`socialranking`: A package for evaluating ordinal power relations in cooperative game theory

Jochen Staudacher, Stefano Moretti, Felix Fritz
(Hochschule Kempten, Université Paris Dauphine)
Contact: felix.fritz@dauphine.eu

2023-08-23

Abstract

This document gives a brief introduction to power relations and social ranking solutions aimed at ranking elements based on their contributions within coalitions. This document accompanies version 1.0.0 of the package `socialranking`.
Keywords: power relation, social ranking solution, cooperative game theory, dominance, cp-majority, Copeland, Kramer-Simpson, ordinal Banzhaf, dominance, lexicographical excellence, relations.

Contents

1 Introduction

In the literature of cooperative games, the notion of *power index* [1–3] has been widely studied to analyze the "influence" of individuals taking into account their ability to force a decision within groups or coalitions. In practical situations, however, the information concerning the strength of coalitions is hardly quantifiable. So, any attempt to numerically represent the influence of groups and individuals clashes with the complex and multi-attribute nature of the problem and it seems more realistic to represent collective decision-making mechanisms using an ordinal coalitional framework based on two main ingredients: a binary relation over groups or coalitions and a ranking over the individuals.

The main objective of the package `socialranking` is to provide answers for the general problem of how to compare the elements of a finite set N given a ranking over the elements of its power-set (the set of all possible subsets of N). To do this, the package `socialranking` implements a portfolio of solutions from the recent literature on *social rankings* [4–10].

1.1 Quick start

A *power relation* (i.e, a ranking over subsets of a finite set N; see the Section 2 for a formal definition) can be constructed using the functions `PowerRelation()` or `as.PowerRelation()`.

```
library(socialranking)
PowerRelation(list(list(c(1,2)), list(1, c()), list(2)))
## 12 > (1 ~ {}) > 2
```

```
as.PowerRelation("12 > 1 ~ {} > 2")
## 12 > (1 ~ {}) > 2
```

```
as.PowerRelation("ab > a ~ {} > b")
## ab > (a ~ {}) > b
```

```
as.PowerRelation(list(c(1,2), 1, c(), 2))
## 12 > 1 > {} > 2
```

```
as.PowerRelation(list(c(1,2), 1, c(), 2), comparators = c(">", "~", ">"))
## 12 > (1 ~ {}) > 2
```

Functions used to analyze a given `PowerRelation` object can be grouped into three main categories:

- *Comparison* functions, only comparing two elements;
- *Score* functions, calculating the scores for each element;
- *Ranking* functions, creating `SocialRanking` objects.

Comparison and score functions are often used to evaluate a social ranking solution (see section 2 for a formal definition). Listed below are some of the most prominent functions and solutions introduced in the aforementioned papers.

These functions may be called as follows.

Comparison functions	Score functions	Ranking functions
`dominates()`		
`cumulativelyDominates()`	`cumulativeScores()`	
`cpMajorityComparison()` `cpMajorityComparisonScore()`	`copelandScores()` `kramerSimpsonScores()`	`copelandRanking()` `kramerSimpsonRanking()`
	`lexcelScores()`	`lexcelRanking()` `dualLexcelRanking()`
	`L1Scores()`	`L1Ranking()`
	`ordinalBanzhafScores()`	`ordinalBanzhafRanking()`

```
pr <- as.PowerRelation("ab > abc ~ ac ~ bc > a ~ c > {} > b")

# a dominates b, but b does not dominate a
c(dominates(pr, "a", "b"),
  dominates(pr, "b", "a"))
## [1]  TRUE FALSE

# calculate cumulative scores
scores <- cumulativeScores(pr)
# show score of element a
scores$a
## [1] 1 3 4 4 4

# performing a bunch of rankings
lexcelRanking(pr)
## a > b > c

L1Ranking(pr)
## a > b > c

dualLexcelRanking(pr)
## a > c > b

copelandRanking(pr)
## a > b ~ c

kramerSimpsonRanking(pr)
## a > b ~ c

ordinalBanzhafRanking(pr)
## a > c > b
```

Lastly, an incidence matrix for all given coalitions can be constructed using `powerRelationMatrix(pr)` or `as.relation(pr)` from the `relations` package [11]. The incidence matrix may be displayed using `relations::relation_incidence()`.

```
rel <- relations::as.relation(pr)
rel
## A binary relation of size 8 x 8.
```

```
relations::relation_incidence(rel)
## Incidences:
##     ab abc ac bc a c {} b
## ab   1   1  1  1 1 1  1 1
## abc  0   1  1  1 1 1  1 1
## ac   0   1  1  1 1 1  1 1
## bc   0   1  1  1 1 1  1 1
## a    0   0  0  0 1 1  1 1
## c    0   0  0  0 1 1  1 1
## {}   0   0  0  0 0 0  1 1
## b    0   0  0  0 0 0  0 1
```

2 `PowerRelation` objects

We first introduce some basic definitions on binary relations. Let X be a set. A set $R \subseteq X \times X$ is said a *binary relation* on X. For two elements $x, y \in X$, xRy refers to their relation, more formally it means that $(x, y) \in R$. A binary relation $(x, y) \in R$ is said to be

- *reflexive*, if for each $x \in X, xRx$,
- *transitive*, if for each $x, y, z \in X, xRy$ and $yRz \Rightarrow xRz$,
- *total*, if for each $x, y \in X, x \neq y \Rightarrow xRy$ or yRx,
- *symmetric*, if for each $x, y \in X, xRy \Leftrightarrow yRx$,
- *asymmetric*, if for each $x, y \in X, (x, y) \in R \Rightarrow (y, x) \notin R$, and
- *antisymmetric*, if for each $x, y \in X, xRy \cap yRx \Rightarrow x = y$.

A *preorder* is defined as a reflexive and transitive relation. If it is total, it is called a *total preorder*. Additionally if it is antisymmetric, it is called a *linear order*.

Let $N = \{1, 2, \ldots, n\}$ be a finite set of *elements*, sometimes also called *players*. For some $p \in \{1, \ldots, 2^n\}$, let $\mathcal{P} = \{S_1, S_2, \ldots, S_p\}$ be a set of *coalitions* such that $S_i \subseteq N$ for all $i \in \{1, \ldots, p\}$. Thus $\mathcal{P} \subseteq 2^N$, where 2^N denotes the power set of N, the set of all subsets or coalitions of N,

$$S \in 2^N \quad \Leftrightarrow \quad S \subseteq N.$$

$\mathcal{T}(N)$ denotes the set of all total preorders on N, $\mathcal{T}(\mathcal{P})$ the set of all total preorders on $\mathcal{P}$. A single total preorder $\succeq \in \mathcal{T}(\mathcal{P})$ is said a *power relation*.

In a given power relation $\succeq \in \mathcal{T}(\mathcal{P})$ on $\mathcal{P} \subseteq 2^N$, its symmetric part is denoted by $\sim$ (i.e., $S \sim T$ if $S \succeq T$ and $T \succeq S$), whereas its asymmetric part is denoted by $\succ$ (i.e., $S \succ T$ if $S \succeq T$ and not $T \succeq S$). In other terms, for $S \sim T$ we say that S is *indifferent* to T, whereas for $S \succ T$ we say that S is *strictly better* than T.

Lastly for a given power relation in the form of $S_1 \succeq S_2 \succeq \ldots \succeq S_m$, coalitions that are indifferent to one another can be grouped into *equivalence classes* $\sum_i$ such that we get the *quotient order* $\sum_1 \succ \sum_2 \succ \ldots \succ \sum_m$.

Example 1. Let $N = \{1, 2\}$ be two players with its corresponding power set $2^N = \{\{1, 2\}, \{1\}, \{2\}, \emptyset\}$. The following power relation is given:

$$\begin{aligned}\succeq = \{&(\{1,2\},\{1,2\}), \quad (\{1,2\},\{2\}), \quad (\{1,2\},\emptyset), \quad (\{1,2\},\{1\}),\\ &(\{2\},\{2\}), \quad (\{2\},\emptyset), \quad (\{2\},\{1\}),\\ &(\emptyset,\emptyset), \quad (\emptyset,\{2\}), \quad (\emptyset,\{1\}),\\ &(\{1\},\{1\})\ \}\end{aligned}$$

This power relation can be rewritten in a consecutive order as: $\{1, 2\} \succ \{2\} \sim \emptyset \succ \{1\}$. Its quotient order is formed by three equivalence classes $\sum_1 = \{\{1, 2\}\}, \sum_2 = \{\{2\}, \emptyset\}$, and $\sum_3 = \{\{1\}\}$; so the quotient order of $\succeq$ is such that $\{\{1, 2\}\} \succ \{\{2\}, \emptyset\} \succ \{\{1\}\}$.

Note that the way the set $\succeq$ is presented in the example is somewhat deliberate to better visualize occurring symmetries and asymmetries. This also lets us neatly represent a power relation in the form of an incidence matrix in chapter 4.

2.1 Creating `PowerRelation` objects

A power relation in the `socialranking` package is defined to be reflexive, transitive and total. In designing the package it was deemed logical to have the coalitions specified in a consecutive order, as seen in Example 1. Each coalition in that order is split either by a `">"` (left side strictly better) or a `"~"` (two coalitions indifferent to one another). The following code chunk shows the power relation from Example 1 and how a correlating `PowerRelation` object can be constructed.

```
library(socialranking)
pr <- PowerRelation(list(
  list(c(1,2)),
  list(2, c()),
  list(1)
))
pr
## 12 > (2 ~ {}) > 1
```

```
class(pr)
## [1] "PowerRelation"      "SingleCharElements"
```

Notice how coalitions such as $\{1, 2\}$ are written as `12` to improve readability. Similarly, passing a string to the function `as.PowerRelation()` saves some typing on the user's end by interpreting each character of a coalition as a separate element. Note that spaces in that function are ignored.

```
as.PowerRelation("12 > 2~{} > 1")
## 12 > (2 ~ {}) > 1
```

The compact notation is only done in `PowerRelation` objects where every element is one digit or one character long. If this is not the case, curly braces and commas are added where needed.

```
prLong <- PowerRelation(list(
  list(c("Alice", "Bob")),
  list("Bob", c()),
```

Attribute	Description	Value in `pr`
`elements`	Sorted vector of elements	`c(1,2)`
`eqs`	List containing lists, each containing coalitions in the same equivalence class	`list(list(c(1,2)), list(c(2), c()), list(c(1)))`
`coalitionLookup`	Function to determine a coalition's equivalence class index	`function(coalition)`
`elementLookup`	Function to determine, which coalitions an element takes part in	`function(element)`

```
  list("Alice")
))
prLong
## {Alice, Bob} > ({Bob} ~ {}) > {Alice}
```

```
class(prLong)
## [1] "PowerRelation"
```

Some may have spotted a `"SingleCharElements"` class missing in `class(prLong)` that has been there in `class(pr)`. `"SingleCharElements"` influences how coalitions are printed. If it is removed from `class(pr)`, the output will include the same curly braces and commas displayed in `prLong`.

```
class(pr) <- class(pr)[-which(class(pr) == "SingleCharElements")]
pr
## {1, 2} > ({2} ~ {}) > {1}
```

Internally a `PowerRelation` is a list with four attributes.

While coalitions are formally defined as sets, meaning the order doesn't matter and each element is unique, the package tries to stay flexible. As such, coalitions will only be sorted during initialization, but duplicate elements will not be removed.

```
prAtts <- PowerRelation(list(
  list(c(2,2,1,1,2)),
  list(c(2,1), c())
))
#! Warning in createLookupTables(equivalenceClasses): Found 1
coalition that contain elements more than once.
#!  - 1, 2 in the coalition {1, 1, 2, 2, 2}
```

```
prAtts
## 11222 > (12 ~ {})
```

```
prAtts$elements
## [1] 1 2
```

```
prAtts$coalitionLookup(c(1,2))
## [1] 2
```

```
prAtts$coalitionLookup(c(2,1))
## [1] 2
```

```
prAtts$coalitionLookup(c(2,1,2,1,2))
## [1] 1
```

```
prAtts$elementLookup(2)
## [[1]]
## [1] 1 1
##
## [[2]]
## [1] 1 1
##
## [[3]]
## [1] 1 1
##
## [[4]]
## [1] 2 1
```

2.2 Manipulating `PowerRelation` objects

It is strongly discouraged to directly manipulate `PowerRelation` objects, as its attributes are so tightly coupled. This would require updates in multiple places. Instead, it is advisable to simply create new `PowerRelation` objects.

To permutate the order of equivalence classes, it is possible to take the equivalence classes in `$eqs` and use a vector of indexes to move them around.

```
pr <- as.PowerRelation("12 > (1 ~ {}) > 2")
PowerRelation(pr$eqs[c(2, 3, 1)])
## (1 ~ {}) > 2 > 12
```

```
PowerRelation(rev(pr$eqs))
## 2 > (1 ~ {}) > 12
```

For permutating individual coalitions, using `as.PowerRelation.list()` may be more convenient since it doesn't require nested list indexing.

```
coalitions <- unlist(pr$eqs, recursive = FALSE)
compares <- c(">", "~", ">")
as.PowerRelation(coalitions[c(2,1,3,4)], comparators = compares)
## 1 > (12 ~ {}) > 2
```

```
# notice that the length of comparators does not need to match
# length(coalitions)-1
as.PowerRelation(rev(coalitions), comparators = c("~", ">"))
## (2 ~ {}) > (1 ~ 12)
```

```
# not setting the comparators parameter turns it into a linear order
as.PowerRelation(coalitions)
## 12 > 1 > {} > 2
```

2.2.1 `appendMissingCoalitions()`

Let $\succeq \in \mathcal{T}(\mathcal{P})$. We may have not included all possible coalitions, such that $\mathcal{P} \subset 2^N, \mathcal{P} \neq 2^N$.

`appendMissingCoalitions()` appends all the missing coalitions $2^N - \mathcal{P}$ as a single equivalence class to the end of the power relation.

```
pr <- PowerRelation(list(
  list(c("AT", "DE"), "FR"),
  list("DE"),
  list(c("AT", "FR"), "AT")
))
pr
## ({AT, DE} ~ {FR}) > {DE} > ({AT, FR} ~ {AT})
```

```
# since we have 3 elements, the super set 2^N should include 8 coalitions
appendMissingCoalitions(pr)
## ({AT, DE} ~ {FR}) > {DE} > ({AT, FR} ~ {AT}) > ({AT, DE, FR} ~
{DE, FR} ~ {})
```

2.2.2 `makePowerRelationMonotonic()`

A power relation $\succeq \in \mathcal{T}(\mathcal{P})$ is monotonic if

$$S \succeq T \quad \Rightarrow \quad T \subset S$$

for all $S, T \subseteq N$. In other terms, given a monotonic power relation, for any coalition, all its subsets cannot be ranked higher.

`makePowerRelationMonotonic()` turns a potentially non-monotonic power relation into a monotonic one by moving and (optionally) adding all missing coalitions in $2^N - \mathcal{P}$ to the corresponding equivalence classes.

```
pr <- as.PowerRelation("a > b > c ~ ac > abc")
makePowerRelationMonotonic(pr)
## (abc ~ ab ~ ac ~ a) > (bc ~ b) > c
```

```
makePowerRelationMonotonic(pr, addMissingCoalitions = FALSE)
## (abc ~ ac ~ a) > b > c
```

```
# notice how an empty coalition in some equivalence class
# causes all remaining coalitions to be moved there
makePowerRelationMonotonic(as.PowerRelation("ab > c > {} > abc > a > b"))
## (abc ~ ab) > (ac ~ bc ~ c) > (a ~ b ~ {})
```

2.3 Creating power sets

As the number of elements n increases, the number of possible coalitions increases to $|2^N| = 2^n$. `createPowerset` is a convenient function that not only creates a power set 2^N which can be used to call `PowerRelation` or `as.PowerRelation`, but also formats the function call in such a way that makes it easy to rearrange the ordering of the coalitions.

RStudio offers shortcuts such as Alt+Up or Alt+Down (Option+Up or Option+Down on MacOS) to move one or multiple lines of code up or down (see fig. 1).

```
createPowerset(
  c("a", "b", "c"),
  result = "print"
)
## as.PowerRelation("
##   abc
##   > ab
##   > ac
##   > bc
##   > a
##   > b
##   > c
##   > {}
## ")
```

Figure 1: Using Alt+Up or Alt+Down to move one or more lines of code

By default, `createPowerset()` returns the power set in the form of a list. This list can be passed directly to `as.PowerRelation()` to create a linear order.

```
ps <- createPowerset(1:2, includeEmptySet = FALSE)
ps
## [[1]]
## [1] 1 2
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] 2
```

```
as.PowerRelation(ps)
## 12 > 1 > 2
```

```
# equivalent
PowerRelation(list(ps))
## (12 ~ 1 ~ 2)
```

```
as.PowerRelation(createPowerset(letters[1:4]))
## abcd > abc > abd > acd > bcd > ab > ac > ad > bc > bd > cd > a > b
> c > d > {}
```

2.4 Generating `PowerRelation` objects

Given a list of coalitions, it is possible to loop through all possible permutations of power relations using `powerRelationGenerator()`. Calling `gen()` in the example below always produces a unique `PowerRelation` object. If all permutations have been exhausted, `NULL` is returned.

```
coalitions <- list(c(1,2), 1, 2)
gen <- powerRelationGenerator(coalitions)
while(!is.null(pr <- gen())) {
  print(pr)
}
## (12 ~ 1 ~ 2)
## (12 ~ 1) > 2
## (12 ~ 2) > 1
## (1 ~ 2) > 12
## 12 > (1 ~ 2)
## 1 > (12 ~ 2)
## 2 > (12 ~ 1)
## 12 > 1 > 2
## 12 > 2 > 1
## 1 > 12 > 2
## 2 > 12 > 1
## 1 > 2 > 12
## 2 > 1 > 12
```

Permutations over power relations can be split into two parts:

1. generating partitions, or, generating differently sized equivalence classes, and
2. moving coalitions between these partitions.

In the code example above, we started with a single partition of size three, wherein all coalitions are considered equally preferable. By the end, we have reached the maximum number of partitions, where each coalition is put inside an equivalence class of size 1.

The partition generation can be reversed, such that we first receive linear power relations.

```
gen <- powerRelationGenerator(coalitions, startWithLinearOrder = TRUE)
while(!is.null(pr <- gen())) {
  print(pr)
}
## 12 > 1 > 2
## 12 > 2 > 1
## 1 > 12 > 2
## 2 > 12 > 1
## 1 > 2 > 12
## 2 > 1 > 12
## 12 > (1 ~ 2)
## 1 > (12 ~ 2)
## 2 > (12 ~ 1)
## (12 ~ 1) > 2
## (12 ~ 2) > 1
## (1 ~ 2) > 12
## (12 ~ 1 ~ 2)
```

Notice that the "moving coalitions" part was not reversed, only the order the partitions come in.

Similarly, we are also able to skip the current partition.

```
gen <- powerRelationGenerator(coalitions)
# partition 3

gen <- generateNextPartition(gen)
# partition 2+1

gen <- generateNextPartition(gen)
# partition 1+2
gen()
## 12 > (1 ~ 2)
```

Note: the number of possible power relations grows tremendously fast as the number of coalitions rises. To get to that number, first consider how many ways n coalitions can be split into k partitions, also known as the Stirling number of second kind,

$$S(n,k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (k-j)^n.$$

The number of all possible partitions given n coalitions is known as the Bell number (see also `numbers::bell()`),

$$B_n = \sum_{j=0}^{k} S(n,k).$$

Given a set of coalitions $\mathcal{P} \in 2^N$, the number of total preorders in $\mathcal{T}(\mathcal{P})$ is

$$|\mathcal{T}(\mathcal{P})| = \sum_{k=0}^{|\mathcal{P}|} k! \ * \ S(|\mathcal{P}|, k)$$

# of coalitions	# of partitions	# of total preorders
1	1	1
2	2	3
3	5	13
4	15	75
5	52	541
6	203	4.683
7	877	47.293
8	4.140	545.835
9	21.147	7.087.261
10	115.975	102.247.563
11	678.570	1.622.632.573
12	4.213.597	28.091.567.595

3 SocialRanking Objects

The main goal of the `socialranking` package is to rank elements based on a given power ranking. More formally we try to map $R : \mathcal{T}(\mathcal{P}) \to \mathcal{T}(N)$, associating to each power relation $\succeq \in \mathcal{T}(\mathcal{P})$ a total preorder $R(\succeq)$ (or $R^{\succeq}$) over the elements of N.

In this context $iR^{\succeq}j$ tells us that, given a power relation $\succeq$ and applying a social ranking solution $R(\succeq)$, i is ranked higher than or equal to j. From here on out, `>` and `~` also denote the asymmetric and the symmetric part of a social ranking, respectively, i `>` j indicating that i is strictly better than j, whereas in i `~` j, i is indifferent to j.

In literature, $iI^{\succeq}j$ and $iP^{\succeq}j$ are often used to denote the symmetric and asymmetric part, respectively. $iI^{\succeq}j$ therefore means that $iR^{\succeq}j$ and $jR^{\succeq}i$, whereas $iP^{\succeq}j$ implies that $iR^{\succeq}j$ but not $jR^{\succeq}j$.

In section 3.1 we show how a general `SocialRanking` object can be constructed using the `doRanking` function. In the following sections, we will introduce the notion of dominance[4], cumulative dominance[12] and CP-Majority comparison[6] that lets us compare two elements before diving into the social ranking solutions of the Ordinal Banzhaf Index[5], Copeland-like and Kramer-Simpson-like methods[10], and lastly the Lexicographical Excellence Solution[9] (Lexcel) and the Dual Lexicographical Excellence solution[13] (Dual Lexcel).

Example 2. Let $\{a, b\} \succ (\{a, c\} \sim \{b, c\}) \succ (\{a\} \sim \{c\}) > \emptyset \succ \{b\}$ be a power ranking. Using the following social ranking solutions, we get:

- `a > b > c` for `lexcelRanking`
- `a > c > b` for `dualLexcelRanking`
- `a > b ~ c` for `copelandRanking` and `kramerSimpsonRanking`
- `a ~ c > b` for `ordinalBanzhafRanking`

3.1 Creating SocialRanking objects

A `SocialRanking` object represents a total preorder in $\mathcal{T}(N)$ over the elements of N. Internally they are saved as a list of vectors, each containing players that are indifferent to one another. This is somewhat similar to the `equivalenceClasses` attribute in `PowerRelation` objects.

The function `doRanking` offers a generic way of creating `SocialRanking` objects. Given a sortable vector or list of scores it determines the power relation between all players, where the names of the elements are determined from the `names()` attribute of `scores`. Hence, a `PowerRelation` object is not necessary to create a `SocialRanking` object.

```
# we define some arbitrary score vector where "a" scores highest.
# "b" and "c" both score 1, thus they are indifferent.
scores <- c(a = 100, b = 1, c = 1)
doRanking(scores)
```

```
## a > b ~ c
```

```
# we can also tell doRanking to punish higher scores
doRanking(scores, decreasing = FALSE)
```

```
## b ~ c > a
```

When working with types that cannot be sorted (i.e., `lists`), a function can be passed to the `compare` parameter that allows comparisons between arbitrary elements. This

function must take two parameters (i.e., `a` and `b`) and return a numeric value based on the comparison:

- `compare(a,b) > 0`: `a` scores higher than `b`,
- `compare(a,b) < 0`: `a` scores lower than `b`,
- `compare(a,b) == 0`: `a` and `b` are equivalent.

```
scores <- list(a = c(3, 3, 3), b = c(2, 3, 2), c = c(7, 0, 2))
doRanking(scores, compare = function(a, b) sum(a) - sum(b))
## a ~ c > b

# a and c are considered to be indifferent, because their sums are the same

doRanking(scores, compare = function(a,b) sum(a) - sum(b), decreasing = FALSE)
## b > a ~ c
```

3.2 Comparison Functions

Comparison functions only compare two elements in a given power relation. They do not offer a social ranking solution. However in cases such as CP-Majority comparison, those comparison functions may be used to construct a social ranking solution in some particular cases.

3.2.1 Dominance

Definition 1. (Dominance [4]) Given a power relation $\succeq \in \mathcal{T}(\mathcal{P})$ and two elements $i, j \in N$, i *dominates* j in $\succeq$ if $S \cup \{i\} \succeq S \cup \{j\}$ for each $S \in 2^{N \setminus \{i,j\}}$. i also *strictly* dominates j if there exists $S \in 2^{N \setminus \{i,j\}}$ such that $S \cup \{i\} \succ S \cup \{j\}$.

The implication is that for every coalition i and j can join, i has at least the same positive impact as j.

The function `dominates(pr, e1, e2)` only returns a logical value `TRUE` if `e1` dominates `e2`, else `FALSE`. Note that `e1` not dominating `e2` does *not* indicate that `e2` dominates `e1`, nor does it imply that `e1` is indifferent to `e2`.

```
pr <- as.PowerRelation("3 > 1 > 2 > 12 > 13 > 23")

# 1 clearly dominates 2
dominates(pr, 1, 2)
## [1] TRUE

dominates(pr, 2, 1)
## [1] FALSE

# 3 does not dominate 1, nor does 1 dominate 3, because
# {}u3 > {}u1, but 2u1 > 2u3
dominates(pr, 1, 3)
## [1] FALSE

dominates(pr, 3, 1)
## [1] FALSE
```

```r
# an element i dominates itself, but it does not strictly dominate itself
# because there is no Sui > Sui
dominates(pr, 1, 1)
## [1] TRUE
```

```r
dominates(pr, 1, 1, strictly = TRUE)
## [1] FALSE
```

For any $S \in 2^{N \setminus \{i,j\}}$, we can only compare $S \cup \{i\} \succeq S \cup \{j\}$ if both $S \cup \{i\}$ and $S \cup \{j\}$ take part in the power relation.

Additionally, for $S = \emptyset$, we also want to compare $\{i\} \succeq \{j\}$. In some situations however a comparison between singletons is not desired. For this reason the parameter `includeEmptySet` can be set to `FALSE` such that $\emptyset \cup \{i\} \succeq \emptyset \cup \{j\}$ is not considered in the CP-Majority comparison.

```r
pr <- as.PowerRelation("ac > bc ~ b > a ~ abc > ab")

# FALSE because ac > bc, whereas b > a
dominates(pr, "a", "b")
## [1] FALSE
```

```r
# TRUE because ac > bc, ignoring b > a comparison
dominates(pr, "a", "b", includeEmptySet = FALSE)
## [1] TRUE
```

3.2.2 Cumulative Dominance

When comparing two players $i, j \in N$, instead of looking at particular coalitions $S \in 2^{N \setminus \{i,j\}}$ they can join, we look at how many stronger coalitions they can form at each point. This property was originally introduced in [12] as a regular dominance axiom.

For a given power relation $\succeq \in \mathcal{T}(\mathcal{P})$ and its corresponding quotient order $\sum_1 \succ \cdots \succ \sum_m$, the power of a player i is given by a vector $\text{Score}_{\text{Cumul}}(i) \in \mathbb{N}^m$ where we cumulatively sum the amount of times i appears in $\sum_k$ for each index k.

Definition 2. (Cumulative Dominance Score) Given a power relation $\succeq \in \mathcal{T}(\mathcal{P})$ and its quotient order $\sum_1 \succ \cdots \succ \sum_m$, the cumulative score vector $\text{Score}_{\text{Cumul}}(i) \in \mathbb{N}^m$ of an element $i \in N$ is given by:

$$\text{Score}_{\text{Cumul}}(i) = \left(\sum_{t=1}^{k} |\{S \in \textstyle\sum_t : i \in S\}| \right)_{k \in \{1,\ldots,m\}} \tag{1}$$

Definition 3. (Cumulative Dominance) Given two elements $i, j \in N$, i *cumulatively dominates* j in $\succeq$, if $\text{Score}_{\text{Cumul}}(i)_k \geq \text{Score}_{\text{Cumul}}(j)_k$ for each $k \in \{1, \ldots, m\}$. i also *strictly* cumulatively dominates j if there exists a k such that $\text{Score}_{\text{Cumul}}(i)_k > \text{Score}_{\text{Cumul}}(j)_k$.

For a given `PowerRelation` object `pr` and two elements `e1` and `e2`, `cumulativeScores(pr)` returns the vectors described in definition 2 for each element, `cumulativelyDominates(pr, e1, e2)` returns `TRUE` or `FALSE` based on definition 3.

```
pr <- as.PowerRelation("ab > (ac ~ bc) > (a ~ c) > {} > b")
cumulativeScores(pr)
```

```
## $a
## [1] 1 2 3 3 3
##
## $b
## [1] 1 2 2 2 3
##
## $c
## [1] 0 2 3 3 3
##
## attr(,"class")
## [1] "CumulativeScores"
```

```
# for each index k, $a[k] >= $b[k]
cumulativelyDominates(pr, "a", "b")
```

```
## [1] TRUE
```

```
# $a[3] > $b[3], therefore a also strictly dominates b
cumulativelyDominates(pr, "a", "b", strictly = TRUE)
```

```
## [1] TRUE
```

```
# $b[1] > $c[1], but $c[3] > $b[3]
# therefore neither b nor c dominate each other
cumulativelyDominates(pr, "b", "c")
```

```
## [1] FALSE
```

```
cumulativelyDominates(pr, "c", "b")
```

```
## [1] FALSE
```

Similar to the dominance property from our previous section, two elements not dominating one or the other does not indicate that they are indifferent.

3.2.3 CP-Majority comparison

The Ceteris Paribus Majority (CP-Majority) relation is a somewhat relaxed version of the dominance property. Instead of checking if $S \cup \{i\} \succeq S \cup \{j\}$ for all $S \in 2^{N \setminus \{i,j\}}$, the CP-Majority relation $iR^{\succeq}_{\text{CP}}j$ holds if the number of times $S \cup \{i\} \succeq S \cup \{j\}$ is greater than or equal to the number of times $S \cup \{j\} \succeq S \cup \{i\}$.

Definition 4. (CP-Majority [6]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Ceteris Paribus majority* relation is the binary relation $R^{\succeq}_{\text{CP}} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR^{\succeq}_{\text{CP}}j \Leftrightarrow d_{ij}(\succeq) \geq d_{ji}(\succeq) \tag{2}$$

where $d_{ij}(\succeq)$ represents the cardinality of the set $D_{ij}(\succeq)$, the set of all coalitions $S \in 2^{N \setminus \{i,j\}}$ for which $S \cup \{i\} \succeq S \cup \{j\}$.

`cpMajorityComparisonScore(pr, e1, e2)` calculates the two scores $d_{ij}(\succeq)$ and $-d_{ji}(\succeq)$. Notice the minus sign - that way we can use the sum of both values to determine the relation between `e1` and `e2`.

```
pr <- as.PowerRelation("ab > (ac ~ bc) > (a ~ c) > {} > b")
cpMajorityComparisonScore(pr, "a", "b")
## [1]  2 -1
```

```
cpMajorityComparisonScore(pr, "b", "a")
## [1]  1 -2
```

```
if(sum(cpMajorityComparisonScore(pr, "a", "b")) >= 0) {
  print("a >= b")
} else {
  print("b > a")
}
## [1] "a >= b"
```

As a slight variation the logical parameter `strictly` calculates $d_{ij}(\succ)$ and $-d_{ji}(\succ)$, the number of coalitions $S \in 2^{N\setminus\{i,j\}}$ where $S \cup \{i\} \succ S \cup \{j\}$.

```
# Now (ac ~ bc) is not counted
cpMajorityComparisonScore(pr, "a", "b", strictly = TRUE)
## [1] 1 0
```

```
# Notice that the sum is still the same
sum(cpMajorityComparisonScore(pr, "a", "b", strictly = FALSE)) ==
  sum(cpMajorityComparisonScore(pr, "a", "b", strictly = TRUE))
## [1] TRUE
```

Coincidentally, `cpMajorityComparisonScore` with `strictly = TRUE` can be used to determine if `e1` (strictly) dominates `e2`.

`cpMajorityComparisonScore` should be used for simple and quick calculations. The more comprehensive function `cpMajorityComparison(pr, e1, e2)` does the same calculations, but in the process retains more information about all the comparisons that might be interesting to a user, i.e., the set $D_{ij}(\succeq)$ and $D_{ji}(\succeq)$ as well as the relation $iR^{\succeq}_{\text{CP}}j$. See the documentation for a full list of available data.

```
# extract more information in cpMajorityComparison
cpMajorityComparison(pr, "a", "b")
## a > b
## D_ab = {c, {}}
## D_ba = {c}
## Score of a = 2
## Score of b = 1
```

```
# with strictly set to TRUE, coalition c does
# neither appear in D_ab nor in D_ba
cpMajorityComparison(pr, "a", "b", strictly = TRUE)
## a > b
## D_ab = {{}}
## D_ba = {}
## Score of a = 1
## Score of b = 0
```

The CP-Majority relation can generate cycles, which is the reason that it is not offered as a social ranking solution. Instead we will introduce the Copeland-like method and Kramer-Simpson-like method in chapters 3.3.2 and 3.3.3 that make use of the CP-Majority functions to determine a power relation between elements. For further readings on CP-Majority, see [7] and [10].

3.3 Social Ranking Solutions

3.3.1 Ordinal Banzhaf

The Ordinal Banzhaf Score is a vector defined by the principle of marginal contributions. Intuitively speaking, if a player joining a coalition causes it to move up in the ranking, it can be interpreted as a positive contribution. On the contrary a negative contribution means that participating causes the coalition to go down in the ranking.

Definition 5. (Ordinal marginal contribution [5]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. For a given element $i \in N$, its *ordinal marginal contribution* $m_i^S(\succeq)$ with right to a coalition $S \in \mathcal{P}$ is defined as:

$$m_i^S(\succeq) = \begin{cases} 1 & \text{if } S \cup \{i\} \succ S \\ -1 & \text{if } S \succ S \cup \{i\} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Definition 6. (Ordinal Banzhaf relation) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Ordinal Banzhaf relation* is the binary relation $R_{\text{Banz}}^{\succeq} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR_{\text{Banz}}^{\succeq}j \Leftrightarrow \text{Score}_{\text{Banz}}(i) \geq \text{Score}_{\text{Banz}}(j), \tag{4}$$

where $\text{Score}_{\text{Banz}}(i) = \sum_S m_i^S(\succeq)$ for all $S \in N \setminus \{i\}$.

Note that if $S \notin \mathcal{P}$ or $S \cup \{i\} \notin \mathcal{P}$, $m_i^S(\succeq) = 0$.

The function `ordinalBanzhafScores()` returns three numbers for each element,

1. the number of coalitions S where a player's contribution has a positive impact,
2. the number of coalitions S where a player's contribution has a negative impact, and
3. the number of coalitions S for which no information can be gathered, because $S \notin \mathcal{P}$ or $S \cup \{i\} \notin \mathcal{P}$.

The sum of the first two numbers determines the score of a player. Players with higher scores rank higher.

```
pr <- as.PowerRelation(list(c(1,2), c(1), c(2)))
pr
## 12 > 1 > 2
```

```
# both players 1 and 2 have an Ordinal Banzhaf Score of 1
# therefore they are indifferent to one another
# note that the empty set is missing, as such we cannot compare {}u{i} with {}
ordinalBanzhafScores(pr)
## $`1`
## [1] 1 0 1
```

```
##
## $`2`
## [1] 1 0 1
##
## attr(,"class")
## [1] "OrdinalBanzhafScores"
```

```
ordinalBanzhafRanking(pr)
```
```
## 1 ~ 2
```

```
pr <- as.PowerRelation("ab > a > {} > b")

# player b has a negative impact on the empty set
# -> player b's score is 1 - 1 = 0
# -> player a's score is 2 - 0 = 2
sapply(ordinalBanzhafScores(pr), function(score) sum(score[c(1,2)]))
```
```
## a b
## 2 0
```

```
ordinalBanzhafRanking(pr)
```
```
## a > b
```

3.3.2 Copeland-like method

The Copeland-like method of ranking elements based on the CP-Majority rule is strongly inspired by the Copeland score from social choice theory[14]. The score of an element $i \in N$ is determined by the amount of the pairwise CP-Majority winning comparisons $iR^{\succeq}_{\text{CP}}j$, minus the number of all losing comparisons $jR^{\succeq}_{\text{CP}}i$ against all other elements $j \in N \setminus \{i\}$.

Definition 7. (Copeland-like relation [10]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Copeland-like* relation is the binary relation $R^{\succeq}_{\text{Cop}} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR^{\succeq}_{\text{Cop}}j \Leftrightarrow \text{Score}_{\text{Cop}}(i) \geq \text{Score}_{\text{Cop}}(j), \tag{5}$$

where $\text{Score}_{\text{Cop}}(i) = |\{j \in N \setminus \{i\} : d_{ij}(\succeq) \geq d_{ji}(\succeq)\}| - |\{j \in N \setminus \{i\} : d_{ij}(\succeq) \leq d_{ji}(\succeq)\}|$

`copelandScores(pr)` returns two numerical values for each element, a positive number for the winning comparisons (shown in $\text{Score}_{\text{Cop}}(i)$ on the left) and a negative number for the losing comparisons (in $\text{Score}_{\text{Cop}}(i)$ on the right).

```
pr <- as.PowerRelation("(abc ~ ab ~ c ~ a) > (b ~ bc) > ac")
scores <- copelandScores(pr)

# Based on CP-Majority, a>=b and a>=c (+2), but b>=a (-1)
scores$a
```
```
## [1]  2 -1
```

```
sapply(copelandScores(pr), sum)
```
```
##  a  b  c
##  1  0 -1
```

```
copelandRanking(pr)
```
```
## a > b > c
```

3.3.3 Kramer-Simpson-like method

Strongly inspired by the Kramer-Simpson method of social choice theory[15, 16], elements are ranked inversely to their greatest pairwise defeat over all possible CP-Majority comparisons.

Definition 8. (Kramer-Simpson-like relation [10]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Kramer-Simpson-like* relation is the binary relation $R_{\text{KS}}^{\succeq} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR_{\text{KS}}^{\succeq}j \Leftrightarrow \text{Score}_{\text{KS}}(i) \leq \text{Score}_{\text{KS}}(j), \tag{6}$$

where $\text{Score}_{\text{KS}}(i) = \max_j d_{ji}(\succeq)$ for all $j \in N \setminus \{i\}$.

`kramerSimpsonScores(pr)` returns a single numerical value for each element, which sorted lowest to highest gives us the ranking solution.

```
pr <- as.PowerRelation("(abc ~ ab ~ c ~ a) > (b ~ bc) > ac")
unlist(kramerSimpsonScores(pr))
```
```
## a b c
## 0 0 1
```

```
kramerSimpsonRanking(pr)
```
```
## a ~ b > c
```

There is a small caveat to Definition 8. By default this function does not compare $d_{ii}(\succeq)$. In other terms, the score of every element is the maximum CP-Majority comparison score against all *other* elements.

This is slightly different from the definition found in [10], where the CP-Majority comparison $d_{ii}(\succeq)$ is also considered. Since by definition $d_{ii}(\succeq) = 0$, the Kramer-Simpson scores in those cases will never be negative, possibly discarding valuable information.

To still account for the original definition in [10], the functions `kramerSimpsonScores` and `kramerSimpsonRanking` offer a `compIvsI` parameter that can be set to `TRUE` if one wishes for $d_{ii}(\succeq)$ to be included in the comparisons.

```
pr <- as.PowerRelation("b > (a ~ c) > ab > (ac ~ bc) > {} > abc")
kramerSimpsonRanking(pr)
```
```
## b > a > c
```

```
# notice how b's score is negative
unlist(kramerSimpsonScores(pr))
```
```
##  a  b  c
##  1 -1  2
```

```
kramerSimpsonScores(pr, elements = "b", compIvsI = TRUE)
```
```
## $b
## [1] 0
```

```
##
## attr(,"class")
## [1] "KramerSimpsonScores"
```

3.3.4 Lexicographical Excellence Solution

The idea behind the lexicographical excellence solution (Lexcel) is to reward elements appearing more frequently in higher ranked equivalence classes.

For a given power relation $\succeq$ and its quotient order $\sum_1 \succ \cdots \succ \sum_m$, we denote by i_k the number of coalitions in $\sum_k$ containing i:

$$i_k = |\{S \in \textstyle\sum_k : i \in S\}| \tag{7}$$

for $k \in \{1, \ldots, m\}$. Now, let $\text{Score}_{\text{Lex}}(i)$ be the m-dimensional vector $\text{Score}_{\text{Lex}}(i) = (i_1, \ldots, i_m)$ associated to $\succeq$. Consider the lexicographic order $\geq_{\text{Lex}}$ among vectors $\mathbf{i}$ and $\mathbf{j}$: $\mathbf{i} \geq_{\text{Lex}} \mathbf{j}$ if either $\mathbf{i} = \mathbf{j}$ or there exists $t : i_r = j_r, r \in \{1, \ldots, t-1\}$, and $i_t > j_t$.

Definition 9. (Lexicographic-Excellence relation [8]) Let $\succeq \in \mathcal{T}(\mathcal{P})$ with its corresponding quotient order $\sum_1 \succ \cdots \succ \sum_m$. The *Lexicographic-Excellence* relation is the binary relation $R^{\succeq}_{\text{Lex}} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR^{\succeq}_{\text{Lex}}j \Leftrightarrow \text{Score}_{\text{Lex}}(i) \geq_{\text{Lex}} \text{Score}_{\text{Lex}}(j) \tag{8}$$

```
pr <- as.PowerRelation("12 > (123 ~ 23 ~ 3) > (1 ~ 2) > 13")

# show the number of times an element appears in each equivalence class
# e.g. 3 appears 3 times in [[2]] and 1 time in [[4]]
lapply(pr$equivalenceClasses, unlist)
## list()
```

```
lexScores <- lexcelScores(pr)
for(i in names(lexScores))
  paste0("Lexcel score of element ", i, ": ", lexScores[i])

# at index 1, element 2 ranks higher than 3
lexScores['2'] > lexScores['3']
## [1] TRUE
```

```
# at index 2, element 2 ranks higher than 1
lexScores['2'] > lexScores['1']
## [1] TRUE
```

```
lexcelRanking(pr)
## 2 > 1 > 3
```

For some generalizations of the Lexcel solution see also [9].

Lexcel score vectors are very similar to the cumulative score vectors (3.2.2) in that the number of times an element appears in a given equivalence class is of interest. In fact,

applying the base function `cumsum` on an element's lexcel score gives us its cumulative score.

```
lexcelCumulated <- lapply(lexScores, cumsum)
cumulScores <- cumulativeScores(pr)

paste0(names(lexcelCumulated), ": ", lexcelCumulated, collapse = ', ')
## [1] "1: 1:4, 2: c(1, 3, 4, 4), 3: c(0, 3, 3, 4)"

paste0(names(cumulScores), ": ", cumulScores, collapse = ', ')
## [1] "1: 1:4, 2: c(1, 3, 4, 4), 3: c(0, 3, 3, 4)"
```

3.3.5 Dual Lexicographical Excellence Solution

Similar to the Lexcel ranking, the Dual Lexcel also uses the Lexcel score vectors from definition 9 to establish a ranking. However, instead of rewarding higher frequencies in high ranking coalitions, it punishes higher frequencies in lower ranking coalitions, or, it punishes mediocrity[13].

Take the values i_k for $k \in \{1, \ldots, m\}$ and the Lexcel score vector $\text{Score}_{\text{Lex}}(i)$ from section 3.3.4. Consider the dual lexicographical order $\geq_{\text{DualLex}}$ among vectors $\mathbf{i}$ and $\mathbf{j}$: $\mathbf{i} \geq_{\text{DualLex}} \mathbf{j}$ if either $\mathbf{i} = \mathbf{j}$ or there exists $t : i_t < j_t$ and $i_r = j_r, r \in \{t+1, \ldots, m\}$.

Definition 10. (Dual Lexicographical-Excellence relation [13]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Dual Lexicographic-Excellence* relation is the binary relation $R^{\succeq}_{\text{DualLex}} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR^{\succeq}_{\text{DualLex}}j \Leftrightarrow \text{Score}_{\text{Lex}}(i) \geq_{\text{DualLex}} \text{Score}_{\text{Lex}}(j) \tag{9}$$

The S3 class `LexcelScores` does not account for Dual Lexcel comparisons. Instead `-rev(x)` is called on a Lexcel score vector `x` such that the resulting comparisons produces a Dual Lexcel ranking solution.

```
pr <- as.PowerRelation("12 > (123 ~ 23 ~ 3) > (1 ~ 2) > 13")

lexScores <- lexcelScores(pr)

# in regular Lexcel, 1 scores higher than 3
lexScores['1'] > lexScores['3']
## [1] TRUE

# turn Lexcel score into Dual Lexcel score
dualLexScores <- structure(
  lapply(lexcelScores(pr), function(r) -rev(r)),
  class = 'LexcelScores'
)

# now 1 scores lower than 3
dualLexScores['1'] > dualLexScores['3']
## [1] FALSE
```

```
# element 2 comes out at the top in both Lexcel and Dual Lexcel
lexcelRanking(pr)
## 2 > 1 > 3
```

```
dualLexcelRanking(pr)
## 2 > 3 > 1
```

4 Relations

4.1 Incidence Matrix

In our vignette we focused more on the intuitive aspects of power relations and social ranking solutions. To reiterate, a power relation is a total preorder, or a reflexive and transitive relation $\succeq \in \mathcal{P} \times \mathcal{P}$, where $\sim$ denotes the symmetric part and $\succ$ its asymmetric part.

A power relation can be viewed as an incidence matrix $(b_{ij}) = B \in \{0,1\}^{|\mathcal{P}|\times|\mathcal{P}|}$. Given two coalitions $i, j \in \mathcal{P}$, if iRj then $b_{ij} = 1$, else 0.

With help of the `relations` package, the functions `relations::as.relation(pr)` and `powerRelationMatrix(pr)` turn a `PowerRelation` object into a `relation` object. `relations` then offers ways to display the `relation` object as an incidence matrix with `relation_incidence(rel)` and to test basic properties such `relation_is_linear_order(rel)`, `relation_is_acyclic(rel)` and `relation_is_antisymmetric(rel)` (see `relations` package for more [11]).

```
pr <- as.PowerRelation("ab > a > {} > b")
rel <- relations::as.relation(pr)

relations::relation_incidence(rel)
## Incidences:
##    ab a {} b
## ab  1 1  1 1
## a   0 1  1 1
## {}  0 0  1 1
## b   0 0  0 1
```

```
c(
  relations::relation_is_acyclic(rel),
  relations::relation_is_antisymmetric(rel),
  relations::relation_is_linear_order(rel),
  relations::relation_is_complete(rel),
  relations::relation_is_reflexive(rel),
  relations::relation_is_transitive(rel)
)
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

Note that the columns and rows are sorted by their names in `relation_domain(rel)`, hence why each name is preceded by the ordering number.

```
# a power relation where coalitions {1} and {2} are indifferent
pr <- as.PowerRelation("12 > (1 ~ 2)")
```

```r
rel <- relations::as.relation(pr)

# we have both binary relations {1}R{2} as well as {2}R{1}
relations::relation_incidence(rel)
```

```
## Incidences:
##    12 1 2
## 12  1 1 1
## 1   0 1 1
## 2   0 1 1
```

```r
# FALSE
c(
  relations::relation_is_acyclic(rel),
  relations::relation_is_antisymmetric(rel),
  relations::relation_is_linear_order(rel),
  relations::relation_is_complete(rel),
  relations::relation_is_reflexive(rel),
  relations::relation_is_transitive(rel)
)
```

```
## [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

4.2 Cycles and Transitive Closure

A cycle in a power relation exists, if there is one coalition $S \in 2^N$ that appears twice. For example, in $\{1,2\} \succ (\{1\} \sim \emptyset) \succ \{1,2\}$, the coalition $\{1,2\}$ appears at the beginning and at the end of the power relation.

Properly handling power relations and calculating social ranking solutions with cycles is somewhat ill-defined, hence a warning message is shown as soon as one is created.

```r
as.PowerRelation("12 > 2 > (1 ~ 2) > 12")
```

```
#! Warning in createLookupTables(equivalenceClasses): Found 2
duplicate coalitions, listed below. This violates transitivity and
can cause issues with certain ranking solutions. You may want to take
a look at socialranking::transitiveClosure().
#!  - {2}
#!  - {1, 2}
```

```
## 12 > 2 > (1 ~ 2) > 12
```

Recall that a power relation is transitive, meaning for three coalitions $x, y, z \in 2^N$, if xRy and yRz, then xRz. If we introduce cycles, we pretty much introduce symmetry. Assume we have the power relation $x \succ y \succ x$. Then, even though xRy and yRx are defined as the asymmetric part of the power relation $\succeq$, together they form the symmetric power relation $x \sim y$.

`transitiveClosure(pr)` is a function that turns a power relation with cycles into one without one. In the process of removing duplicate coalitions, it turns all asymmectric relations within a cycle into symmetric relations.

```r
pr <- suppressWarnings(as.PowerRelation(list(1, 2, 1)))
pr
```

```
## 1 > 2 > 1
```

```r
transitiveClosure(pr)
```
```
## (1 ~ 2)
```

```r
# two cycles, (1>3>1) and (2>23>2)
pr <- suppressWarnings(
  as.PowerRelation("1 > 3 > 1 > 2 > 23 > 2")
)

transitiveClosure(pr)
```
```
## (1 ~ 3) > (2 ~ 23)
```

```r
# overlapping cycles
pr <- suppressWarnings(
  as.PowerRelation("c > ac > b > ac > (a ~ b) > abc")
)

transitiveClosure(pr)
```
```
## c > (ac ~ b ~ a) > abc
```

Bibliography

1. Shapley LS, Shubik M. A method for evaluating the distribution of power in a committee system. American political science review. 1954;48:787–92.
2. Banzhaf III JF. Weighted voting doesn't work: A mathematical analysis. Rutgers L Rev. 1964;19:317.
3. Holler MJ, Nurmi H. Power, voting, and voting power: 30 years after. Springer; 2013.
4. Moretti S, Öztürk M. Some axiomatic and algorithmic perspectives on the social ranking problem. In: International conference on algorithmic DecisionTheory. Springer; 2017. p. 166–81.
5. Khani H, Moretti S, Öztürk M. An ordinal banzhaf index for social ranking. In: 28th international joint conference on artificial intelligence (IJCAI 2019). 2019. p. 378–84.
6. Haret A, Khani H, Moretti S, Öztürk M. Ceteris paribus majority for social ranking. In: 27th international joint conference on artificial intelligence (IJCAI-ECAI-18). 2018. p. 303–9.
7. Fayard N, Öztürk-Escoffier M. Ordinal social ranking: Simulation for CP-majority rule. In: DA2PL'2018 (from multiple criteria decision aid to preference learning). 2018.
8. Bernardi G, Lucchetti R, Moretti S. Ranking objects from a preference relation over their subsets. Social Choice and Welfare. 2019;52:589–606.
9. Algaba E, Moretti S, Rémila E, Solal P. Lexicographic solutions for coalitional rankings. Social Choice and Welfare. 2021;1–33.
10. Allouche T, Escoffier B, Moretti S, Öztürk M. Social ranking manipulability for the CP-majority, banzhaf and lexicographic excellence solutions. In: Bessiere C, editor. Proceedings of the twenty-ninth international joint conference on artificial intelligence, IJCAI-20. International Joint Conferences on Artificial Intelligence Organization; 2020. p. 17–23. doi:10.24963/ijcai.2020/3.
11. Meyer D, Hornik K. Relations: Data structures and algorithms for relations. 2022. https://CRAN.R-project.org/package=relations.
12. Moretti S. An axiomatic approach to social ranking under coalitional power relations. Homo Oeconomicus. 2015;32:183–208.
13. Serramia M, López-Sánchez M, Moretti S, Rodríguez-Aguilar JA. On the dominant set selection problem and its application to value alignment. Autonomous Agents and

Multi-Agent Systems. 2021;35:1–38.
14. Copeland AH. A reasonable social welfare function. mimeo, 1951. University of Michigan; 1951.
15. Simpson PB. On defining areas of voter choice: Professor tullock on stable voting. The Quarterly Journal of Economics. 1969;83:478–90.
16. Kramer GH. A dynamical model of political equilibrium. Journal of Economic Theory. 1975;16:310–34.