we only need $\mathit{least}(C, v) = v \cap \overline{\overline{C \cup \mathsf{I}}\, v}$. For a *strict-order relation* $C : X \leftrightarrow X$ i.e., an *asymmetric* ($C \cap C^\mathsf{T} = \mathsf{O}$) and transitive relation, and a vector $v : X \leftrightarrow \mathbf{1}$ this relational function yields either a point that represents the least element of v wrt. C or an empty vector if no least element exists.

With $R \setminus S = \overline{R^\mathsf{T}\, \overline{S}} : Y \leftrightarrow Z$ the *right residual* of $R : X \leftrightarrow Y$ and $S : X \leftrightarrow Z$ is introduced. For $S = R$ this means in particular that $R \setminus R$ has type $Y \leftrightarrow Y$ and for all $x, y \in Y$ the x-column of R is contained in the y-column of R iff $(R \setminus R)_{x,y}$ holds. Hence, $R \setminus R$ coincides with the is-contained relation on the columns of R. The expression $(R^\mathsf{T} \setminus S^\mathsf{T})^\mathsf{T}$ defines the *left residual* $S \,/\, R : X \leftrightarrow Y$ of $S : X \leftrightarrow Z$ and $R : Y \leftrightarrow Z$. Here the case $S = R$ yields the transposed is-contained relation, i.e., the contains relation $S \,/\, S : X \leftrightarrow X$ on the rows of S.

4 A Short Look at the RelView Tool

RelView is an interactive and graphic-oriented specific purpose Computer Algebra system for relation algebra. All data it works on are represented as relations which the system visualizes as directed graphs (via several sophisticated graph drawing algorithms) or as Boolean matrices. RelView allows to compute with very large relations, for instance membership-, set inclusion-, and size comparison relations on powersets, as the system uses a highly efficient implementation of relations via binary decision diagrams (see [3, 4] for details). The user can manipulate and analyse relations by pre-defined operations and tests. Based on the operations and tests and certain additional control structures relational functions and relational programs may be defined. We exhibit three of them as examples, which we will need later.

The following unary RelView-function `Hasse` computes the Hasse-diagram $H_C = C \cap \overline{CC}$ of a strict-order relation C.

```
Hasse(C) = C & -(C * C).
```

A relational program essentially is a while-program based on the main datatype realized, namely relations. Such a RelView-program has many similarities with a function procedure in programming languages like Pascal or Modula-2. It starts with a head line containing the program's name and the list of formal parameters. Then the local declarations follow. The last part is the body, a sequence of statements which are separated by semicolons and terminated by the `RETURN`-clause. For an example, we recall E. Szpilrajn who proved in [20]

that every partial order relation possesses a linear extension, where R is *linear* if $R \cup R^{\mathsf{T}} = \mathsf{L}$. This theorem follows from Zorn's lemma and the fact that, given a partial order relation $E : X \leftrightarrow X$ and an incomparable pair $\langle a, b\rangle \in X \times X$, there exists an extension R of E that contains $\langle a, b\rangle$. Using element-wise notation, we can define R for all $x, y \in X$ by $R_{x,y}$ iff $E_{x,y}$ or $E_{x,a}$ and $E_{b,y}$. A relation-algebraic specification of R is $R = E \cup EAE$, where the relation A is a product pq^{T} of different points $p, q : X \leftrightarrow \mathbf{1}$ representing the elements a and b, respectively. In the finite case the extension may be iterated and this leads to the following RelView-program `Szpilrajn` for computing a linear extension of E.

```
Szpilrajn(E)
  DECL R, A
  BEG  R = E;
       WHILE -empty(-(R | R^)) DO
         A = atom(-(R | R^));
         R = R | R*A*R OD
       RETURN R
  END.
```

The next example is the RelView-program `Classes` (formally developed in [2]) that computes for an equivalence relation $R : X \leftrightarrow X$ with set $\mathfrak{C}$ of equivalence classes the canonical epimorphism from X to $\mathfrak{C}$ as a relation $\Phi : X \leftrightarrow \mathfrak{C}$.

```
Classes(R)
  DECL C, x, p
  BEG  C = R*point(Ln1(R));
       x = -C;
       WHILE -empty(x) DO
         p = point(x);
         C = (C^ + (R*p)^)^;
         x = x & -(R*p) OD
       RETURN C
  END.
```

Since Φ is the relational version of the canonical epimorphism, we have for all $x \in X$ and equivalence classes $c \in \mathfrak{C}$ that $\Phi_{x,c}$ iff x belongs to c. Hence, if we consider the columns of the result C of the RelView-program `Classes` as single vectors, then these vectors are pair-wise disjoint and precisely represent the elements of the set $\mathfrak{C}$. In the literature this is also called a column-wise representation of a set of sets, or the natural projection for the equivalence.

5 Some Simple Rearranging Algorithms

Our rearrangement algorithms are all based on pure relation algebra, supported by one additional fact: Sets X between which the relations hold we work with are necessarily finite; we assume them to be equipped with a linear strict-order relation $\Omega_X : X \leftrightarrow X$, the *base strict-order*. In RelView this order is implicitly

given by the internal enumeration of the base set X within the tool. Its Hasse-diagram is $succ : X \leftrightarrow X$. To be more precise, if $x_1, x_2, \ldots, x_n$ is the internal enumeration of X within the tool, then we have $succ_{x_i,x_{i+1}}$ for all i, $1 \leq i \leq n-1$, and the base strict-order Ω_X is obtained as the transitive closure $succ^+$ of $succ$.

5.1 Linear Strict-Order Relations

In what follows, let $C : X \leftrightarrow X$ be any linear strict-order relation on the set X and let $\Omega_X : X \leftrightarrow X$ be the base strict-order on X. Assuming Ω_X to be depicted as a full upper right triangle matrix as the Boolean matrix of the relation $succ^+$ in RelView, how can we permute the set X via a permutation relation $P : X \leftrightarrow X$ so as to see the linear strict-order relation C permuted accordingly as the full upper right triangle? Of course, this is a rather trivial task – but tedious when one has to actually execute it by hand. Subsequently, we show how it can be mechanized and how the algorithm can be formulated in RelView.

In the first step, we compute the two Hasse-diagrams $H_C = C \cap \overline{CC}$ and $H_{\Omega_X} = \Omega_X \cap \overline{\Omega_X \Omega_X}$ of the linear strict-order relations C and Ω_X. Next, we consider the least elements with respect to both these orders, represented by vectors $least(C, \mathsf{L})$ and $least(\Omega_X, \mathsf{L})$, where $\mathsf{L} : X \leftrightarrow \mathbf{1}$. Since L represents the entire base set X, the vectors represent the respective least element of the strict-ordered sets (X, C) and (X, Ω_X). The permutation relation P we are looking for, now is defined iteratively. We start with the relation

$$P_0 = least(C, \mathsf{L})\, least(\Omega_X, \mathsf{L})^\mathsf{T}$$

that precisely relates the least element of (X, C) with that of (X, Ω_X). It is easy to verify that the extension $P_0 \cup {H_C}^\mathsf{T} P_0 H_{\Omega_X}$ of the relation P_0 additionally relates the second smallest element of (X, C) with the second smallest element of (X, Ω_X), and no further relationships are introduced. Based on this observation, we successively apply the relational function

$$\tau(R) = R \cup {H_C}^\mathsf{T} R H_{\Omega_X}$$

to P_0. This leads to a finite chain $P_0 \subset \tau(P_0) \subset \tau^2(P_0) \subset \ldots \subset \tau^{|X|-1}(P_0)$ and the last relation of this chain obviously is the desired permutation relation P. A RelView-implementation of this procedure looks as follows.

```
PermLSO(C)
  DECL L, HC, HO, P, Q
  BEG  L = Ln1(C);
       HC = Hasse(C);
       HO = succ(L);
       P = least(C,L)*least(trans(HO),L)^;
       Q = P | HC^*P*HO;
       WHILE -eq(P,Q) DO
         P = Q;
         Q = P | HC^*P*HO OD
       RETURN P
  END.
```

Having a program at hand for computing this permutation relation P, the following RelView-program for the transformation of C is an immediate consequence.

```
RearrLSO(C)
  DECL P
  BEG  P = PermLSO(C)
       RETURN P^*C*P
  END.
```

The body of `RearrLSO` says that the desired form is obtained via $P^\mathsf{T}CP$ since, as already mentioned in Sect. 2, $P^\mathsf{T}C$ rearranges the rows of C accordingly and a subsequent multiplication with P from the right does so for the columns.

5.2 Pre-Order Relations

As a second kind of orderings, we consider *pre-order relations*. They are defined to be reflexive and transitive and appear quite frequently in optimization problems. The name *quasi-order relation* is also common for pre-order relations.

Each pre-order relation $Q : X \leftrightarrow X$ can be transformed into a form that consists of an upper right triangle of rectangles of 1's and 0's and is shown in the motivating example of Sect. 2. Again the decisive step is the computation of an appropriate permutation relation $P : X \leftrightarrow X$ on the base set X that rearranges the rows and columns accordingly by multiplying Q with P^T from the left and with P from the right. The computation of P is rather straightforward. First, we form the equivalence relation $R = Q \cap Q^\mathsf{T}$. In the second step, we remove this relation from Q. It is easy to verify that this yields a strict-order relation $Q \cap \overline{R}$ and, hence, the reflexive closure $(Q \cap \overline{R}) \cup \mathsf{I}$ is a partial order relation on the base set X. In the third step, we compute a linear extension $E : X \leftrightarrow X$ of this partial order relation. Since each permutation relation that transforms the linear strict-order relation $E \cap \overline{\mathsf{I}}$ into a full upper right triangle obviously transforms Q into an upper right triangle of rectangles, the last step consists in the application of the procedure of Sect. 5.1. In the syntax of the RelView tool the computation of the permutation relation P looks as follows.

```
PermPreO(Q)
  DECL I, R, E
  BEG  I = I(Q);
       R = Q & Q^;
       E = Szpilrajn((Q & -R) | I)
       RETURN PermLSO(E & -I)
  END.
```

A formulation of a RelView-program `RearrPreO` for computing the product $P^\mathsf{T}QP$ is completely analogous to that of `RearrLSO` and, therefore, omitted. At this place it should be mentioned that we have used `PermPreO` and `RearrPreO` to generate the last two matrices of the motivating example in Sect. 2.

5.3 Weak-Order Relations

A relation $W : X \leftrightarrow X$ is said to be a *weak-order relation* if it is asymmetric and *negatively transitive*, where the latter property is defined by the inclusion $\overline{W}\,\overline{W} \subseteq \overline{W}$ to hold. Weak-order relations W are precisely those strict-order relations in which the set of indifferent pairs (i.e., the set of pairs $\langle x, y\rangle \in X \times X$ such that neither $W_{x,y}$ nor $W_{y,x}$) forms an equivalence relation $R = \overline{W} \cap \overline{W}^{\mathsf{T}}$ and the equivalence classes of R are linearly ordered by the order of the representatives via W. Informally this means that their Hasse-diagrams are composed by a series of complete bipartite strict-orders, one above another. Due to this property, weak-order relations are often used to model preferences with indifference, for instance in mathematical psychology. See [10, 17] for example.

Each weak-order relation $W : X \leftrightarrow X$ can be transformed into an upper right block triangle form. From the single blocks of this form the above mentioned complete bipartite strict-orders and their rearrangement immediately becomes apparent. To obtain a permutation relation on X that transforms W into an upper right block triangle form is possible by performing three steps. First, W is joined with the identity relation $\mathsf{I} : X \leftrightarrow X$. The resulting reflexive closure $E = W \cup \mathsf{I}$ of W is a partial order relation on X. Next, a linear extension E' of E is determined. And, finally, a permutation relation $P : X \leftrightarrow X$ is computed that rearranges the linear strict-order relation $E' \cap \overline{\mathsf{I}}$ into the full upper right triangle $P^{\mathsf{T}}(E' \cap \overline{\mathsf{I}})P$. A little reflection shows that the same permutation relation also transforms the original weak-order relation W into the desired upper right block triangle form $P^{\mathsf{T}}WP$. The following RelView-program `PermWeakO` is a direct translation of the above steps into the programming language of the tool.

```
PermWeakO(W)
  DECL I
  BEG  I = I(W)
       RETURN PermLSO(Szpilrajn(W | I) & -I)
  END.
```

5.4 Semi-Order Relations

By definition, a *semi-order relation* $S : X \leftrightarrow X$ is irreflexive ($S \subseteq \overline{\mathsf{I}}$), *semi-transitive* ($SS\overline{S}^{\mathsf{T}} \subseteq S$ or, equivalently, $\overline{S}\,\overline{S} \subseteq \overline{SS}$), and possesses the *Ferrers* property $S\overline{S}^{\mathsf{T}}S \subseteq S$. The interest in this specific kind of orders mainly stems from the following fact, known as Scott-Suppes-Theorem (see [19, 11]): Let X be a finite set. Then $S : X \leftrightarrow X$ is a semi-order relation iff there exist a mapping $f : X \to \mathbb{R}$ and a positive $\rho \in \mathbb{R}$ such that the relationship $S_{x,y}$ and the estimation $f(x) + \rho < f(y)$ are equivalent for all $x, y \in X$. The constant ρ can be seen as some sort of threshold. It allows to model preferences with indifference by defining $x, y \in X$ as indifferent iff $|f(x) - f(y)| \leq \rho$.

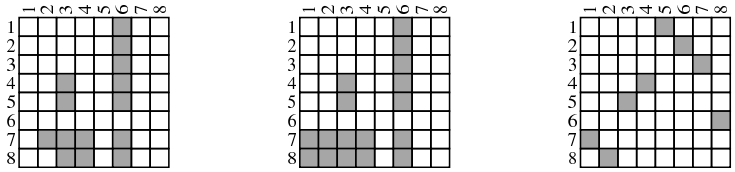
Irreflexive relations with the Ferrers property are called *interval order relations* since they are strict-orders that have interval representations. We will discuss this later in Sect. 6.2. In the case of semi-order relations the additional

assumption of semi-transitivity allows a representation with all intervals of the same positive length, e.g., of length 1.

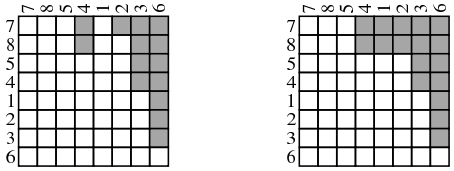
Each semi-order relation $S : X \leftrightarrow X$ can be rearranged into a threshold in an upper right block triangle form. As in the cases of the transformations of Sects. 5.1 to 5.3, the decisive step here again is the computation of a permutation relation $P : X \leftrightarrow X$ on the base set X that simultaneously transforms rows and columns via $P^{\mathsf{T}}SP$. To obtain P, we start with $W = S \cup \overline{S}^{\mathsf{T}} S \cup S \overline{S}^{\mathsf{T}}$. It is not very hard to verify relation-algebraically that W is a weak-order relation on X, which in turn yields the strict-order property for W. A little reflection furthermore shows that we can take as P any permutation relation that transforms W into an upper right block triangle form. In the syntax of RelView the entire procedure looks as follows:

```
PermSemiO(S)
  DECL W
  BEG  W = S | -S^*S | S*-S^
       RETURN PermWeakO(W)
  END.
```

One may think that in an analogous manner also interval order relations J may be handled by embedding them into some sort of a semi-order closure S, where $S = J \cup JJ\overline{J}^{\mathsf{T}}$, i.e., by adding whatever is missing for semi-transitivity. The relation-algebraic proof that S is indeed a semi-order relation is too long to be included here. But the result of the approach is not as expected. In the following three pictures we see on the left an interval order relation J, in the middle the semiorder relation $S = J \cup JJ\overline{J}^{\mathsf{T}}$, and on the right a permutation relation P that transforms S in an upper right block triangle form $P^{\mathsf{T}}SP$.



The next two pictures show the two rearranged relations, viz. the Boolean matrix of $P^{\mathsf{T}}JP$ on the left and that of $P^{\mathsf{T}}SP$ on the right.



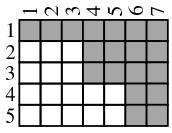
The transformed semi-order relation is in the desired form. However, for the transformed interval order relation we have some objections, as the Ferrers property is not yet visible. In the next section we will demonstrate that a visualization of the latter is also possible, however, at the cost of permuting rows and columns no longer simultaneously.

6 More Complex Rearranging Algorithms

Having developed some simple algorithms which rearrange the matrix representation of specific order relations appropriately by permuting rows and columns simultaneously, we now concentrate on two more complex cases. Here a transformation into an appropriate form requires rows and columns to be permuted independently. We will also show an example that once again exhibits the power of RelView when dealing with the visualization of relations.

6.1 Ferrers Relations

In the theory of partitions of numbers (see e.g., [5]), a partition of a natural number n into a sum $n = a_1 + a_2 + \ldots + a_k$, where $a_1 \geq a_2 \geq \ldots \geq a_k$, is frequently visualized by means of a Ferrers diagram. Such a diagram is drawn as a rectangular array of squares[1] in which the i'th row has the number of squares equal to the number a_i, $1 \leq i \leq k$. All rows are right-justified (or left-justified) and sorted by their lengths (i.e., the number of squares) in decreasing order from the top to the bottom. As a small example, the pictorial representation of the Ferrers diagram for the partition $19 = 7 + 4 + 4 + 2 + 2$ of the number 19 as a RelView-matrix looks as follows.



The black parts of the rows of this 5×7 Boolean matrix exactly correspond to the 5 rows of the Ferrers diagram.

Ferrers diagrams motivated the definition of Ferrers relations by demanding that the latter type of relations can be transformed into upper right staircase block form – the form shown in the above RelView-matrix, but additionally allowing empty columns to occur at the left or empty rows at the bottom – by permuting rows and columns. To clarify that this rearrangement property is equivalent to the simple relation-algebraic definition given in Sect. 5.4, a combination of the predicate logic formulation of the inclusion $R\overline{R}^{\mathsf{T}}R \subseteq R$ and a graph interpretation of R is very helpful; see [18] for details. Ferrers relations have a lot of applications in mathematics and computer science. Here we only want to mention their use in the theory of measurement (ranking via Guttmann scaling) and in formal concept analysis; see [12] and [14] for example.

The following two remarks further exhibit that Ferrers relations enjoy important properties: If $R : X \leftrightarrow Y$ is a Ferrers relation, then so are $R\overline{R}^{\mathsf{T}}R$, $R\overline{R}^{\mathsf{T}}$ as well as $\overline{R}^{\mathsf{T}}R$. For a finite Ferrers relation, there exists a natural number $k \geq 0$ that gives rise to a strictly increasing exhaustion as follows:

$$\mathsf{O} = (R\overline{R}^{\mathsf{T}})^k \subset (R\overline{R}^{\mathsf{T}})^{k-1} \subset \ldots \subset R\overline{R}^{\mathsf{T}}R\overline{R}^{\mathsf{T}} \subset R\overline{R}^{\mathsf{T}}$$

[1] Sometimes also boxes, dots or circles are used instead of squares.

Another characterization says that R has the Ferrers property iff the contains pre-order relation R / R on the rows (see Sect. 3) is linear. This applies for the is-contained pre-order relation $R \setminus R$ on the columns, too. The relation-algebraic proof of the first fact using the Schröder equivalences (see e.g., [18]) is simple:

$$R\overline{R}^{\mathsf{T}}R \subseteq R \Leftrightarrow \overline{R}R^{\mathsf{T}}\overline{R} \subseteq \overline{R} \Leftrightarrow R\overline{R}^{\mathsf{T}} \subseteq \overline{\overline{R}R^{\mathsf{T}}} \Leftrightarrow \overline{R / R} \subseteq (R / R)^{\mathsf{T}}$$

The latter inclusion means that R / R is indeed linear.

Now, let $R : X \leftrightarrow Y$ be a Ferrers relation for which we intend to develop a relation-algebraic specification of the row permutation relation $P_r : X \leftrightarrow X$ as well as the column permutation relation $P_c : Y \leftrightarrow Y$ such that ${P_r}^{\mathsf{T}}R$ rearranges the rows of R in decreasing inclusion order from the top to the bottom, and after that ${P_r}^{\mathsf{T}}RP_c$ rearranges the columns of the intermediate relation ${P_r}^{\mathsf{T}}R$ in increasing inclusion order from the left to the right. Obviously, one will choose the pre-order relation rearrangement of Sect. 5.2 based on the contains relation R / R for the rows and the is-contained relation $R \setminus R$ for the columns, respectively. The corresponding RelView-functions look as follows:

```
PermRFerr(R) = PermPreO(R / R).
PermCFerr(R) = PermPreO(R \ R).
```

Finally, the procedure for the upper right staircase block form of R is described by the following RelView-program:

```
RearrFerr(R)
  DECL Pr, Pc
  BEG  Pr = PermRFerr(R);
       Pc = PermCFerr(R)
       RETURN Pr^*R*Pc
  END.
```

6.2 Interval Order Relations

As already mentioned in Sect. 5.4, an interval order relation $J : X \leftrightarrow X$ is an irreflexive relation that possesses the Ferrers property. Hence, the relational programs for transforming Ferrers relations can also be applied to transform the relation J into an upper right staircase block form by permuting rows and columns independently. Since, however, interval order relations are also transitive (and, consequently, specific strict-orders), the 1-blocks of the staircase block form are completely contained in the upper right triangle. Hence, we obtain an upper right block triangle form, as in the case of the relations of Sects. 5.3 and 5.4.

Here is the RelView-function for the permutation of the rows of an interval order relation via a permutation relation $P_r : X \leftrightarrow X$ on the base set X.

```
PermRIntervalO(J) = PermRFerr(J).
```

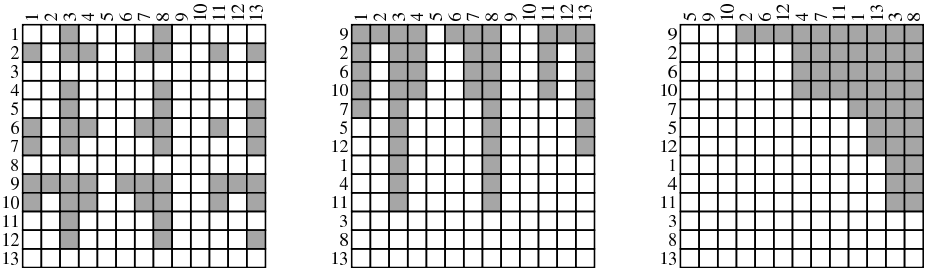
In exactly the same way we obtain two RelView-functions `PermCIntervalO` and `RearrIntervalO` for the permutation relation $P_c : X \leftrightarrow X$ that rearranges the columns of J and for the desired upper right block triangle form ${P_r}^{\mathsf{T}}JP_c$.

Each interval order relation can be represented by a set of intervals of a linearly ordered set, ordered by strict left-to-right precedence. Formally we have the following theorem (see [11]): $J : X \leftrightarrow X$ is an interval order relation iff there is a function $f : X \to 2^P$ that assigns to each $x \in X$ a closed interval $f(x) = [a_x, b_x] \subseteq P$ of a linearely ordered set $(P. \leq)$ such that for all $x, y \in X$ we have $J_{x,y}$ iff $b_x < a_y$. In case of $(\mathbb{R}, \leq)$ as $(P. \leq)$ one speaks of a *real interval representation*. Interval representations of interval order relations J via the rows of Boolean matrices may be specified by purely relation-algebraic expressions. The RelView-program resulting from this specification looks as follows; because of lack of space we cannot go into details.

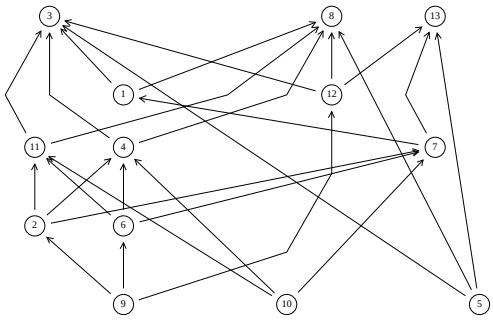
```
IntervalRepr(J)
  DECL fringe(R) = R & -(R*-R^*R);
       coleq(R) = syq(R,R);
       roweq(R) = syq(R^,R^);
       SR, SC, M, C, I, Pc
  BEG  SR = Classes(roweq(J));
       SC = Classes(coleq(J));
       M = SR^*fringe(-J)*SC;
       C = M^*SR^*J*SC;
       I = I(C);
       Pc = PermLSO(C)
       RETURN SR*M*(C | I)^*Pc & SC*(C | I)*Pc
  END.
```

6.3 An Example

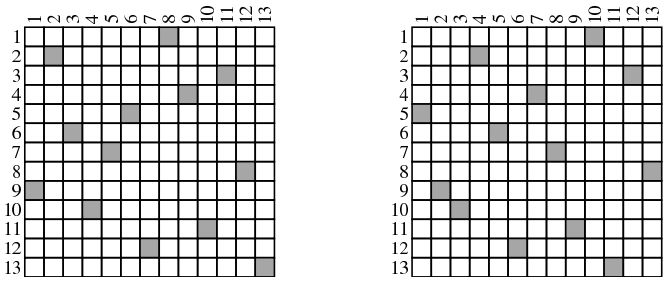
Now, let us present a small application of the programs of Sect. 6.2. We consider the following three 13×13 RelView-matrices.



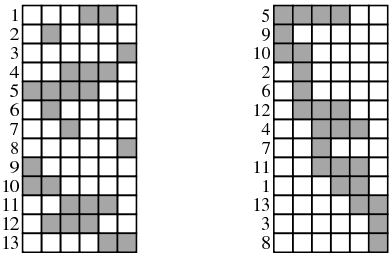
It can easily be checked that the three Boolean matrices represent the same interval order relation R on a 13-element base set. The matrix on the left shows the order's original version, the matrix in the middle the version after sorting the rows in decreasing inclusion order from the top to the bottom, and the matrix on the right the final version, which is an upper right block triangle form. From the latter Boolean matrix we immediately obtain a layered graph-representation of R. The next picture shows the Hasse-diagram of R, drawn by RelView's implementation of the graph drawing algorithm of [13] and slightly prettified by hand to enhance visibility of the minimal elements 5, 9 and 10.



From the attached row and column numbers of the above matrices we directly obtain the following permutation relations P_r for the rows (left matrix) and P_c for the columns (right matrix). Of course, the real way of computation was the other way around. We first computed the permutation relations P_r and P_c and then labeled the rearranged Boolean matrices according to them.



The left one of the following two Boolean matrices is the result of the RelView-program `IntervalRepr` applied to the interval order relation R, and the matrix right aside is generated from it by sorting the rows according to the first occurrence of the entry 1 via a small RelView-program.



If $\{y_1, y_2, y_3, y_4, y_5, y_6\}$ is the base set of the columns of these Boolean matrices and the linear base strict-order is given by the order of the element's indices, then, e.g., the interval $[y_4, y_5]$ is assigned to the element 1 of the 13-element base set and the singleton interval $[y_2, y_2]$ is assigned to the element 2 of this set. To obtain a real interval representation from the result of `IntervalRepr`, we first have to interpret each black square as a unit interval on the real line, where the left border of the matrix describes 0. This does not yet yield the desired

result since it allows comparable intervals to be tangent, whereas the definition demands strict left-to-right precedence. But it is very easy to transform it into a real interval representation. We only have to shorten each interval from the right by a small constant. If, after that, each copy of a multiple interval is accordingly shortened from the left to get different left end points, we even obtain a so-called distinguishing real interval representation. Doing so, e.g., each of the two above matrices yields such a representation $f(i) = \alpha_i$, where the intervals $\alpha_i, 1 \leq i \leq 13$, are given as follows:

$$\begin{array}{lllll} \alpha_1 = [3, 4.9] & \alpha_2 = [1, 1.9] & \alpha_3 = [5, 5.9] & \alpha_4 = [2, 4.9] & \alpha_5 = [0, 3.9] \\ \alpha_6 = [1.1, 1.9] & \alpha_7 = [2, 2.9] & \alpha_8 = [5.1, 5.9] & \alpha_9 = [0, 0.9] & \alpha_{10} = [0, 1.9] \\ \alpha_{11} = [2, 4.9] & \alpha_{12} = [1, 3.9] & \alpha_{13} = [4, 5.9] & & \end{array}$$

Here we have shortened each interval from the right by 0.1 and, in addition, two intervals from the left by 0.1 to obtain uniqueness.

7 Conclusion

We have investigated how to visualize relations represented as Boolean matrices using the specific purpose Computer Algebra system RelView. We were able to rearrange Boolean matrices into specific forms which allow to discover structural properties that are not evident in the first place. Such an approach may be useful in various application fields. Starting with basic rearrangement algorithms, we constructed algorithms also for non-trivial cases.

In addition to the types of relations treated in the present paper, we have transformed several other simple types, like injective and univalent relations and (partial) equivalence relations. Relations of the first type can be rearranged into four blocks, where the left upper block looks like an identity relation and the remaining three blocks are empty. Equivalence relations can be transformed into block diagonal form with quadratic blocks of 1's in the diagonal and 0's otherwise. From the latter form one immediately obtains the equivalence classes. In the case of partial equivalence relations we additionally obtain a right lower empty block in the diagonal. We could, due to limited space, not present how to apply our technique to other complex examples. In particular, we have studied the "maximum pair of independent sets rearrangements" of general relations R based on a maximum matching λ contained in R. They are closely related to the term rank of a relation. We have also developed algorithms where quotients are taken according to some congruence relations on the base sets. An example for such a problem is the rearrangement of a relation R according to its so-called difunctional closure $R(R^\mathsf{T}R)^+$ into block diagonal form, where the difunctional closure has rectangular blocks of 1's in the diagonal and 0's otherwise.

A future aim is to use the efficiency and visualization power of RelView and the conceptual simplicity of its programming language so as to enable one to scan any given – even real-valued – matrix of moderate size for possibly hidden interesting properties. In the real-valued case, one would use moving so-called cuts at different levels to arrive at Boolean matrices similar to the cuts used in

the theory of fuzzy sets. Because of their close relationship we are also interested in the relation-algebraic treatment of interval graphs.

References

1. Behnke R., Berghammer R., Meyer E., Schneider P.: RelView – A system for calculation with relations and relational programming. In: Astesiano E. (ed.): Proc. 1st Conf. *Fundamental Approaches to Software Engineering*, LNCS 1382, Springer, 318-321 (1998).
2. Berghammer R., Hoffmann T.: Modelling sequences within the RelView system. J. Universal Comput. Sci. 7, 107-123 (2001).
3. Berghammer R., Leoniuk B., Milanese U.: Implementation of relation algebra using binary decision diagrams. In: de Swart H. (ed.): Proc. 6th Int. Workshop *Relational Methods in Computer Science*, LNCS 2561, Springer, 241-257 (2002).
4. Berghammer R., Neumann F.: RelView – An OBDD-based Computer Algebra system for relations. In: Gansha V.G., Mayr E.W., Vorozhtsov E. (eds.): Proc. 8th Int. Workshop *Computer Algebra in Scientific Computing*, LNCS 3718, Springer, 40-51 (2005).
5. Bona M.: A walk through combinatorics: An introduction to combinatorics and graph theory. World Scientific Publishing (2002).
6. Colorni A., Paruccini M., Roy B.: A-MCD-A – Multiple criteria decision aiding. Joint Research Centre of the European Commission (2001).
7. de Swart H., Orłowska E., Schmidt G., Roubens M. (eds.): Theory and Applications of Relational Structures as Knowledge Instruments. LNCS 2929, Springer (2003).
8. de Swart H., Orłowska E., Schmidt G., Roubens M. (eds.): Theory and Applications of Relational Structures as Knowledge Instruments II. LNAI 4342, Springer (2006).
9. di Battista G., Eades P., Tamassia R., Tollis I.G.: Graph drawing – Algorithms for the visualization of graphs. Prentice Hall (1999).
10. Fishburn P.: On the construction of weak orders from fragmentary information. Psychometrika 38, 459-472 (1973).
11. Fishburn P.: Interval orders and interval graphs. Wiley (1985).
12. Guttmann L.: A basis for scaling qualitative data. American Sociological Review 9, 139-150 (1944).
13. Gansner E.R., North S.C., Vo K.P.: DAG – A program that draws directed graphs. Software – Practice and Experience 17, 1047-1062 (1988).
14. Ganter B., Wille R.: Formal concept analysis, Springer (1999).
15. Kamada T., Kawai S.: An algorithm for drawing general undirected graphs. Inf. Proc. Letters 31, 7-15 (1989).
16. Moret B.M.E., Shapiro H.D.D.: Algorithms and experiments: The new (and old) methodology. J. Universal Comput. Sci. 7, 434-446 (2001).
17. Öztürk M., Tsoukias A.,Vincke P.: Preference modelling. In: Ehrgott M., Greco S., Figueira J. (eds.): Multiple criteria decision analysis: State of the art surveys, Int. Series in Operat. Res. and Manag. Sci. 78, Springer, 27-71 (2005).
18. Schmidt G., Ströhlein T.: Relationen und Graphem. Springer (1989). Available also in English: Relations and graphs. EATCS Monographs on Theoret. Comput. Sci., Springer (1993).
19. Scott D., Suppes P.: Foundational aspects of the theories of measurement. J. Symbolic Logic 23, 113-128 (1958).
20. Szpilrajn E.: Sur l'extension de l'ordre partiel. Fund. Math. 16, 386-389 (1930).
21. Tarski A.: On the calculus of relations. J. Symbolic Logic 6, 73-89 (1941).