cna: An R Package for Configurational Causal Inference and Modeling

Michael Baumgartner
University of Bergen, Norway

Mathias Ambühl
Consult AG, Switzerland

Abstract

The R package **cna** provides comprehensive functionalities for causal inference and modeling with *Coincidence Analysis* (CNA), which is a configurational comparative method of causal data analysis. In this vignette, we first review the theoretical and methodological foundation of CNA. Second, we introduce the data types processable by CNA, the package's core analytical functions with their arguments, and some auxiliary functions for data simulations. Third, CNA's output along with relevant fit parameters and output attributes are discussed. Fourth, we provide guidance on how to interpret that output and, in particular, on how to proceed in case of model ambiguities. Finally, some considerations are offered on benchmarking the reliability of CNA.

Keywords: configurational comparative methods, set-theoretic methods, Coincidence Analysis, Qualitative Comparative Analysis, INUS causation, Boolean causation.

1. Introduction

Since the mid-1980ies, different variants of *configurational comparative methods* (CCMs) have gradually been added to the toolkit for causal data analysis in the social sciences. The most prominent CCM is *Qualitative Comparative Analysis* (QCA) with its main variants *crisp-set* QCA (csQCA) Ragin (1987), *multi-value* QCA (mvQCA) Cronqvist and Berg-Schlosser (2009), and *fuzzy-set* QCA (fsQCA) Ragin (2008, 2009) (cf. also Thiem 2014 for an attempt to unify these variants). Since its first introduction, QCA has gained considerable popularity and has been applied in areas as diverse as social and political science, international relations, business administration, management, environmental science, evaluation science, and public health (for an overview over corresponding publications and detailed references see the bibliography section on the `http://compasss.org` website).

Coincidence Analysis (CNA) was added to the family of CCMs in Baumgartner (2009a; 2009b) and substantively extended and reworked in Baumgartner and Ambühl (2018). There are two key areas of difference between CNA and original variants of QCA, the algorithmic machinery and the search target:

1. QCA (in its original form) builds causal models using Quine-McCluskey optimization (QMC, Quine 1959; McCluskey 1965), which is an algorithm not designed for causal inference and which, accordingly, gives rise to various problems when nonetheless implemented for that purpose. For instance, data fragmentation forces QMC to draw on counterfactual reasoning that goes beyond the data and sometimes requires assumptions

contradicting the very causal structures under investigation (cf. Baumgartner 2015); or QMC has built-in protocols for ambiguity reduction that are inadequate in the context of causal discovery (cf. Baumgartner and Thiem 2017a). CNA, by contrast, is relying on an algorithmic machinery that is tailor-made for causal inference and, consequently, steers clear of the problems induced by QMC.

2. QCA searches for causal structures with single outcomes only; that is, it standardly treats exactly one variable in processed data as endogenous. CNA, by contrast, can treat any number of variables as endogenous and, hence, builds multi-outcome structures as causal chains or common-cause structures.

In recent developments of QCA, these differences have been diminished. For instance, the problems due to QMC have been alleviated by the development of an algorithm called *enhanced QMC* (*e*QMC, Duşa and Thiem 2015). The currently most dependable QCA program, the **QCApro** package for R (Thiem 2018), implements *e*QMC in a way that avoids both recourse to counterfactual reasoning and inadequate ambiguity reduction.[1] Moreover, apart from **QCApro**, also the **QCA** R package now provides functionalities for investigating the causes of multiple outcomes (cf. also Thiem 2015). Still, no available QCA program follows CNA in building causal models representing multi-outcome structures, in testing for structural redundancies (section 5.5 below) that may arise when single-outcome models are combined to multi-outcome ones, and in measuring the latter's model fit.

Most importantly, substantial algorithmic differences remain between CNA and these latest advancements of QCA. While QCA builds causal models from the *top down* by first identifying maximal dependency structures and then gradually reducing them to minimal, that is, redundancy-free ones, CNA (since Baumgartner and Ambühl 2018) uses a *bottom-up approach* that progressively combines atomic structural components to complex but redundancy-free structures. When applied to noise-free data, the two approaches yield the same results, but when applied to noisy data, they tend to come apart. While the top-down approach runs a risk of failing to eliminate all redundant elements from causal models and of abandoning an analysis prematurely, the bottom-up approach is not affected by these problems.

In the broader methodological landscape, CCMs differ from other techniques as regression analytical methods (RAMs) (e.g. Gelman and Hill 2007) or Bayes-nets methods (BNMs) (e.g. Spirtes *et al.* 2000) in a number of respects. For instance, while RAMs and BNMs search for causal dependencies among *variables*, CCMs search for causal dependencies among concrete *values of variables*. More specifically, RAMs scrutinize covariation hypotheses as "the more/less of X, the more/less of Y" and BNMs analyze hypotheses about conditional (in)dependencies as "X and Y can/cannot be rendered conditionally independent". CCMs, by contrast, study *implication hypotheses* as "$X{=}\chi_i$ is (non-redundantly) sufficient/necessary for $Y{=}\gamma_i$", where χ_i and γ_i are concrete values of X and Y. These types of hypotheses are logically independent: certain values of X and Y may be implicationally dependent, while X and Y themselves are covariationally or conditionally independent, or X and Y may be covariationally and conditionally dependent, while none of their concrete values are implicationally dependent (cf. Thiem and Baumgartner 2016; Thiem *et al.* 2016). Moreover, whereas

[1] The other available QCA R package—**QCA** (Duşa 2007)—also uses *e*QMC (among other algorithms) but, unlike **QCApro**, still has default parameter settings that follow QMC's (causally inadequate) protocol for ambiguity reduction. All other QCA programs continue to rely on standard QMC: **fs/QCA** (Ragin 2014), **fuzzy** (Longest and Vaisey 2008), **Tosmana** (Cronqvist 2011) and **Kirq** (Reichert and Rubinson 2014).

RAMs and BNMs quantify net effects and effect sizes,[2] CCMs place a Boolean ordering on sets of causes by grouping their elements conjunctively, disjunctively, and sequentially. In short, RAMs, BNMs, and CCMs study different properties/aspects of causal structures: RAMs and BNMs study statistical and probabilistic properties as characterized by statistical or probabilistic theories of causation (Simon 1954; Suppes 1970), CCMs scrutinize Boolean properties as described by *regularity theories* of causation (Mackie 1974).

The Boolean properties of causation encompass three complexity dimensions. The first is *conjunctivity*: to bring about an effect, say, liberal democracy in early modern Europe (D=1), different factors need to be instantiated (or *not* instantiated) jointly; for instance, according to Downing's (1992) theory of the origins of liberal democracy, a country must have a history of medieval constitutionalism (C=1) and absent military revolution (in the early modern period) (R=0) (cf. Goertz 2006, 252-254). Only a coincident instantiation of the conjunction C=1 $*$ R=0 produces the effect D. *Disjunctivity* (or equifinality) is a second complexity dimension: an effect can be brought about along alternative causal paths. Downing (1992, 78-79, 240) identifies four paths leading to R=0: a geography that deters invading armies (G=1), commercial wealth (W=1), foreign resource mobilization (M=1), and foreign alliances (A=1). Each condition in the disjunction G=1 + W=1 + M=1 + A=1 can bring about R=0 independently of the other conditions. The third complexity dimension is *sequentiality*: effects tend to cause further effects, propagating causal influence along causal chains. In Downing's theory there are multiple chains, for instance, W=1 is causally relevant to R=0, which, in turn, is causally relevant to D=1, or there is a chain from A=1 via R=0 to D=1. Overall, the theory entails the following Boolean model (cf. Goertz 2006, 254), where "$\rightarrow$" stands for the Boolean operation of implication:

$$(G{=}1 \ + \ W{=}1 \ + \ M{=}1 \ + \ A{=}1 \ \rightarrow \ R{=}0) * (C{=}1 * R{=}0 \ \rightarrow \ D{=}1) \tag{1}$$

The **cna** package is currently the only available software for configurational causal data analysis that builds complex models as (1). This vignette provides a detailed introduction to **cna**. We first exhibit **cna**'s theoretical and methodological background. Second, we discuss the main inputs of the package's core function `cna()` along with numerous auxiliary functions for data review and simulation. Third, the working of the algorithm implemented in `cna()` is presented. Fourth, we explain `cna()`'s output along with relevant fit parameters and output attributes. Fifth, we provide some guidance on how to interpret that output and, in particular, on how to proceed in case of model ambiguities. Finally, some considerations are offered on benchmarking the reliability of `cna()`.

2. CNA's regularity theoretic background

Modern regularity theories of causation define causation in terms of Boolean difference-making within a fixed causal context. To this end, they rely on the metaphysical background assumption that causation ultimately is a deterministic dependence relation, meaning that the indeterminism often encountered in ordinary data is due to our epistemic limitations and our resulting inability to sufficiently control for confounding and noise. Against that metaphysical background (not further discussed here), X=χ_i is more explicitly defined to be a regularity theoretic cause of Y=γ_i if there exists a context $\mathcal{F}$ in which no alternative causes (i.e. no

[2] For effect size estimation using BNMs see, for example, the R package **pcalg** (Kalisch *et al.* 2012).

causes not containing $X{=}\chi_i$) of $Y{=}\gamma_i$ are operative such that, in $\mathcal{F}$, a change from $X{=}\chi_i$ to $X{=}\chi_k$, where $\chi_i \neq \chi_k$, is systematically associated with a change from $Y{=}\gamma_i$ to $Y{=}\gamma_k$, where $\gamma_i \neq \gamma_k$. If $X{=}\chi_i$ does not make a difference to $Y{=}\gamma_i$ in any context $\mathcal{F}$, $X{=}\chi_i$ is redundant to account for $Y{=}\gamma_i$ and, thus, no cause of $Y{=}\gamma_i$. The most influential theory defining causation along these lines is Mackie's (1974) *INUS-theory*. Refinements of it have been proposed by Graßhoff and May (2001), Baumgartner (2008; 2013), and Baumgartner and Falk (2018).

To further clarify CNA's regularity theoretic background, a number of preliminaries are required.

2.1. Factors and their values

As reflected in the types of hypotheses scrutinized by CNA, regularity theoretic causation is a relation that holds between *variables/factors taking on specific values*. (We will use the terms "variable" and "factor" interchangeably.) Factors represent categorical properties that partition sets of units of observation (cases) either into two sets, in case of binary properties, or into more than two (but finitely many) sets, in case of multi-value properties. Factors representing binary properties can be *crisp-set* (*cs*) or *fuzzy-set* (*fs*); the former can take on 0 and 1 as possible values, whereas the latter can take on any (continuous) values from the unit interval $[0, 1]$. Factors representing multi-value properties are called *multi-value (mv) factors*; they can take on any of an open (but finite) number of possible values $\{0, 1, 2, \ldots, n\}$.

Values of a *cs* or *fs* factor X can be interpreted as membership scores in the set of cases exhibiting the property represented by X. A case of type $X{=}1$ is a full member of that set, a case of type $X{=}0$ is a (full) non-member, and a case of type $X{=}\chi_i$, $0 < \chi_i < 1$, is a member to degree χ_i. An alternative interpretation, which lends itself particularly well for causal modeling, is that "$X{=}1$" stands for the full presence of the property represented by X, "$X{=}0$" for its full absence, and "$X{=}\chi_i$" for its partial presence (to degree χ_i). By contrast, the values of an *mv* factor X designate the particular way in which the property represented by X is exemplified. For instance, if X represents the education of subjects, $X{=}2$ may stand for "high school", with $X{=}1$ ("no completed primary schooling") and $X{=}3$ ("university") designating other possible property exemplifications. *Mv* factors taking on one of their possible values also define sets, but the values themselves must not be interpreted as membership scores; rather they denote the relevant property exemplification.

As the explicit "Variable=value" notation yields convoluted syntactic expressions with increasing model complexity, the **cna** package uses the following shorthand notation, which is conventional in Boolean algebra: membership in a set is expressed by italicized upper case and non-membership by lower case Roman letters. Hence, in case of *cs* and *fs* factors, we write "X" for $X{=}1$ and "x" for $X{=}0$. It must be emphasized that, while this notation significantly simplifies the syntax of Boolean models, it introduces a risk of misinterpretation, for it yields that the factor X and its taking on the value 1 are both expressed by "X". Disambiguation must hence be facilitated by the concrete context in which "X" appears. Therefore, whenever we do not explicitly characterize italicized Roman letters as "factors", we use them in terms of the shorthand notation. In case of *mv* factors, value assignments to variables are not abbreviated but always written out, using the "Variable=value" notation.

2.2. Boolean operations

Regularity theories spell out causation in terms of the Boolean operations of negation ($\neg X$, or

Inputs		Outputs				
X	Y	$\neg X$	$X{*}Y$	$X+Y$	$X \rightarrow Y$	$X \leftrightarrow Y$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

Table 1: Classical Boolean operations applied to *cs* factors.

x), conjunction ($X{*}Y$), disjunction ($X{+}Y$), implication ($X \rightarrow Y$), and equivalence ($X \leftrightarrow Y$). Negation is a unary truth function, the other operations are binary truth functions. That is, they take one resp. two truth values as inputs and output a truth value. When applied to *cs* factors, both their input and output set is $\{0, 1\}$. Negation is typically translated by "not", conjunction by "and", disjunction by "or", implication by "if ... then", and equivalence by "if and only if (iff)". Their classical definitions are given in Table 1.

These operations can be straightforwardly applied to *mv* factors as well, in which case they amount to functions from the *mv* factors' domain of values into the set $\{0, 1\}$. To illustrate, assume that both X and Y are ternary factors with values from the domain $\{0, 1, 2\}$. The negation of $X{=}2$, *viz.* $\neg(X{=}2)$, then returns 1 iff X is not 2, meaning iff X is 0 or 1. $X{=}2{*}Y{=}0$ yields 1 iff X is 2 and Y is 0. $X{=}2 + Y{=}0$ returns 1 iff X is 2 or Y is 0. $X{=}2 \rightarrow Y{=}0$ yields 1 iff either X is not 2 or Y is 0. $X{=}2 \leftrightarrow Y{=}0$ issues 1 iff either X is 2 and Y is 0 or X is not 2 and Y is not 0.

For fs factors whose values are interpreted as membership scores in fuzzy sets, the classical Boolean operations must be translated into fuzzy logic. There exist numerous systems of fuzzy logic (for an overview cf. Hájek 1998), each of which comes with its own rendering of Boolean operations. In the context of CCMs, the following fuzzy-logic renderings have become standard: negation $\neg X$ is translated in terms of $1 - X$, conjunction $X{*}Y$ in terms of the minimum membership score in X and Y, i.e., $\min(X, Y)$, disjunction $X + Y$ in terms of the maximum membership score in X and Y, i.e., $\max(X, Y)$, an implication $X \rightarrow Y$ is taken to express that the membership score in X is smaller or equal to Y ($X \leq Y$), and an equivalence $X \leftrightarrow Y$ that the membership scores in X and Y are equal ($X = Y$).

Based on the implication operator, the notions of *sufficiency* and *necessity* are defined, which are the two Boolean dependencies exploited by regularity theories:

Sufficiency X is sufficient for Y iff $X \rightarrow Y$ (or equivalently: $x + Y$; and colloquially: "if X is given, then Y is given");

Necessity X is necessary for Y iff $Y \rightarrow X$ (or equivalently: $\neg X \rightarrow \neg Y$ or $y + X$; and colloquially: "if Y is given, then X is given").

Analogously for more complex expressions:

- $X{=}3{*}Z{=}2$ is sufficient for $Y{=}4$ iff $X{=}3{*}Z{=}2 \rightarrow Y{=}4$;
- $X{=}3 + Z{=}2$ is necessary for $Y{=}4$ iff $Y{=}4 \rightarrow X{=}3 + Z{=}2$;
- $X{=}3 + Z{=}2$ is sufficient and necessary for $Y{=}4$ iff $X{=}3 + Z{=}2 \leftrightarrow Y{=}4$.

2.3. Boolean causal models

Boolean dependencies of sufficiency and necessity amount to mere patterns of co-occurrence of factor values; as such, they carry no causal connotations whatsoever. In fact, most Boolean dependencies do not reflect causal dependencies. To mention just two well-rehearsed examples: the sinking of a (properly functioning) barometer is sufficient for bad weather but it does not cause the weather; or whenever the street is not wet, it does not rain, hence, wetness of the street is necessary for rainfall but certainly not causally relevant for it. At the same time, some dependencies of sufficiency and necessity are in fact due to underlying causal dependencies: rainfall is sufficient for wet streets and also a cause thereof, or the presence of oxygen is necessary for fires and also a cause thereof.

That means the crucial problem to be solved by a regularity theory is to filter out those Boolean dependencies that are due to underlying causal dependencies and are, hence, amenable to a causal interpretation. The main reason why most structures of Boolean dependencies do not reflect causation is that they tend to contain different types of redundancies—redundancies in sufficiency and necessity relations but also structural redundancies (section 5.5)—, whereas structures of causal dependencies do not feature redundant elements. Every part of a causal structure makes a difference to the behavior of the factors in that structure in at least one context $\mathcal{F}$. Accordingly, to filter out the causally interpretable Boolean dependencies, regularity theories rely on a non-redundancy principle:

Non-redundancy (NR) A Boolean dependency structure is causally interpretable only if it does not contain any redundant elements.

Applied to sufficient and necessary conditions, (NR) entails that whatever can be removed from such conditions without affecting their sufficiency and necessity is not a difference-maker and, hence, not a cause (Baumgartner 2015). Causes are elements of sufficient and necessary conditions for which at least one causal context $\mathcal{F}$ exists in which they are indispensable to account for a scrutinized outcome, because no alternative causes are operative in $\mathcal{F}$. Or in Mackie's (1974, 62) words, causes are at least *INUS conditions*, *viz.* insufficient but non-redundant parts of unnecessary but sufficient conditions.

Modern regularity theories formally cash this idea out on the basis of the notion of a *minimal theory*, which essentially amounts to an expression of a Boolean dependency structure that is rigorously freed of all redundancies. To do justice to the different types of redundancies that Boolean dependency structures may be affected by, the complete definition of the notion of a minimal theory is intricate and beyond the scope of this vignette (for the latest definition cf. Baumgartner and Falk 2018). For our subsequent purposes, the following simplified definition will suffice: an *atomic minimal theory* of an outcome Y is a minimally necessary disjunction (in disjunctive normal form) of minimally sufficient conditions of Y.

Minimal sufficiency A conjunction Φ of coincidently instantiated factor values (e.g., $X_1{*}X_2{*}\ldots{*}X_n$) is a minimally sufficient condition of Y iff $\Phi \rightarrow Y$ and there does not exist a proper part Φ' of Φ such that $\Phi' \rightarrow Y$, where a proper part Φ' of Φ is the result of eliminating one or more conjuncts from Φ.

Minimal necessity A disjunction Ψ of minimally sufficient conditions (e.g., $\Phi_1 + \Phi_2 + \ldots + \Phi_n$) is a minimally necessary condition of Y iff $Y \rightarrow \Psi$ and there does not exist a

proper part Ψ' of Ψ such that $Y \rightarrow \Psi'$, where a proper part Ψ' of Ψ is the result of eliminating one or more disjuncts from Ψ.

An atomic minimal theory of Y states an equivalence of the form $\Psi \leftrightarrow Y$. Atomic minimal theories represent single-outcome structures. Conjunctions of atomic minimal theories that are themselves redundancy-free represent multi-outcome structures and are called *complex minimal theories*.

Minimal theories connect Boolean dependencies, which—by themselves—are purely functional and non-causal, to causal dependencies: only those Boolean dependencies are causally interpretable that appear in minimal theories. That does not mean that every minimal theory inferred from a data set δ is guaranteed to express the δ-generating causal structure. As we shall see below, it frequently happens that multiple minimal theories can be inferred from δ, in which case only one of these theories may truthfully reflect the δ-generating structure. Or, as shown in Baumgartner (2013, 93-95), if the analyzed set of factors is underspecified, minimal theories may be unfaithful to the δ-generating structure. It does mean, though, that minimal theories inferred from a data set δ express *the empirical evidence* on causal dependencies contained in δ. In other words, the data from which a minimal theory $\Psi \leftrightarrow Y$ has been inferred contain evidence—although possibly indeterminate or fallacious evidence—for the causal relevance of all factors in Ψ.

To further clarify the causal interpretation of minimal theories, consider the following complex example:

$$(A{*}b \ + \ a{*}B \ \leftrightarrow \ C) \ * \ (C{*}f \ + \ D \ \leftrightarrow \ E) \tag{2}$$

Functionally put, (2) claims that, in the analyzed data δ, the presence of A in conjunction with the absence of B (i.e., b) as well as a in conjunction with B are two alternative minimally sufficient conditions of C, and that $C{*}f$ and D are two alternative minimally sufficient conditions of E. Moreover, both $A{*}b + a{*}B$ and $C{*}f + D$ are claimed to be minimally necessary for C and E in δ, respectively. Against the background of a regularity theory, these functional relations entail the following causal claims:

1. the factor values listed on the left-hand sides of "$\leftrightarrow$" are causally relevant for the factor values on the right-hand sides;
2. A and b are jointly relevant to C and located on a causal path that differs from the path on which the jointly relevant a and B are located; C and f are jointly relevant to E and located on a path that differs from D's path;
3. there is a causal chain from $A{*}b$ and $a{*}B$ via C to E.

More generally put, minimal theories ascribe causal relevance to their constitutive factor values, place them on the same or different paths to the outcomes, and order them sequentially. That is, they render transparent the three Boolean complexity dimensions of causality—which is why they are also referred to as *Boolean causal models*.

Two fundamentals of the interpretation of Boolean causal models must be emphasized. First, ordinary Boolean models make claims about causal relevance *but not about causal irrelevance*. With some additional constraints that are irrelevant for our current purposes (for details cf. Baumgartner and Falk 2018), a regularity theory defines X_1 to be a cause of an outcome Y iff there exists a difference-making context $\mathcal{F}$ for X_1 with respect to Y—meaning that in $\mathcal{F}$ (in

which no alternative causes of Y are operative) $X_1{*}\mathcal{F}$ and $x_1{*}\mathcal{F}$ are systematically associated with different Y-values. While establishing causal relevance merely requires demonstrating the existence of at least one such difference-making context, establishing causal irrelevance would require demonstrating the non-existence of such a context, which is impossible on the basis of the non-exhaustive data samples that are typically analyzed in real-life studies. Correspondingly, the fact that G does not appear in (2) does not imply G to be causally irrelevant to C or E. The non-inclusion of G simply means that the data from which (2) has been derived do not contain evidence for the causal relevance of G. However, future research having access to additional data might reveal the existence of a difference-making context for G and, hence, entail the causal relevance of G to C or E after all.

Second, as anticipated above, Boolean models are to be interpreted relative to the data set δ from which they have been derived. They do not purport to reveal all Boolean properties of the data-generating causal structure. That is, Boolean models typically are *incomplete*. They only detail those causally relevant factor values along with those conjunctive, disjunctive, and sequential groupings for which δ contains evidence. By extension, two different Boolean models $\mathbf{m}_i$ and $\mathbf{m}_j$ derived from two different data sets δ_i and δ_j are in no disagreement if the causal claims entailed by $\mathbf{m}_i$ and $\mathbf{m}_j$ stand in a subset relation.

In the CCM literature, yet another term that signifies essentially the same as *minimal theory* or *Boolean causal model* has become customary: *solution formula*. There is only a slight meaning difference. While the terms *minimal theory* and *Boolean model* refer to any expressions of the form $(\Psi_1 \leftrightarrow Y_1) * \ldots * (\Psi_n \leftrightarrow Y_n)$, *solution formula* refers, more precisely, only to those expressions of this form that are output by a CCM. That is, if $\Psi_1 \leftrightarrow Y$ is issued by CNA, it is also called an *atomic solution formula* (***asf***), whereas $(\Psi_1 \leftrightarrow Y) * (\Psi_2 \leftrightarrow Z)$ as issued by CNA is called a *complex solution formula* (***csf***).

3. The input of CNA

The goal of CNA is thus to output all *asf* and *csf* that fit an input of configurational data (relative to provided thresholds of model fit). The algorithm performing this task in the **cna** package is implemented in the function `cna()`. Its most important arguments are:

```
cna(x, type, ordering = NULL, strict = FALSE, con = 1, cov = 1, con.msc = con,
   notcols = NULL, maxstep = c(3, 3, 9), inus.only = FALSE, suff.only = FALSE,
   what = if (suff.only) "m" else "ac", details = FALSE)
```

This section explains most of these inputs and introduces some auxiliary functions. The arguments `inus.only`, `what`, and `details` will be discussed in section 5.

3.1. Data

Configurational data δ have the form of $m \times k$ matrices, where m is the number of units of observation (cases) and k is the number of factors in δ. Data processed by CNA can either be of type "crisp-set" (cs), "multi-value" (mv) or "fuzzy-set" (fs). Data that feature cs factors only are cs. If the data contain at least one mv factor, they count as mv. Data featuring at least one fs factor are fs.[3] Examples of each data type are given in Table 2.

[3]Mixing mv and fs factors in one analysis is (currently) not supported.

	A	*B*	*C*	*D*
c_1	0	0	0	0
c_2	0	1	0	0
c_3	1	1	0	0
c_4	0	0	1	0
c_5	1	0	0	1
c_6	1	0	1	1
c_7	0	1	1	1
c_8	1	1	1	1

(a) *cs* data

	A	*B*	*C*	*D*
c_1	1	3	3	1
c_2	2	2	1	2
c_3	2	1	2	2
c_4	2	2	2	2
c_5	3	3	3	2
c_6	2	4	3	2
c_7	1	3	3	3
c_8	1	4	3	3

(b) *mv* data

	A	*B*	*C*	*D*	*E*
c_1	0.17	0.02	0.15	0.26	0.09
c_2	0.97	0.23	0.73	0.08	0.10
c_3	0.10	0.72	0.61	0.38	0.08
c_4	0.64	0.73	0.82	0.12	0.66
c_5	0.11	0.30	0.06	0.99	0.78
c_6	0.69	0.23	0.91	0.98	0.84
c_7	0.31	0.80	0.62	0.65	0.74
c_8	0.65	0.87	0.92	0.82	0.85

(c) *fs* data

Table 2: Data types processable by CNA.

Data is given to the `cna()` function via the argument `x`, which is a data frame or an object of class "truthTab" as output by the `truthTab()` function (see section 3.1.1 below). The **cna** package contains a number of exemplary data sets from published CCM studies: `d.autonomy`, `d.educate`, `d.irrigate`, `d.jobsecurity`, `d.minaret`, `d.pacts`, `d.pban`, `d.performance`, `d.volatile`, `d.women`. For details on their contents and sources, see the **cna** reference manual. After having loaded the **cna** package, all of them are directly (i.e. without separate loading) available for processing:

```
R> library(cna)
R> cna(d.educate)
R> cna(d.women)
```

If the data are not of type *cs*, `cna()` must be told explicitly what type of data `x` contains using the `type` argument, which takes the values `"mv"` for *mv* data and `"fs"` for *fs* data. The functions `mvcna(x, ...)` and `fscna(x, ...)` are available as shorthands for `cna(x, type = "mv", ...)` and `cna(x, type = "fs", ...)`, respectively.

```
R> cna(d.jobsecurity, type = "fs")
R> fscna(d.jobsecurity)
R> cna(d.pban, type = "mv")
R> mvcna(d.pban)
```

Truth tables

To facilitate the reviewing of data, the `truthTab()` function assembles cases with identical configurations in a table called a *truth table*.[4]

```
truthTab(x, type = c("cs", "mv", "fs"), case.cutoff = 0)
```

[4]Note that a truth table is a very different type of object in the context of CNA than it is in the context of QCA. While a QCA truth table indicates for every minterm (which is a configuration of all exogenous factors) whether it is sufficient for the outcome, a CNA truth table is simply an integrated representation of the data that lists all configurations exactly once. A CNA truth table does not express relations of sufficiency.

The first input `x` is a data frame or matrix. The function then merges multiple rows of `x` featuring the same configuration into one row, such that each row of the resulting table corresponds to one determinate configuration of the factors in `x`. The number of occurrences of a configuration and an enumeration of the cases instantiating it are saved as attributes "n" and "cases", respectively. When not applied to *cs* data, the data type must be specified with the `type` argument. Alternatively, the shorthand functions `cstt(x)`, `mvtt(x)` and `fstt(x)` are available.

```
R> truthTab(d.women)
R> mvtt(d.pban)
```

Finally, `truthTab()` provides a numeric argument called `case.cutoff`, which allows for setting a minimum frequency cutoff determining that configurations with less instances in the data are not included in the truth table and the ensuing analysis. For instance, `truthTab(x, case.cutoff = 3)` entails that configurations that are instantiated in less than 3 cases are excluded.

Truth tables produced by `truthTab()` can be directly passed to `cna()`. Moreover, as truth tables generated by `truthTab` are objects that are very particular to the **cna** package, the function `tt2df()` is available to transform truth tables back into ordinary R data frames.

```
R> pact.tt <- truthTab(d.pacts, type = "fs", case.cutoff = 2)
R> tt2df(pact.tt)
```

Data simulations

The **cna** package provides extensive functionalities for data simulations—which, in turn, are essential for inverse search trials benchmarking e.g. the correctness of CNA's output (see section 7). In a nutshell, the functions `allCombs()` and `full.tt()` generate the space of all logically possible configurations over a given set of factors, `selectCases()` selects, from this space, the configurations that are compatible with a data-generating causal structure, which, in turn, can be randomly drawn by `randomAsf()` and `randomCsf()`, `makeFuzzy()` introduces noise into that data, and `some()` randomly selects cases, for instance, to produce data fragmentation.

More specifically, `allCombs(x)` takes an integer vector `x` as input and generates a data frame of all possible value configurations of `length(x)` factors, the first factor having `x[1]` values, the second `x[2]` values etc. The factors are labeled using capital letters in alphabetical order. Analogously, but more flexibly, `full.tt(x)` generates a `truthTab` with all logically possible value configurations of the factors defined in the input `x`, which can be a `truthTab`, a data frame, an integer, a list specifying the factors' value ranges, or a character vector featuring all admissible factor values.

```
R> allCombs(c(2, 2, 2)) - 1
R> allCombs(c(3, 4, 5))
R> full.tt("A + B*c")
R> full.tt(6)
R> full.tt(list(A = 1:2, B = 0:1, C = 1:4))
```

The input of `selectCases(cond, x)` is a character string `cond` specifying a Boolean function, which typically (but not necessarily) expresses a data-generating causal structure, as well as, optionally, a data frame or `truthTab` `x`. If `x` is specified, the function selects the cases that are compatible with `cond` from `x`; if `x` is not specified, it selects from `full.tt(cond)`. It is possible to randomly draw `cond` using `randomAsf(x)` or `randomCsf(x)`, which generate random atomic and complex solutions formulas, respectively, from a data frame or `truthTab` `x`.

```
R> dat1 <- allCombs(c(2, 2, 2)) - 1
R> selectCases("A + B <-> C", dat1)
R> selectCases("(h*F + B*C*k + T*r <-> G)*(A*b + H*I*K <-> E)")
R> target <- randomCsf(full.tt(6))
R> selectCases(target)
```

The closely related function `selectCases1(cond, x, con = 1, cov = 1)` additionally allows for providing consistency (`con`) and coverage (`cov`) thresholds (see section 3.2), such that some cases that are incompatible with `cond`—*viz.* outliers—are also selected, as long as `cond` still meets `con` and `cov` in the resulting data. Thereby, the existence of outliers is simulated.

```
R> dat2 <- full.tt(list(EN = 0:2, TE = 0:4, RU = 1:4))
R> selectCases1("EN=1*TE=3 + EN=2*TE=0 <-> RU=2", dat2, con = .75, cov = .75)
```

`makeFuzzy(x, fuzzvalues = c(0, 0.05, 0.1))` generates fs data by simulating the addition of random noise from the uncontrolled causal background to a *cs* data frame `x`. In addition to `x`, it takes as input a vector of `fuzzvalues` to be randomly added to the 0's and subtracted from the 1's in `x`.

```
R> makeFuzzy(selectCases("Hunger + Heat <-> Run"),
+             fuzzvalues = seq(0, 0.4, 0.05))
```

Finally, `some(x, n = 10, replace = TRUE)` randomly selects `n` cases from a data frame or `truthTab` `x`, with or without replacement. If `x` features all configurations that are compatible with a data-generating structure and `n < nrow(x)`, the data frame or `truthTab` issued by `some()` is *fragmented*, meaning it does not contain all empirically possible configurations. For example:

```
R> dat3 <- allCombs(c(3, 4, 5))
R> dat4 <- selectCases("A=1*B=3 + A=3 <-> C=2", mvtt(dat3))
R> some(dat4, n = 10, replace = FALSE)
```

3.2. Consistency and coverage

As real-life data tend to feature noise induced by unmeasured causes of endogenous factors, strictly sufficient or necessary conditions for an outcome often do not exist. To approximate the deterministic dependency structures that figure in the regularity theoretic definition of causation, Ragin (2006) imported so-called *consistency* and *coverage* measures (with values from the interval $[0, 1]$) into the QCA protocol. Both of these measures are also serviceable for

the purposes of CNA. Informally put, *consistency* reflects the degree to which the behavior of an outcome obeys a corresponding sufficiency or necessity relationship or a whole model, whereas *coverage* reflects the degree to which a sufficiency or necessity relationship or a whole model accounts for the behavior of the corresponding outcome. As the implication operator underlying the notions of sufficiency and necessity is defined differently in classical and in fuzzy logic, the two measures are defined differently for crisp-set and multi-value data (which both have a classical footing), on the one hand, and fuzzy-set data, on the other. *Cs-consistency* (con^{cs}) of $X \to Y$ is defined as the number of cases featuring $X{*}Y$ divided by the number of cases featuring X, and *cs-coverage* (cov^{cs}) of $X \to Y$ amounts to the number of cases featuring $X{*}Y$ divided by the number of cases featuring Y (where $|\ldots|$ represents the cardinality of the set of cases instantiating the corresponding expression):

$$con^{cs}(X \to Y) = \frac{|X{*}Y|}{|X|} \qquad cov^{cs}(X \to Y) = \frac{|X{*}Y|}{|Y|}$$

Fs-consistency (con^{fs}) and *fs-coverage* (cov^{fs}) of $X \to Y$ are defined as follows, where n is the number of cases in the data:

$$con^{fs}(X \to Y) = \frac{\sum_{i=1}^{n} \min(X_i, Y_i)}{\sum_{i=1}^{n} X_i} \qquad cov^{fs}(X \to Y) = \frac{\sum_{i=1}^{n} \min(X_i, Y_i)}{\sum_{i=1}^{n} Y_i}$$

Although defined differently, the cs and fs variants of these measures are not logically independent: con^{cs} and cov^{cs} are special cases of con^{fs} and cov^{fs} where all membership scores are equal to 0 or 1. As the data type processed in a concrete analysis determines the appropriate consistency and coverage measures, we will henceforth not explicitly distinguish between the cs and fs measures.

Consistency and coverage thresholds can be given to the `cna()` function using the arguments `con.msc`, `con`, and `cov` that take values from the interval $[0, 1]$. `con.msc` sets the consistency threshold for minimally sufficient conditions (*msc*), `con` does the same for *asf* and *csf*, while `cov` sets the coverage threshold for *asf* and *csf* (no coverage threshold is imposed on *msc*). As illustrated on pp. 16-17 of the **cna** reference manual, setting different consistency thresholds for *msc* and *asf/csf* can enhance the informativeness of `cna()`'s output in certain cases but is non-standard. The standard setting is `con = con.msc`.

The default numeric value for all thresholds is 1, i.e. perfect consistency and coverage. Contrary to QCA, which often returns solutions that do not comply with the chosen consistency threshold and which does not impose a coverage threshold at all, CNA uses consistency and coverage as authoritative model building criteria such that, if they are not met, CNA abstains from issuing solutions. That means, if the default thresholds are used, `cna()` will only output perfectly consistent *msc*, *asf*, and *csf* and only perfectly covering *asf* and *csf*.

Frequently, though, the default thresholds will not yield any solution formulas—due to noise and outliers. In such cases, `con` and `cov` may be gradually lowered (e.g. in steps of 0.1) until `cna()` builds solution formulas. For example, by lowering `con` to 0.8 in a *cs* analysis, `cna()` is given permission to treat X as sufficient for Y, even though in 20% of the cases X is not associated with Y. Or by lowering `cov` to 0.8 in an fs analysis, `cna()` is allowed to treat X as necessary for Y, even though the sum of the membership scores in Y over all cases in the data exceeds the sum of the membership scores in $\min(X, Y)$ by 20%.

To illustrate, `cna()` does not build solutions for the `d.jobsecurity` data at the following `con` and `cov` thresholds:

```
R> fscna(d.jobsecurity, con = 1, cov = .9)
R> fscna(d.jobsecurity, con = .9, cov = 1)
R> fscna(d.jobsecurity, con = .9, cov = .9)
```

But if `con` is lowered further, multiple equally well fitting solutions are issued (the function `csf()` used below merely extracts the *csf* from a `cna()` solution object; see section 5):

```
R> ana.job.1 <- fscna(d.jobsecurity, con = .8, cov = .9)
R> printCols <- c("condition", "consistency", "coverage")
R> csf(ana.job.1)[printCols]
```

```
  condition                  consistency coverage
1 S*R + C*L + C*l <-> JSR          0.804    0.910
2 C*L + C*l + c*R <-> JSR          0.808    0.902
3 S*R + C*V + s*C*v <-> JSR        0.825    0.914
4 c*R + C*V + s*C*v <-> JSR        0.822    0.912
5 S*R + C*L + s*C*r <-> JSR        0.804    0.910
```

Lowering `con` and `cov` must be done with great caution, for the lower these thresholds, the higher the chance that causal fallacies are committed, i.e. that spurious associations are mistaken for causal ones. Neither threshold should be lowered below 0.75. If `cna()` does not find solutions at `con = cov = .75`, the corresponding data feature such a high degree of noise that causal inferences become too hazardous (cf. Baumgartner and Ambühl 2018).

3.3. Ordering

CNA does not need to be told which factors in the analyzed data δ are endogenous (i.e. effects) and which ones are exogenous (i.e. causes). It attempts to infer that from δ. But if prior causal knowledge is available as to which factors can figure as effects and which ones cannot, this information can be given to CNA via a *causal ordering*. A causal ordering is a relation $X_i \prec X_j$ defined on the factors in δ entailing that X_j cannot be a cause of X_i (e.g. because X_i is instantiated temporally before X_j). That is, an ordering excludes certain causal dependencies but does not stipulate any. If an ordering is provided, CNA only searches for Boolean models in accordance with the ordering; if no ordering is provided, CNA treats all values of the factors in δ as potential outcomes and explores whether a causal model for them can be inferred.

An ordering is given to `cna()` via the argument `ordering`, which takes as value a list of character vectors specifying the causal ordering of the factors in `x`. For example, `ordering = list(c("A","B"),"C")` determines that C is causally located after A and B (i.e. $A, B \prec C$), meaning that C is not a potential cause of A and B. The latter are located on the same level of the ordering, for A and B are unrelated by $\prec$, whereas C is located on a level that is downstream of the A, B-level. `cna()` then only checks whether values of A and B can be modeled as causes of values of C; the test for a causal dependency in the upstream direction is skipped. If the argument ordering is not specified, `cna()` searches for dependencies between all factors in `x`. An ordering does not need to explicitly mention all factors in `x`. If only a subset of the factors are included in the ordering, the non-included factors are entailed to be causally before the included ones. Hence, `ordering = list("C")` means that C is causally located after all other factors in `x`.

Additionally, the logical argument `strict` is available. It determines whether the elements of one level in an ordering can be causally related or not. For example, if `ordering = list(c("A","B"),"C")` and `strict = TRUE`, then A and B are excluded to be causally related and `cna()` skips corresponding tests. By contrast, if `ordering = list(c("A","B"),"C")` and `strict = FALSE`, then `cna()` also searches for dependencies among A and B.

Let us illustrate with the data set `d.autonomy`. Relative to the following function call, which stipulates that AU cannot be a cause of EM, SP, and CO and that the latter factors are not mutually causally related, `cna()` infers that SP is causally relevant to AU (i.e. $SP \leftrightarrow AU$):

```
R> dat.aut.1 <- d.autonomy[15:30, c("AU","EM","SP","CO")]
R> ana.aut.1 <- fscna(dat.aut.1, ordering = list(c("EM","SP","CO"), "AU"),
+    strict = TRUE, con = .91, cov = .91)
R> csf(ana.aut.1)[printCols]

  condition consistency coverage
1 SP <-> AU       0.935    0.915
```

If we set `strict` to `FALSE` and, thereby, allow for causal dependencies among EM, SP, and CO, it turns out that SP not only causes AU, but, on another causal path, also makes a difference to EM:

```
R> ana.aut.2 <- fscna(dat.aut.1, ordering = list(c("EM","SP","CO"), "AU"),
+    strict = FALSE, con = .91, cov = .91)
R> csf(ana.aut.2)[printCols]

  condition                     consistency coverage
1 (SP <-> AU)*(SP + CO <-> EM)       0.912    0.915
```

3.4. Maxstep

As anticipated in section 1 and as will be exhibited in more detail in section 4, `cna()` builds minimally necessary disjunctions of minimally sufficient conditions (i.e. *asf*) from the bottom up by gradually permutating and testing conjunctions and disjunctions of increasing complexity for sufficiency and necessity. The combinatorial search space that this algorithm has to scan depends on a variety of different aspects, for instance, on the number of factors in `x`, on the number of values these factors can take, on the number and length of the *msc* recovered in the first computational phase, etc. As the search space may be too large to be exhaustively scanned in reasonable time, the argument *maxstep* allows for setting an upper bound for the complexity of the generated *asf*. `maxstep` takes a vector of three integers $c(i, j, k)$ as input, entailing that the generated *asf* have maximally j disjuncts with maximally i conjuncts each and a total of maximally k factors. The default is `maxstep = c(3,3,9)`. The user can set it to any complexity level if computational time is not an issue.

The `maxstep` argument is particularly relevant for the analysis of data featuring severe model ambiguities. A telling case in point is the data set `d.volatile`. At the default `maxstep`, `cna()` recovers 416 *csf*. By increasing `maxstep`, this number increases rapidly (from 2860 to 4264 to 30012 *csf*).

```
R> cna(d.volatile, ordering = list("VO2"))
R> cna(d.volatile, ordering = list("VO2"), maxstep = c(4,3,10))
R> cna(d.volatile, ordering = list("VO2"), maxstep = c(4,3,11))
R> cna(d.volatile, ordering = list("VO2"), maxstep = c(4,4,11))
```

If the values of `maxstep` are further increased, the analysis will quickly fail to terminate in reasonable time. When a complete analysis cannot be completed, `cna()` can be told to only search for minimally sufficient conditions (*msc*) by setting the argument `suff.only` to its non-default value `TRUE`. As the search for *msc* is the part of a CNA analysis that is least computationally demanding, it will typically terminate quickly and, thus, shed some light on the dependencies among the factors in `x` even when a complete analysis is infeasible.

```
R> cna(d.volatile, ordering = list("VO2"), maxstep = c(8,10,40),
+    suff.only = TRUE)
```

While the `maxstep` argument is very valuable for controlling the search space in case of large and ambiguous data sets, it also comes with a pitfall: it may happen that `cna()` fails to find a model because of a `maxstep` that is too low. An example is `d.women`. At the default `maxstep`, `cna()` does not build a solution, but if `maxstep` is increased, two solutions with perfect consistency and coverage are found.

```
R> ana.wom.1 <- cna(d.women)
R> csf(ana.wom.1)[printCols]

[1] condition   consistency coverage
<0 rows> (or 0-length row.names)

R> ana.wom.2 <- cna(d.women, maxstep = c(3,4,10))
R> csf(ana.wom.2)[printCols]

  condition                          consistency coverage
1 WS + ES*WM + es*LP + QU*LP <-> WNP           1        1
2 WS + ES*WM + QU*LP + WM*LP <-> WNP           1        1
```

In sum, there are two possible reasons for why `cna()` fails to build a solution: (i) the chosen `maxstep` is too low; (ii) the chosen `con` and/or `cov` values are too high, meaning the processed data `x` are too noisy. Accordingly, in case of a null result, two paths should be explored (in that order): (i) gradually increase `maxstep`; (ii) gradually lower `con` and `cov`, as described in section 3.2 above.

3.5. Notcols

In classical Boolean logic, the law of Contraposition ensures that an expression of type $\Psi \leftrightarrow Y$ is equivalent to the expression that results from negating both sides of the double arrow: $\neg\Psi \leftrightarrow \neg Y$. Applied to the context of configurational causal modeling that entails that an *asf* for Y can be transformed into an *asf* for the negation of Y, *viz.* y, based on logical principles alone, i.e. without a separate data analysis. However, that transformability only

holds for *asf* with perfect consistency and coverage (`con = cov = 1`) that are inferred from exhaustive (non-fragmented) data (see section 5.3 for details on exhaustiveness). If an *asf* of an outcome Y does not reach perfect consistency or coverage or is inferred from fragmented data, identifying the causes of y requires a separate application of `cna()` explicitly targeting the causes of the negated outcome.

To this end, the argument `notcols` allows for negating the values of factors in *cs* and *fs* data (in case of *mv* data, `cna()` automatically searches for models of all possible values of endogenous factors, thereby rendering `notcols` redundant). If `notcols = "all"`, all factors are negated, i.e. their membership scores i are replaced by $1-i$. If `notcols` is given a character vector of factors in the data, only the factors in that vector are negated. For example, `notcols = c("A", "B")` determines that only factors A and B are negated.

When processing *cs* or *fs* data, CNA should first be applied to recover causal models for the positive outcomes. If resulting *asf* and *csf* do not reach perfect consistency, coverage, and exhaustiveness scores, a second CNA should be run negating the values of all factors that have been modeled as outcomes in the first CNA. To illustrate, we revisit our analyses of `d.jobsecurity` from section 3.2, which identified JSR as outcome, and of `d.autonomy` from section 3.3, which identified AU and EM as outcomes.

```
R> fscna(d.jobsecurity, con = .8, cov = .9, notcols = "JSR")
R> fscna(dat.aut.1, ordering = list(c("EM","SP","CO"), "AU"),
+    strict = FALSE, con = .88, cov = .82, notcols = c("AU", "EM"))
```

4. The CNA algorithm

This section explains the working of the algorithm implemented in the `cna()` function. We first provide an informal summary and then a detailed outline in four stages. The aim of `cna()` is to find all *msc*, *asf*, and *csf* that meet `con.msc`, `con` and `cov` in the input data `x` in accordance with `ordering` and `maxstep`. The algorithm starts with single factor values and tests whether they meet `con.msc`; if that is not the case, it proceeds to test conjunctions of two factor values, then to conjunctions of three, and so on. Whenever a conjunction meets `con.msc` (and no proper part of it has previously been identified to meet `con.msc`), it is automatically a minimally sufficient condition *msc*, and supersets of it do not need to be tested any more. Then, it tests whether single *msc* meet `con` and `cov`; if not, it proceeds to disjunctions of two, then to disjunctions of three, and so on. Whenever a disjunction meets `con` and `cov` (and no proper part of it has previously been identified to meet `con` and `cov`), it is automatically a minimally necessary disjunction of *msc*, and supersets of it do not need to be tested any more. All and only those disjunctions of *msc* that meet both `con` and `cov` are then issued as *asf*, which, finally, are concatenated to *csf*.

The `cna()` algorithm can be more specifically broken down into four stages.

Stage 1 On the basis of `ordering`, `cna()` first builds a set of potential outcomes $\mathbf{O} = \{O_h{=}\omega_f, \ldots, O_m{=}\omega_g\}$ from the set of factors $\mathbf{F} = \{O_1, \ldots, O_n\}$ in `x`,[5] where

[5]Note that if `x` is a data frame, cna() first transforms `x` into a truth table using `truthTab(x)`, thereby passing the argument `type` (and the two additional arguments `rm.dup.factors` and `rm.const.factors`) to the `truthTab()` function.

$1 \leq h \leq m \leq n$, and second assigns a set of potential cause factors $\mathbf{C}_{O_i}$ from $\mathbf{F} \setminus \{O_i\}$ to every element $O_i{=}\omega_k$ of $\mathbf{O}$. If no `ordering` is provided, all value assignments to all elements of $\mathbf{F}$ are treated as possible outcomes in case of *mv* data, whereas in case of *cs* and *fs* data $\mathbf{O}$ is set to $\{O_1{=}1, \ldots, O_n{=}1\}$.

Stage 2 `cna()` attempts to build a set $\mathbf{msc}_{O_i=\omega_k}$ of minimally sufficient conditions that meet `con.msc` for each $O_i{=}\omega_k \in \mathbf{O}$. To this end, it first checks for each value assignment $X_h{=}\chi_j$ of each element of $\mathbf{C}_{O_i}$, such that $X_h{=}\chi_j$ has a membership score above 0.5 in at least one case in `x`, whether the consistency of $X_h{=}\chi_j \rightarrow O_i{=}\omega_k$ meets `con.msc`, i.e. whether $con(X_h{=}\chi_j \rightarrow O_i{=}\omega_k) \geq$ `con.msc`. If, and only if, that is the case, $X_h{=}\chi_j$ is put into the set $\mathbf{msc}_{O_i=\omega_k}$. Next, `cna()` checks for each conjunction of two factor values $X_m{=}\chi_j * X_n{=}\chi_l$ from $\mathbf{C}_{O_i}$, such that $X_m{=}\chi_j * X_n{=}\chi_l$ has a membership score above 0.5 in at least one case in `x` and no part of $X_m{=}\chi_j * X_n{=}\chi_l$ is already contained in $\mathbf{msc}_{O_i=\omega_k}$, whether $con(X_m{=}\chi_j * X_n{=}\chi_l \rightarrow O_i{=}\omega_k) \geq$ `con.msc`. If, and only if, that is the case, $X_m{=}\chi_j * X_n{=}\chi_l$ is put into the set $\mathbf{msc}_{O_i=\omega_k}$. Next, conjunctions of three factor values with no parts already contained in $\mathbf{msc}_{O_i=\omega_k}$ are tested, then conjunctions of four factor values, etc., until either all logically possible conjunctions of the elements of $\mathbf{C}_{O_i}$ have been tested or `maxstep` is reached. Every non-empty $\mathbf{msc}_{O_i=\omega_k}$ is passed on to the third stage.

Stage 3 `cna()` attempts to build a set $\mathbf{asf}_{O_i=\omega_k}$ of atomic solution formulas for every $O_i{=}\omega_k \in \mathbf{O}$, which has a non-empty $\mathbf{msc}_{O_i=\omega_k}$, by disjunctively concatenating the elements of $\mathbf{msc}_{O_i=\omega_k}$ to minimally necessary conditions of $O_i{=}\omega_k$ that meet `con` and `cov`. To this end, it first checks for each single condition $\Phi_h \in \mathbf{msc}_{O_i=\omega_k}$ whether $con(\Phi_h \rightarrow O_i{=}\omega_k) \geq$ `con` and $cov(\Phi_h \rightarrow O_i{=}\omega_k) \geq$ `cov`. If, and only if, that is the case, Φ_h is put into the set $\mathbf{asf}_{O_i=\omega_k}$. Next, `cna()` checks for each disjunction of two conditions $\Phi_m + \Phi_n$ from $\mathbf{msc}_{O_i=\omega_k}$, such that no part of $\Phi_m + \Phi_n$ is already contained in $\mathbf{asf}_{O_i=\omega_k}$, whether $con(\Phi_m + \Phi_n \rightarrow O_i{=}\omega_k) \geq$ `con` and $cov(\Phi_m + \Phi_n \rightarrow O_i{=}\omega_k) \geq$ `cov`. If, and only if, that is the case, $\Phi_m + \Phi_n$ is put into the set $\mathbf{asf}_{O_i=\omega_k}$. Next, disjunctions of three conditions from $\mathbf{msc}_{O_i=\omega_k}$ with no parts already contained in $\mathbf{asf}_{O_i=\omega_k}$ are tested, then disjunctions of four conditions, etc., until either all logically possible disjunctions of the elements of $\mathbf{msc}_{O_i=\omega_k}$ have been tested or `maxstep` is reached. Every non-empty $\mathbf{asf}_{O_i=\omega_k}$ is passed on to the fourth stage.

Stage 4 `cna()` attempts to build a set $\mathbf{csf_O}$ of complex solution formulas encompassing all elements of $\mathbf{O}$. To this end, all logically possible conjunctions of exactly one element from every non-empty $\mathbf{asf}_{O_i=\omega_k}$ are constructed. If there is only one non-empty set $\mathbf{asf}_{O_i=\omega_k}$, that is, if only one potential outcome can be modeled as an actual outcome, the set of complex solution formulas $\mathbf{csf_O}$ is identical to $\mathbf{asf}_{O_i=\omega_k}$.

To illustrate, the following code chunk, first, simulates the data in Table 2c, p. 9, and second, runs `cna()` on that data with `con = .8` and `cov = .9`, with default `maxstep`, and without `ordering`.

```
R> dat5 <- allCombs(c(2, 2, 2, 2, 2)) -1
R> dat6 <- selectCases("(A + B <-> C)*(A*B + D <-> E)", dat5)
R> set.seed(28)
R> tab2c <- makeFuzzy(tt2df(dat6), fuzzvalues = seq(0, 0.4, 0.01))
R> fscna(tab2c, con = .8, cov = .9, what = "mac")
```

Table 2c contains data of type fs, meaning that the values in the data matrix are interpreted as membership scores in fuzzy sets. As is customary for this data type, we use uppercase letters for membership in a set and lowercase letters for non-membership. In the absence of an ordering, the set of potential outcomes is determined to be $\mathbf{O} = \{A, B, C, D, E\}$ in stage 1, that is, the presence of each factor in Table 2c is treated as a potential outcome. Moreover, all other factors are potential cause factors of every element of $\mathbf{O}$, hence, $\mathbf{C}_A = \{B, C, D, E\}$, $\mathbf{C}_B = \{A, C, D, E\}$, $\mathbf{C}_C = \{A, B, D, E\}$, $\mathbf{C}_D = \{A, B, C, E\}$, and $\mathbf{C}_E = \{A, B, C, D\}$.

In stage 2, `cna()` succeeds in building non-empty sets of minimally sufficient conditions for all elements of $\mathbf{O}$: $\mathbf{msc}_A = \{b{*}C, d{*}E\}$, $\mathbf{msc}_B = \{a{*}C, A{*}E, d{*}E\}$, $\mathbf{msc}_C = \{A, B, d{*}E\}$, $\mathbf{msc}_D = \{E, a{*}C\}$, $\mathbf{msc}_E = \{D, A{*}B\}$. But only the elements of $\mathbf{msc}_C$ and $\mathbf{msc}_E$ can be disjunctively combined to atomic solution formulas that meet `cov` in stage 3: $\mathbf{asf}_C = \{A + B \leftrightarrow C\}$ and $\mathbf{asf}_E = \{D + A{*}B \leftrightarrow E\}$. For the other three factors in $\mathbf{O}$, the coverage threshold of 0.9 cannot be satisfied. `cna()` therefore abstains from issuing *asf* for A, B and D.

Finally, stage 4 conjunctively concatenates the *asf* in $\mathbf{asf}_C$ and $\mathbf{asf}_E$ to the *csf* in the set $\mathbf{csf}_\mathbf{O}$, which constitutes `cna()`'s final output for Table 2c:

$$(A \ + \ B \leftrightarrow C) * (D \ + \ A{*}B \leftrightarrow E) \qquad con = 0.808;\ cov = 0.925 \tag{3}$$

5. The output of CNA

5.1. Customizing the output

The default output of `cna()` first lists the provided ordering, second, the *asf* that were recovered in accordance with the ordering, and third, the *csf*. For *asf* and *csf*, three attributes are standardly computed: consistency, coverage, and complexity. Consistency and coverage, have been explained in section 3.2 above; the complexity score simply amounts to the number of factors on the left-hand sides of "→" or "↔" in *asf* and *csf*.

`cna()` can compute a number of additional solution attributes, all of which will be explained below: `inus`, `exhaustiveness`, and `faithfulness` for both *asf* and *csf*, as well as `coherence` and `redundant` for *csf*. These attributes are accessible via the `details` argument, which can be given the values `TRUE`/`FALSE`, for computing all/none of the additional attributes, or a character vector specifying the specific attributes to be computed: for example, `details = c("inus", "exhaustiveness")`—the strings can also be abbreviated, e.g. `"i"` for `"inus"`, `"f"` for `"faithfulness"`, etc.

```
R> cna(d.educate, details = TRUE)
R> cna(d.educate, details = c("i", "e", "r"))
```

The output of the `cna()` function can be further customized through the argument `what` that controls which solution items to print. It can be given a character string specifying the requested solution items: `"t"` stands for the truth table, `"m"` for minimally sufficient conditions (*msc*), `"a"` for *asf*, `"c"` for *csf*, and `"all"` for all solution items.

```
R> cna(d.educate, what = "tm")
R> cna(d.educate, what = "mac")
R> cna(d.educate, what = "all")
```

As shown in section 3.4, it can happen that many *asf* and *csf* fit the data equally well. `cna()` standardly only returns 5 solution items of each type. All *msc*, *asf* and *csf* can be recovered using the functions `msc(x)`, `asf(x)`, and `csf(x)`, where `x` is a solution object generated by `cna()`.

```
R> vol1 <- cna(d.volatile, ordering = list("VO2"))
R> msc(vol1)
R> asf(vol1)
R> csf(vol1)
```

5.2. INUS vs. non-INUS solutions

Regularity theories of causation as Mackie's (1974) INUS-theory have been developed for strictly Boolean discovery contexts, meaning for noise-free (i.e. deterministic) data that feature perfectly sufficient and necessary conditions. In such contexts, some Boolean expressions can be identified as non-minimal (i.e. as featuring redundant elements) on mere *logical grounds*, that is, independently of data. For instance, in an expression as

$$A + a{*}B \leftrightarrow C \tag{4}$$

a in the second disjunct is redundant, for (4) is logically equivalent to $A + B \leftrightarrow C$. These two formulas state exactly the same. Under no conceivable circumstances could a as contained in (4) ever make a difference to C. To see this, note that a necessary condition for $a{*}B$ to be a complex cause of C is that there exists a context $\mathcal{F}$ such that C is only instantiated when *both* a and B are given. That means that, in $\mathcal{F}$, C is not instantiated if B is given but a is not, which, in turn, means that C is not instantiated if B is given and A (*viz.* not-a) is given. But such an $\mathcal{F}$ cannot possibly exist, for A itself is sufficient for C according to (4). It follows that in every context where B is instantiated, a change from A to a is not associated with a change in C (which takes the value 1 throughout the change in the factor A), meaning that a cannot possibly make a difference to C and, hence, cannot be a cause of C, subject to a regularity theory of causation. That is, (4) can be identified as non-minimal independently of all data. (4) is not a well-formed causal model. It is not a minimal theory—not an *INUS solution*.

Correspondingly, the solution attribute `inus` indicates whether an *asf* or *csf* is an INUS solution.

When CNA is applied to noise-free data, it will never build a solution that is not INUS. This can be illustrated by simulating noise-free data on the non-INUS solution in (4); `cna()` will always, i.e. upon an open number of re-runs of the following code chunk, return $A + B \leftrightarrow C$, regardless of `selectCases1("A + a*B <-> C", ...)`.

```
R> dat.inu.1 <- allCombs(c(2, 2, 2)) -1
R> dat.inu.2 <- some(dat.inu.1, 40, replace = TRUE)
R> dat.inu.3 <- selectCases1("A + a*B <-> C",  con = 1, cov = 1, dat.inu.2)
R> printCols <- c("condition", "consistency", "coverage", "complexity",
+    "inus")
R> asf(cna(dat.inu.3, con = 1, cov = 1, details = "inus"))[printCols]
```

```
  condition   consistency coverage complexity inus
1 A + B <-> C           1        1          2 TRUE
```

But, as indicated in section 3.2, in real-life discovery contexts, especially in observational studies, deterministic dependencies are the exception rather than the norm. Ordinary (observational) data are noisy, meaning that causes tend to be combined both with the presence and the absence of outcomes. In such discovery contexts, which can be simulated by lowering `con` and `cov` in `selectCases1()`, it is no longer guaranteed that only INUS solutions are returned.

```
R> set.seed(4)
R> dat.inu.4 <- some(dat.inu.1, 40, replace = TRUE)
R> dat.inu.5 <- selectCases1("A + a*B <-> C", con = .8, cov = .8, dat.inu.4)
R> asf(cna(dat.inu.5, con = .8, cov = .8, details = "inus"))[printCols]
```

```
  condition     consistency coverage complexity  inus
1 A + a*B <-> C        0.81     0.81          3 FALSE
```

In scenarios where `con` and `cov` are below 1 it can happen that a is needed to lift the consistency of B above the chosen thresholds. In such a case, a can indeed be argued to make a difference to C: only in conjunction with a does B reach the `con` threshold; and this holds notwithstanding the fact that A itself also meets `con`. Or put differently, when `con` is below 1, there exist cases where A is instantiated and C is not, which, in turn, yields that it becomes possible for a change from A to a, while B is constantly instantiated, to be associated with a change in C, meaning that a can turn out to be a difference-maker for C.

That factors may count as difference-makers for outcomes in noisy contexts, which could not make a difference to these outcomes in noise-free contexts, is a phenomenon that not only occurs in simulated but also in real-life data, for instance, in `d.jobsecurity`:

```
R> ana.job.3 <- fscna(d.jobsecurity, con = .8, cov = .9, details = "inus")
R> asf(ana.job.3)[printCols]
```

```
  condition                 consistency coverage complexity  inus
1 S*R + C*L + C*l <-> JSR         0.804    0.910          6 FALSE
2 C*L + C*l + c*R <-> JSR         0.808    0.902          6 FALSE
3 S*R + C*V + s*C*v <-> JSR       0.825    0.914          7 FALSE
4 c*R + C*V + s*C*v <-> JSR       0.822    0.912          7 FALSE
5 S*R + C*L + s*C*r <-> JSR       0.804    0.910          7  TRUE
```

The first non-INUS *asf* in `ana.job.3` is logically reducible to the INUS solution $S{*}R + C \leftrightarrow JSR$, the second is reducible to $C + R \leftrightarrow JSR$, while in the third and fourth non-INUS *asf*, v is logically redundant in the last disjunct $s{*}C{*}v$.

If `cna()` returns non-INUS solutions, the crucial follow-up question is whether the indeterminism in the data is due to insufficient control of background influences (i.e. to noise) or to the inherent indeterministic nature of the physical processes themselves (as can e.g. be found in the domain of quantum mechanics, cf. Albert 1992). If (and only if) the former is the case

and, hence, the scrutinized causal structure can be assumed to be of deterministic nature, the difference-making relations stipulated by non-INUS solutions should be disregarded as being mere artifacts of the noise in the data, meaning that they would disappear if the corresponding causal structure were investigated under less noisy discovery circumstances. In that case (which typically obtains in macro domains), the logical argument `inus.only` should be set to its non-default value `TRUE` in the `cna()` function such that non-INUS solutions are not built to being with:

```
R> ana.job.4 <- fscna(d.jobsecurity, con = .8, cov = .9, details = "inus",
+    inus.only = TRUE, what = "a")
R> asf(ana.job.4)[printCols]

  condition                 consistency coverage complexity inus
1 S*R + C*L + s*C*r <-> JSR       0.804     0.91          7 TRUE
```

The function behind the solution attribute `inus` is also available as stand-alone function `is.inus()`. Logical redundancies as contained in non-INUS solutions can be eliminated by means of the function `minimalize()` (see the **cna** reference manual for details).

5.3. Exhaustiveness and faithfulness

Exhaustiveness and faithfulness are two measures of model fit that quantify the degree of correspondence between the configurations that are, in principle, compatible with a solution and the configurations actually contained in the data from which that solution is derived. Exhaustiveness is high when *all* or most configurations *compatible* with a solution are in the data. More specifically, it amounts to the ratio of the number of configurations in the data that are compatible with a solution to the number of configurations in total that are compatible with a solution. To illustrate, consider `d.educate`, which contains all configurations that are compatible with the two *csf* issued by `cna()`:

```
R> printCols <- c("condition", "consistency", "coverage", "exhaustiveness")
R> csf(cna(d.educate, details = "exhaust"))[printCols]

  condition                       consistency coverage exhaustiveness
1 (L + G <-> E)*(U + D <-> L)               1        1              1
2 (U + D + G <-> E)*(U + D <-> L)           1        1              1
```

If, say, the first configuration in `d.educate` (*viz.* $U{*}D{*}L{*}G{*}E$) is not observed or removed—as in `d.educate[-1,]`—, `cna()` still builds the same solutions (with perfect consistency and coverage). In that case, however, the resulting *csf* are not exhaustively represented in the data, for one configuration that is compatible with both *csf* is not contained in the data.

```
R> csf(cna(d.educate[-1,], details = "exhaust"))[printCols]

  condition                       consistency coverage exhaustiveness
1 (L + G <-> E)*(U + D <-> L)               1        1          0.875
2 (U + D + G <-> E)*(U + D <-> L)           1        1          0.875
```

In a sense, faithfulness is the complement of exhaustiveness. It is high when *no* or only few configurations that are *incompatible* with a solution are in the data. More specifically, faithfulness amounts to the ratio of the number of configurations in the data that are compatible with a solution to the total number of configurations in the data. The two *csf* resulting from `d.educate` also reach perfect faithfulness:

```
R> printCols <- c("condition", "consistency", "coverage", "faithfulness")
R> csf(cna(d.educate, details = "faithful"))[printCols]

  condition                          consistency coverage faithfulness
1 (L + G <-> E)*(U + D <-> L)                1        1            1
2 (U + D + G <-> E)*(U + D <-> L)            1        1            1
```

If we add a configuration that is not compatible with these *csf*, say, $U{*}D{*}l{*}G{*}e$ and lower the consistency threshold, the same solutions along with two others result—this time, however, with non-perfect faithfulness scores.

```
R> csf(cna(rbind(d.educate,c(1,1,0,1,0)), con = .8, details = "f"))[printCols]

  condition                          consistency coverage faithfulness
1 (L + G <-> E)*(E <-> L)                0.857        1        0.778
2 (L + G <-> E)*(U + D <-> L)            0.857        1        0.889
3 (U + D + G <-> E)*(E <-> L)            0.857        1        0.778
4 (U + D + G <-> E)*(U + D <-> L)        0.857        1        0.889
```

If both exhaustiveness and faithfulness are high, the configurations in the data are all and only the configurations that are compatible with the solution. Low exhaustiveness and/or faithfulness, by contrast, means that the data do not contain many configurations compatible with the solution and/or the data contain many configurations not compatible with the solution. In general, solutions with higher exhaustiveness and faithfulness scores are preferable over solutions with lower scores.

5.4. Coherence

Coherence is a measure for model fit that is custom-built for *csf*. It measures the degree to which the *asf* combined in a *csf* cohere, that is, are instantiated together in the data rather than independently of one another. Coherence is intended to capture the following intuition. Suppose a *csf* entails that A is a sufficient cause of B, which, in turn, is entailed to be a sufficient cause of C. Corresponding data δ should be such that the $A-B$ link of that causal chain and the $B-C$ link are either both instantiated or both not instantiated in the cases recored in δ. By contrast, a case in δ such that, say, only the $A-B$ link is instantiated but the $B-C$ link is not, pulls down the coherence of that *csf*. The more such non-cohering cases are contained in δ, the lower the overall coherence score of the *csf*.

Coherence is more specifically defined as the ratio of the number of cases satisfying all *asf* contained in a *csf* to the number of cases satisfying at least one *asf* in the *csf*. More formally, let a *csf* contain $asf_1, asf_2, \ldots, asf_n$, coherence then amounts to (where $|\ldots|$ represents the

cardinality of the set of cases instantiating the corresponding expression):

$$\frac{|asf_1{*}asf_2{*}\ldots{*}asf_n|}{|asf_1 + asf_2 + \ldots + asf_n|}$$

To illustrate, we add a case of type $U{*}d{*}L{*}g{*}e$ to `d.educate`. When applied to the resulting data (`d.edu.exp1`), `cna()` issues four *csf*.

```
R> d.edu.exp1 <- rbind(d.educate, c(1,0,1,0,0))
R> printCols <- c("condition", "consistency", "coverage", "coherence")
R> csf(cna(d.edu.exp1, con = .8, details = "cohere"))[printCols]
```

```
  condition                          consistency coverage coherence
1 (L + G <-> E)*(U + D <-> L)              0.875        1     0.889
2 (L + G <-> E)*(U + E <-> L)              0.875        1     0.778
3 (U + D + G <-> E)*(U + D <-> L)          0.875        1     0.889
4 (U + D + G <-> E)*(U + E <-> L)          0.875        1     0.778
```

In the added case, none of these four *csf* cohere, as only one of their component *asf* is instantiated. Moreover, for the second and the fourth *csf* there is yet another non-cohering case in `d.edu.exp1` (case #7).

Coherence is an additional parameter of model fit that allows for selecting among multiple solutions: the higher the coherence score of a *csf*, the better the overall model fit. In `d.edu.exp1`, thus, the first and the third *csf* are preferable over the other solutions subject to their superior coherence.

5.5. Structural redundancies

The last `cna()` solution attribute that requires explanation is `redundant`, which identifies *csf* containing so-called *structural redundancies*. It is not only possible that Boolean expressions describing the behavior of single endogenous factors contain redundant proper parts, but such expressions can themselves—as a whole—be redundant in superordinate structures, in violation of (NR). More concretely, when *asf* are conjunctively concatenated to *csf*, it can happen that the resulting conjunction of *asf* contains a structural redundancy because it is logically equivalent to a proper part of itself. For a detailed discussion of this problem see Baumgartner and Falk (2018).

The problem is best introduced with an example. Consider the following causal model:

$$(A{*}B + C \leftrightarrow D) * (a + c \leftrightarrow E) \tag{5}$$

(5) represents a causal structure such that $A{*}B$ and C are the two alternative causes of D and a and c are the two alternative causes of E. That is, factors A and C are positively relevant to D and negatively relevant to E. A possible interpretation of these factors might be the following. Suppose a city has two power stations: a wind farm and a nuclear plant. Let A express that the wind farm is operational and C that the nuclear plant is operational and let operationality be sufficient for a nuclear plant to produce electricity, while a wind farm produces electricity provided it is operational and there is wind (B). Hence, the wind

farm being operational while it is windy or the nuclear plant being operational ($A{*}B + C$) are two alternative causes of the city being power supplied (D). Whereas the wind farm or the nuclear plant not being operational ($a + c$) are two alternative causes of an alarm being triggered (E).

The following data frame (`dat.redun`) contains all and only the configurations that are compatible with (5):

```
R> (dat.redun <- tt2df(selectCases("(A*B + C <-> D)*(a + c <-> E)")))
```

```
   A B C D E
2  0 1 1 1 1
4  0 0 1 1 1
5  1 1 0 1 1
14 0 1 0 0 1
15 1 0 0 0 1
16 0 0 0 0 1
17 1 1 1 1 0
19 1 0 1 1 0
```

The problem now is that `dat.redun` does not only entail the two *asf* contained in (5), *viz.* (6) and (7), but also a third one, *viz.* (8):

$$A{*}B + C \leftrightarrow D \tag{6}$$

$$a + c \leftrightarrow E \tag{7}$$

$$a{*}D + e \leftrightarrow C \tag{8}$$

That means the behavior of factor C, which is exogenous in the data-generating structure (5), can be expressed as a redundancy-free Boolean function of its two effects D and E. (8), hence, amounts to an upstream (or backtracking) *asf*, which, obviously, must not be causally interpreted. Indeed, when (8) is embedded in the superordinate dependency structure (9) that results from a conjunctive concatenation of all *asf* that follow from `dat.redun`, it turns out that (8) is redundant. The reason is that (9) has a proper part which is logically equivalent to (9), namely (5).

$$(A{*}B + C \leftrightarrow D) * (a + c \leftrightarrow E) * (a{*}D + e \leftrightarrow C) \tag{9}$$

(9) and (5) state exactly the same about the behavior of the factors in `dat.redun`, meaning that (8) makes no difference to that behavior over and above (6) and (7). By contrast, neither (6) nor (7) can be eliminated from (9) such that the remaining expression is logically equivalent to (9). Both of these downstream *asf* make their own distinctive difference to the behavior of the factors in `dat.redun`. The upstream *asf* (8), however, is a *structural redundancy* in (9). It must not be causally interpreted due to a violation of (NR).

Accordingly, the `cna()` function tests all *csf* for structural redundancies. If a `csf` contains a structurally redundant `asf`, the solution attribute `redundant` is `TRUE`. To illustrate with `dat.redun`:

```
R> ana.redun <- cna(dat.redun, details = TRUE)
R> printCols <- c("condition", "consistency", "coverage", "exhaustiveness",
```

```
+    "faithfulness", "redundant")
R> csf(ana.redun)[printCols]

  condition                                          consistency coverage
1 (e + a*D <-> C)*(C + A*B <-> D)*(a + c <-> E)             1        1
  exhaustiveness faithfulness redundant
1              1            1      TRUE
```

Csf with `redundant = TRUE` must never be causally interpreted; rather, they must be further processed with the function `minimalizeCsf(x)`, whose input `x` is a solution object generated by `cna()`. `minimalizeCsf()` reduces the *csf* contained in `x` by recursively testing their component *asf* for redundancy and eliminating the redundant ones. The function outputs all redundancy-free *csf* that are logically equivalent to the *csf* in `x`, that is, it builds the *csf* from `x` that satisfy (NR) and, thus, represent well-formed causal structures.

```
R> minimalizeCsf(ana.redun)

Object of class 'minimalizeCsf' containing 1 solution(s)

=== Solution 1 ===
Condition:
(C+A*B<->D)*(a+c<->E)

  outcome con cov n.asf
1     D,E   1   1     2

redundant parts:
(e+a*D<->C)
```

The function behind the solution attribute `redundant` is also available as stand-alone function `redundant()` (see the **cna** reference manual for details).

6. Interpreting the output

The ultimate output of `cna()` and, if applicable, `minimalizeCsf()` is a set of *csf*. The causal inferences that are warranted based on the data input `x` relative to the chosen `con` and `cov` thresholds and the provided `ordering` and `maxstep` have to be read off that issued *csf* set. This section explains this final interpretative step of a CNA analysis.

There are three possible types of outputs:

1. a *csf* set with no element (and, correspondingly, no *asf*);
2. a *csf* set with exactly one element (and, correspondingly, exactly one *asf* for each endogenous factor);
3. a *csf* set with more than one element (and, correspondingly, more than one *asf* for at least one endogenous factor).

6.1. No solution

As indicated in section 3.4, a null result can have two sources: either the data are too noisy to render the chosen `con` and `cov` thresholds satisfiable or the selected `maxstep` is too low. If increasing the `maxstep` does not induce `cna()` to build solutions at the chosen `con` and `cov` thresholds, the latter should be gradually lowered. If no solutions are recovered at `con = cov = .75`, the data are too noisy to warrant reliable causal inferences. Users are then advised to go back to the data and follow standard guidelines (known from other methodological frameworks) to improve data quality, e.g. by integrating further relevant factors into the analysis, enhancing the control of unmeasured causes, expanding the population of cases or disregarding inhomogeneous cases, correcting for measurement error, supplying missing values, etc.

It must be emphasized again (see section 2.3) that, under normal circumstances, an empty *csf* set does not warrant the conclusion that the factors contained in the data input `x` are causally irrelevant to one another. The inference to causal irrelevance is much more demanding than the inference to causal relevance. While the latter inference merely requires evidence for the *existence* of at least one difference-making context, the former inference requires evidence for the *non-existence* of such a context. A null result only furnishes evidence for causal irrelevance if there are independent reasons to assume that all potentially relevant factors are measured in `x` and that `x` exhausts the space of empirically possible configurations.

6.2. A unique solution

That `cna()` (or `minimalizeCsf()`) outputs a *csf* set with exactly one element amounts to the optimal completion of a CNA analysis. It means that the data input `x` contains sufficient evidence for a determinate causal inference. The factor values on the left-hand sides of "↔" in the *asf* constituting that *csf* can be interpreted as causes of the factor values on the right-hand sides. Moreover, their conjunctive, disjunctive, and sequential groupings reflect the Boolean properties of the data-generating causal structure.

Plainly, as with any other method of causal inference, the reliability of CNA's causal conclusions essentially hinges on the quality of the processed data. If the data are free of deficiencies as confounding, measurement error, fragmentation etc., a unique solution is guaranteed to correctly reflect the data-generating structure. With increasing data deficiencies, the (inductive) risk of committing causal fallacies inevitably increases as well. For details on the degree to which the reliability of CNA's causal conclusions decreases with increasing data deficiencies see Baumgartner and Ambühl (2018).

6.3. Multiple solutions

If `cna()` (or `minimalizeCsf()`) outputs a *csf* set with more than one element, the processed data underdetermine their own causal modeling. That means the evidence contained in the data is insufficient to determine which of the issued solutions corresponds to the data-generating causal structure. An output set of multiple solutions $\{csf_1, csf_2, ..., csf_n\}$ is to be interpreted *disjunctively*: the data-generating causal structure is

$$csf_1 \text{ OR } csf_2 \text{ OR } ... \text{ OR } csf_n$$

but, based on the evidence contained in the data, it is ambiguous which disjunct is actually operative.

That empirical data underdetermine their own causal modeling is a very common phenomenon in all methodological traditions (Simon 1954; Spirtes *et al.* 2000, 59-72; Kalisch *et al.* 2012; Eberhardt 2013; Baumgartner and Thiem 2017a). But while some methods are designed to automatically generate all fitting models, e.g. Bayes nets methods and CCMs, other methods require their users to manually vary the available model building parameters in order to generate the whole space of fitting models, e.g. regression analytic methods. Whereas model ambiguities are a thoroughly investigated topic in certain traditions, e.g. Bayes nets methods, they are only beginning to be studied in the literature on CCMs, in particular on QCA. There is still an unfortunate practice of model-underreporting in QCA studies. In fact, most QCA software regularly fails to find all data-fitting models. The only currently available QCA program that recovers the whole model space by default is **QCApro** (Thiem 2018).

CNA—on a par with any other method—cannot disambiguate what is empirically underdetermined. Rather, it draws those and only those causal conclusions for which the data *de facto* contain evidence. In cases of empirical underdetermination it therefore renders transparent all data-fitting models and leaves the disambiguation up to the analyst.

That `cna()` issues multiple solutions for some data input `x` does not necessarily mean that `x` is deficient. In fact, even data that is *ideal* by all quality standards of configurational causal modeling can give rise to model ambiguities. The following simulates a case in point:

```
R> dat7 <- selectCases("a*B + A*b + B*C <-> D")
R> printCols <- c("condition", "consistency", "coverage", "inus",
+    "exhaustiveness")
R> csf(cna(dat7, details = c("exhaust", "inus")))[printCols]

  condition             consistency coverage inus exhaustiveness
1 a*B + A*b + A*C <-> D           1        1 TRUE              1
2 a*B + A*b + B*C <-> D           1        1 TRUE              1
```

`dat7` induces perfect consistency and coverage scores and is free of data fragmentation; it contains all and only the configurations that are compatible with the target structure, which accordingly is exhaustively and faithfully reflected in `dat7`. Nonetheless, two *csf* can be inferred. The causal structures expressed by these two *csf* generate the exact same configurational data, meaning they are *configurationally indistinguishable*.

Although, a unique solution is more determinate and, thus, preferable to multiple solutions, the fact that `cna()` generates multiple equally data-fitting models is not generally an uninformative result. In the above example, both resulting *csf* feature $a{*}B + A{*}b$. That is, the data contain enough evidence to establish the joint relevance of $a{*}B$ and of $A{*}b$ for D (on alternative paths). What is more, it can be conclusively inferred that D has a further complex cause, *viz.* either $A{*}C$ or $B{*}C$. It is merely an open question which of these candidate causes is actually operative.

That different model candidates have some *msc* in common is a frequent phenomenon. Here's a real-life example, where two alternative causes, *viz.* $C{=}1 + F{=}2$, are present in all solutions:

```
R> csf(mvcna(d.pban, cov = .95, maxstep = c(4,5,12)))["condition"]

  condition
1 C=1 + F=2 + C=0*F=1 + C=2*V=0 <-> PB=1
```

```
2 C=1 + F=2 + C=0*T=2 + C=2*V=0 <-> PB=1
3 C=1 + F=2 + C=2*F=0 + C=0*F=1 + F=1*V=0 <-> PB=1
4 C=1 + F=2 + C=2*F=0 + C=0*T=2 + F=1*V=0 <-> PB=1
5 C=1 + F=2 + C=0*F=1 + C=2*T=1 + T=2*V=0 <-> PB=1
6 C=1 + F=2 + C=0*F=1 + T=1*V=0 + T=2*V=0 <-> PB=1
7 C=1 + F=2 + C=0*T=2 + C=2*T=1 + T=2*V=0 <-> PB=1
8 C=1 + F=2 + C=0*T=2 + T=1*V=0 + T=2*V=0 <-> PB=1
```

Such commonalities can be reported as conclusive results.

Moreover, even though multiple solutions do not permit pinpointing the causal structure behind an outcome, they nonetheless allow for constraining the range of possibilities. In a context where the causes of some outcome are unknown it amounts to a significant gain of scientific insight when a study can show that the structure behind that outcome has one of a small number of possible forms, even if it cannot determine which one exactly.

However, the larger the amount of data-fitting solutions and the lower the amount of commonalities among them, the lower the overall informativeness of a `cna()` output. Indeed, the ambiguity ratio in configurational causal modeling can reach dimensions where nothing at all can be concluded about the data-generating structure any more. Hence, a highly ambiguous result is on a par with a null result. A telling example of this sort is `d.volatile` which was discussed in section 3.4 above (cf. also Baumgartner and Thiem 2017a).

As the problem of model ambiguities is still underinvestigated in the CCM literature, there do not yet exit explicit guidelines for how to proceed in cases of ambiguities. The model fit scores and solution attributes reported in the output of `cna()` often provide some leverage to narrow down the space of model candidates. For instance, if, in a particular discovery context, there is reason to assume that data have been exhaustively collected, to the effect that all configurations that are compatible with an investigated causal structure are contained in the data, the model space may be restricted to *csf* with a high score on exhaustiveness. By way of example, for `d.pacts` a total of 240 *csf* are built at `con = cov = .8`:

```
R> ana.pact.1 <- fscna(d.pacts, ordering = list("PACT"), con = .8, cov = .8,
+    maxstep = c(4,4,12), details = TRUE)
R> csf.pact.1 <- csf(ana.pact.1, Inf)
R> length(csf.pact.1$condition)

[1] 240
```

If only *csf* with `exhaustiveness == 1` are considered, the amount of candidate *csf* is divided in half:

```
R> csf.pact.1.ex <- subset(csf.pact.1, exhaustiveness==1)
R> length(csf.pact.1.ex$condition)

[1] 115
```

To further reduce the model space, coherence may be brought to bear as well. The higher the coherence of a *csf*, the higher its overall model fit. In the above example, if a coherence score of at least 0.85 is imposed, the 115 candidate *csf* can be reduced to 2:

```
R> csf.pact.1.ex.co <- subset(csf.pact.1.ex, coherence >= 0.85)
R> length(csf.pact.1.ex.co$condition)

[1] 2
```

If the whole analysis is moreover run with a restriction to INUS solutions, the initial ambiguity with 240 *csf* can be resolved completely, with only one *csf* satisfying all of `exhaustiveness == 1`, `coherence >= 0.85`, and `inus = TRUE`:

```
R> ana.pact.2 <- fscna(d.pacts, ordering = list("PACT"), con = .8, cov = .8,
+    maxstep = c(4,4,12), details = TRUE, inus.only = TRUE)
R> csf.pact.2 <- csf(ana.pact.2, Inf)
R> csf.pact.2.inus.ex.co <- subset(csf.pact.2,
+    exhaustiveness==1 & coherence >= 0.85)
R> csf.pact.2.inus.ex.co

  outcome condition                      consistency coverage
2 E,PACT  (w + p <-> E)*(E + P <-> PACT)       0.844    0.835
  complexity inus exhaustiveness faithfulness coherence redundant
2          4 TRUE              1        0.615     0.854     FALSE
```

Clearly though, the fit parameters and solution attributes provided by `cna()` will not always provide a basis for ambiguity reduction. The evidence contained in analyzed data may simply be insufficient to draw determinate causal conclusions. Maybe background theories or case knowledge can be brought to bear to select among the model candidates (see section 6.4). Nevertheless, the most important course of action in the face of ambiguities is to *render them transparent*. By default, readers of CCM publications should be presented with all data-fitting models and if space constraints do not permit so, readers must at least be informed about the degree of ambiguity. Full transparency with respect to model ambiguities, first, allows readers to determine for themselves how much confidence to have in the conclusions drawn in a study, and second, paves the way for follow-up studies that are purposefully designed to resolve previously encountered ambiguities.

6.4. "Back to the cases"

When CCMs are applied to small- or intermediate-N data, researchers may be familiar with some or all of the cases in their data. For instance, they may know that in a particular case certain causes of an outcome are operative while others are not. Or they may know why certain cases are outliers or why others feature an outcome but none of the potential causes. A proper interpretation of the `cna()` output may therefore require that the performance of the obtained models be assessed on the case level and against the background of the available case knowledge.

The function that facilitates the evaluation of recovered *msc*, *asf*, and *csf* on the case level is `condition(x, tt, type)`. Its first input is a character vector `x` specifying Boolean expressions—typically *asf* or *csf*—and its second input a truth table `tt`. In case of *cs* or *mv* data, the output of `condition()` then highlights in which cases `x` is instantiated, whereas for

fs data, the output lists relevant membership scores in exogenous and endogenous factors. Moreover, if `x` is an *asf* or *csf*, `condition()` issues their consistency and coverage scores.

Instead of a truth table, it is also possible to give `condition()` a data frame as second input. In this case, the data type must be specified using the `type` argument. To abbreviate the specification of the data type, the functions `cscond(x, tt)`, `mvcond(x, tt)`, and `fscond(x, tt)` are available as shorthands.

To illustrate, we re-analyse `d.autonomy`:

```
R> dat.aut.2 <- d.autonomy[15:30, c("AU","EM","SP","CO","RE","DE")]
R> ana.aut.3 <- fscna(dat.aut.2, con = .91, cov = .91,
+    ordering = list(c("RE", "DE","SP","CO"),"EM","AU"),
+    strict = TRUE)
R> fscond(csf(ana.aut.3)$condition, dat.aut.2)
```

That function call returns a list of three tables, each corresponding to one of the three *csf* contained in `ana.aut.3` and breaking down the relevant *csf* to the case level by contrasting the membership scores in the left-hand and right-hand sides of the component *asf*. A case with a higher left-hand score is one that pulls down consistency, whereas a case with a higher right-hand score pulls down coverage. For each *csf*, `condition()` moreover returns overall consistency and coverage scores as well as consistency and coverage scores for the component *asf*.

The three *csf* in `ana.aut.3` differ only in regard to their component *asf* for outcome *AU*. The function `group.by.outcome(condlst)`, which takes an output object `condlst` of `condition()` as input, lets us more specifically compare these different *asf* with respect to how they fare on the case level.

```
R> group.by.outcome(fscond(asf(ana.aut.3)$condition, dat.aut.2))$AU
```

```
          SP EM*RE+re*DE EM*RE+CO*DE  AU | n.obs
ENacg1   1.0         1.0         1.0 1.0 |     1
ENacg2   0.6         0.4         0.4 0.4 |     1
ENacg3   0.8         0.6         0.9 0.8 |     1
ENacg4   0.6         1.0         1.0 1.0 |     1
ENacg5   0.4         0.4         0.4 0.4 |     1
ENacg6   0.6         0.7         0.7 0.6 |     1
ENacg7   1.0         0.8         0.8 1.0 |     1
ENacg8   1.0         1.0         1.0 1.0 |     1
ENacto1  0.4         0.4         0.6 0.4 |     1
ENacto2  0.4         0.4         0.4 0.4 |     1
ENacosa1 0.4         0.4         0.2 0.4 |     1
ENacosa2 0.4         0.4         0.4 0.2 |     1
ENacosa3 0.4         0.4         0.4 0.6 |     1
ENacat1  0.4         0.4         0.4 0.2 |     1
ENacat2  0.4         0.4         0.4 0.6 |     1
ENacat3  0.4         0.6         0.4 0.4 |     1
```

The first three columns of that table list the membership scores of each case in the left-hand sides of the *asf*, and the fourth column reports the membership scores in *AU*. The

table shows that the first *asf* ($SP \leftrightarrow AU$) outperforms the other *asf* in cases ENacg3/6/7, ENacto1, ENacosa1, and ENacat3, while it is outperformed by another *asf* in cases ENacg2 and ENacg4. In all other cases, the three solution candidates fare equally. If prior knowledge is available about some of these cases, this information can help to choose among the candidates. For instance, if it is known that in case ENacg7, no other relevant factors are operative than the ones contained in `dat.aut.2`, it follows that ENacg7's full membership in AU must be brought about by SP—which, in turn, disqualifies the other solutions. By contrast, if the absence of other relevant factors can be assumed for case ENacg4, the *asf* featuring SP as cause of AU is disqualified.

7. Benchmarking

Benchmarking the reliability of a method of causal inference is an essential element of method development and validation. In a nutshell, it amounts to testing to what degree the benchmarked method recovers the true data-generating structure Δ or proper substructures of Δ from data of varying quality. As Δ is not normally known in real-life discovery contexts, the reliability of a method cannot be assessed by applying it to real-life data. Instead, reliability benchmarking is done in so-called *inverse searches*, which reverse the order of causal discovery as it is commonly conducted in scientific practice. An inverse search comprises three steps:

(1) a data-generating causal structure Δ is presupposed/drawn (as ground truth),

(2) artificial data δ is simulated from Δ, possibly featuring various deficiencies (e.g. noise, fragmentation, measurement error etc.),

(3) δ is processed by the tested method in order to check whether its output meets the tested reliability benchmark (e.g. whether the output is true of or identical to Δ).

A benchmark test can measure various properties of a method's output, for instance, whether it is correct or complete or to what degree it is informative or ambiguous, etc. As real-life configurational data tend to be fragmented, meaning that only a proper subset of all empirically possible configurations are actually contained in the data, configurational comparative methods like CNA typically do not infer the complete Δ from a real-life δ but only proper substructures thereof (see section 2.3). Accordingly, the core reliability benchmark for CNA should not be completeness but *correctness*.

Spelling out what exactly correctness amounts to requires two preliminary qualifications. First, CNA cannot be expected to draw a causal inference from any δ. For instance, if δ is too noisy, consistency or coverage thresholds cannot be met, which will, in turn, induce CNA to abstain from issuing any models (see section 3.2). Abstaining from drawing a causal inference from data of insufficient quality is not a defective but a correct response. Hence, an empty solution set is to be counted as correct. Second, as indicated in section 6.3, whenever there exist multiple models that fit δ equally well, CNA will output all of them. The set of issued models $\mathbf{csf} = \{csf_1, csf_2, ..., csf_n\}$ is to be interpreted in terms of a disjunction—which, as a whole, is true of Δ iff at least one disjunct $csf_i \in \mathbf{csf}$ is true of Δ. In sum, CNA's output is *correct* iff, whenever CNA infers a set **csf** from δ, at least one $csf_i \in \mathbf{csf}$ is true of Δ (Baumgartner and Thiem 2017b). In other words, CNA's output is correct iff it does not commit a causal fallacy.

The **cna** package provides all the functionalities necessary to conduct inverse searches that are tailor-made to benchmark the correctness of the output of the `cna()` function. The functions `randomAsf()` and `randomCsf()` can be used to draw a data-generating structure Δ in step (1). `randomAsf(x)` generates a structure with a single outcome (i.e. a random *asf*) and `randomCsf(x)` an acyclic multi-outcome structure (i.e. a random *csf*), where `x` is a data frame or `truthTab` defining the factors and their possible values from which the structures are drawn. The function `selectCases()`, which has already been discussed in section 3.1.2, can be employed to simulate data δ in the course of step (2). Finally, `is.submodel()` is serviceable to determining whether the output inferred from δ by `cna()` is true of Δ.

`is.submodel(x, y)` takes a character vector of *asf* or *csf* as first input and tests whether the elements of that vector are submodels of `y`, which, in turn, is a character string of length 1 representing the target *asf* or *csf* (i.e. Δ). `x` being a submodel of `y` means that all the causal claims entailed by `x` are also entailed by `y`, which is the case if a causal interpretation of `x` entails conjunctive and disjunctive causal relevance relations that are all likewise entailed by a causal interpretation of `y`. More specifically, `x` is a *submodel* of `y` iff the following conditions are satisfied (cf. Baumgartner and Thiem 2017b or Baumgartner and Ambühl 2018, online appendix):

(i) all factor values causally relevant according to `x` are also causally relevant according to `y`,

(ii) all factor values contained in two different disjuncts in `x` are also contained in two different disjuncts in `y`,

(iii) all factor values contained in the same conjunct in `x` are also contained in the same conjunct in `y`,

(iv) if `x` is a *csf* with more than one *asf*, (i) to (iii) are satisfied for all *asf* in `x`.

Against that background, the following might be a core of a correctness benchmark test that simulates multi-value data with 20% missing observations and 5% random noise, and that runs `cna()` at the lower consistency and coverage bound of 0.75 without giving the algorithm any prior causal information in an ordering.

```
R> # Draw the ground truth.
R> fullData <- mvtt(allCombs(c(4,4,4,4,4)))
R> groundTruth <- randomCsf(fullData, n.asf = 2, compl = 2:3)
R> # Generate the complete data.
R> x <- tt2df(selectCases(groundTruth, fullData))
R> # Introduce fragmentation.
R> x <- x[-sample(1:nrow(x), nrow(x)*0.2), ]
R> # Introduce random noise.
R> x <- rbind(tt2df(fullData[sample(1:nrow(fullData), nrow(x)*0.05), ]), x)
R> # Run CNA without an ordering.
R> csfs <- csf(mvcna(x, con = .75, cov = .75, maxstep = c(3, 3, 12)))
R> min.csfs <- if(nrow(csfs)>0) {
+                as.vector(minimalizeCsf(csfs$condition, mvtt(x))$condition)
+              } else {NA}
R> # Check whether no causal fallacy (no false positive) is returned.
```

```
R> if(length(min.csfs)==1 && is.na(min.csfs)) {
+        TRUE } else {any(is.submodel(min.csfs, groundTruth))}
```

Every re-run of this code chunk generates a different ground truth and different data; in some runs CNA passes the test, in others it does not. To determine CNA's correctness ratio under these test conditions, the above core must be embedded in a suitable test loop. To estimate CNA's overall correctness ratio, the test conditions should be systematically varied by, for instance, varying the complexity of the ground truth, the degree of fragmentation and noise, the consistency and coverage thresholds, or by supplying CNA with more or less prior causal information via an ordering. For single-outcome structures (*asf*), benchmark tests with some of this variation have been conducted in Baumgartner and Ambühl (2018).

8. Summary

This vignette introduced the theoretical foundations as well as the main functions of the **cna** R package for configurational causal inference and modeling with Coincidence Analysis (CNA). Moreover, we explained how to interpret the output of CNA, provided some guidance for how to use various model fit parameters for the purpose of ambiguity reduction, and supplied a template for correctness benchmarking.

CNA is currently the only CCM that builds multi-outcome models and, hence, uncovers all Boolean dimensions of causality: conjunctivity, disjunctivity, and sequentiality. Moreover, it builds causal models on the basis of a bottom-up algorithm that is unique among CCMs and gives CNA an edge over CCMs building models from the top down not only with respect to multi- but also single-outcome structures. Overall, CNA constitutes a powerful methodological alternative for researchers interested in the Boolean dimensions of causality, and the **cna** package makes that inferential power available to end-users.

Acknowledgments

We are grateful to Alrik Thiem for helpful comments on an earlier draft of this vignette, and we thank the Toppforsk-program of the Bergen Research Foundation and the University of Bergen (grant nr. 811886) and the Swiss National Science Foundation (grant nr. PP00P1_144736/1) for generous support of this research.

References

Albert DZ (1992). *Quantum Mechanics and Experience*. Harvard University Press, Cambridge.

Baumgartner M (2008). "Regularity theories reassessed." *Philosophia*, **36**, 327–354.

Baumgartner M (2009a). "Inferring causal complexity." *Sociological Methods & Research*, **38**, 71–101.

Baumgartner M (2009b). "Uncovering deterministic causal structures: A Boolean approach." *Synthese*, **170**, 71–96.

Baumgartner M (2013). "A regularity theoretic approach to actual causation." *Erkenntnis*, **78**, 85–109.

Baumgartner M (2015). "Parsimony and causality." *Quality & Quantity*, **49**, 839–856.

Baumgartner M, Ambühl M (2018). "Causal modeling with multi-value and fuzzy-set Coincidence Analysis." *Political Science Research and Methods*. Doi: 10.1017/psrm.2018.45.

Baumgartner M, Falk C (2018). "Boolean difference-making: A modern regularity theory of causation." PhilSci Archive. URL http://philsci-archive.pitt.edu/id/eprint/14876.

Baumgartner M, Thiem A (2017a). "Model ambiguities in configurational comparative research." *Sociological Methods & Research*, **46**(4), 954–987.

Baumgartner M, Thiem A (2017b). "Often trusted but never (properly) tested: Evaluating Qualitative Comparative Analysis." *Sociological Methods & Research*. Doi: 10.1177/0049124117701487.

Cronqvist L (2011). *Tosmana: Tool for Small-N Analysis, version 1.3.2.0 [computer program]*. University of Trier, Trier.

Cronqvist L, Berg-Schlosser D (2009). "Multi-value QCA (mvQCA)." In B Rihoux, CC Ragin (eds.), *Configurational Comparative Methods: Qualitative Comparative Analysis (QCA) and Related Techniques*, pp. 69–86. Sage Publications, London.

Downing BM (1992). *The Military Revolution and Political Change: Origins of Democracy and Autocracy in Early Modern Europe*. Princeton University Press, N.J.

Duşa A (2007). "User manual for the **QCA**(GUI) package in R." *Journal of Business Research*, **60**(5), 576–586.

Duşa A, Thiem A (2015). "Enhancing the minimization of Boolean and multivalue output functions with *e*QMC." *Journal of Mathematical Sociology*, **39**(2), 92–108.

Eberhardt F (2013). "Experimental indistinguishability of causal structures." *Philosophy of Science*, **80**(5), 684–696.

Gelman A, Hill J (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, Cambridge.

Goertz G (2006). *Social Science Concepts: A User's Guide*. Princeton University Press, Princeton.

Graßhoff G, May M (2001). "Causal regularities." In W Spohn, M Ledwig, M Esfeld (eds.), *Current Issues in Causation*, pp. 85–114. Mentis, Paderborn.

Hájek P (1998). *Metamatematics of Fuzzy Logic*. Kluwer, Dordrecht.

Kalisch M, Maechler M, Colombo D, Maathuis MH, Buehlmann P (2012). "Causal inference using graphical models with the R package **pcalg**." *Journal of Statistical Software*, **47**(11), 1–26.

Longest KC, Vaisey S (2008). "**fuzzy**: A program for performing Qualitative Comparative Analyses (QCA) in Stata." *Stata Journal*, **8**(1), 79–104.

Mackie JL (1974). *The Cement of the Universe: A Study of Causation*. Clarendon Press, Oxford.

McCluskey EJ (1965). *Introduction to the Theory of Switching Circuits*. Princeton University Press, Princeton.

Quine WvO (1959). "On cores and prime implicants of truth functions." *The American Mathematical Monthly*, **66**(9), 755–760.

Ragin CC (1987). *The Comparative Method*. University of California Press, Berkeley.

Ragin CC (2006). "Set relations in social research: Evaluating their consistency and coverage." *Political Analysis*, **14**(3), 291–310.

Ragin CC (2008). *Redesigning Social Inquiry: Fuzzy Sets and Beyond*. University of Chicago Press, Chicago.

Ragin CC (2009). "Qualitative Comparative Analysis using fuzzy sets (fsQCA)." In B Rihoux, CC Ragin (eds.), *Configurational Comparative Methods: Qualitative Comparative Analysis (QCA) and Related Techniques*, pp. 87–121. Sage, Thousand Oaks.

Ragin CC (2014). "Comment: Lucas and Szatrowski in critical perspective." *Sociological Methodology*, **44**(1), 80–94.

Reichert C, Rubinson C (2014). ***Kirq***. version 2.1.12, Houston: University of Houston-Downtown.

Simon HA (1954). "Spurious correlation: A causal interpretation." *Journal of the American Statistical Association*, **49**(267), 467–479.

Spirtes P, Glymour C, Scheines R (2000). *Causation, Prediction, and Search*. 2 edition. MIT Press, Cambridge.

Suppes P (1970). *A Probabilistic Theory of Causality*. North Holland, Amsterdam.

Thiem A (2014). "Unifying configurational comparative methods: Generalized-set Qualitative Comparative Analysis." *Sociological Methods & Research*, **43**(2), 313–337.

Thiem A (2015). "Using Qualitative Comparative Analysis for identifying causal chains in configurational data: A methodological commentary on Baumgartner and Epple." *Sociological Methods & Research*, **44**(4), 723–736.

Thiem A (2018). ***QCApro***: *Advanced functionality for performing and evaluating Qualitative Comparative Analysis*. R Package Version 1.1-2. URL http://www.alrik-thiem.net/software/.

Thiem A, Baumgartner M (2016). "Modeling causal irrelevance in evaluations of configurational comparative methods." *Sociological Methodology*, **46**, 345–357. Doi: 10.1177/0081175016654736.

Thiem A, Baumgartner M, Bol D (2016). "Still lost in translation! A correction of three misunderstandings between configurational comparativists and regressional analysts." *Comparative Political Studies*, **49**, 742–774.

Affiliation:

Michael Baumgartner
University of Bergen
Department of Philosophy
Postboks 7805
5020 Bergen
Norway
E-mail: michael.baumgartner@uib.no
URL: http://people.uib.no/mba110/

Mathias Ambühl
Consult AG Statistical Services
Tramstrasse 10
8050 Zürich
E-mail: mathias.ambuehl@consultag.ch