How To ... Use Software for Set-Theoretic Analysis: Online Appendix to Carsten Q. Schneider and Claudius Wagemann (2012) "Set-Theoretic Methods for the Social Sciences", Cambridge University Press

Mario Quaranta[1]
Carsten Q. Schneider[2]

Version 1.2

July 2013

[1]LUISS "Guido Carli", Roma (Italy). E-mail: mquaranta@luiss.it
[2]Central European University, Budapest (Hungary). E-mail: schneiderc@ceu.hu

Contents

List of Figures

List of Tables

Introductory Remarks

This is the online appendix to Carsten Q. Schneider's and Claudius Wagemann's (2012) book Set-Theoretic Methods for the Social Sciences, Cambridge University Press (henceforth SMSS). It provides practical guidelines on how to employ the major currently available software packages in order to perform analyses that are crucial in set-theoretically based empirical social science research. Whenever appropriate, we will discuss in parallel the following four packages: fsQCA 2.5 (Ragin, Drass, and Davey, 2006); Tosmana 1.3.2 (Cronqvist, 2006); Stata, ado file `fuzzy` (Longest and Vaisey, 2008); and R, packages `QCA 1.0.3` (Thiem and Dusa, 2012a, 2012b) and `QCA3 0.0.5` (Huang, 2009). The former two programs are specifically designed for set-theoretic analyses, whereas the latter two are off-the-shelf general statistical packages that are adapted for set-theoretic analyses.

We proceed in a chapter-by-chapter fashion, always presenting the software tricks of the trade in performing those analyses that are discussed in the respective chapter of the book. Throughout this online appendix, we will use data from published set-theoretic analysis but the aim is, of course, not to contribute to the substantive debates addressed in these publications, nor to replicate the analyses. Rather, we want to show how software-based set-theoretic analyses look like in an applied setting.

The data files that are needed for replicating our analyses are also made available at `www.cambridge.org/schneider-wagemann`. We save all data in a comma-delimited `csv` format, which can be read by all the software packages that we discuss. For the non-syntax based packages, we denote with square brackets ([...]) the commands that need to be performed by pointing and clicking at them. When useful, we also provide screenshots. All the commands and the outputs, in particular for Stata and R, are show in a box.

For R we use two conventions. The R code is preceded by > in the box, while the output is not preceded by anything. When a line of code is loo long an we break the line the code is preceded by +. When three dots are in the box below a command, this means that we omit the output. The commented text

is preceded by #, which means that the software would not read the text that comes afterwords.

Below is an example:

```
# This is an example
> vec <- c(2, 3, 5)
> vec

[1] 2 3 5

> vec
.
.
.

> vec <- rbind(c(3, 4, 6),
+      c(5, 7, 8))
```

Furthermore, in this appendix we present a new R package that is companion to the book, called `SetMethods` by Mario Quaranta, which contains new functions and the data necessary to replicate the analyses shown here.

Each of the following chapters is structured as follows. We first list the analytic steps that are dealt with in the respective chapter and mention the data files that are used. Then we perform the analyses with all those software packages that are applicable to the task at hand. In each chapter, we start with fsQCA, then Tosmana, Stata, and R. Often times, a specific software cannot be used for the given task, such as, for instance, Tosmana for any of the analyses that involve fuzzy sets. Sometimes, a specific package could be used for a given task, but only after some cumbersome and essentially impractical tweaks. In both instances, we skip the discussion of those packages. Our R instructions are written under the assumption that users have a basic knowledge of R. Last but not least, the development of software is an ongoing – and currently even accelerating – process, and so will be the development of these Software How-To instructions. We will update, extend, and improve these instructions on a regular basis.

Part I

Set-Theoretic Methods: The Basics

Chapter 1

Sets, Set Membership, and Calibration: How to . . . Calibrate Sets

Overview:

- Calibration of sets
  - Crisp sets, fuzzy sets, multi-value sets
- Calibration of fuzzy sets
  - Direct and indirect method of calibration
  - Other functional forms for calibration
- Empirical examples
  - Emmenegger (2011): used in book
  - Freitag and Schlicht (2009): used in book
  - Vis (2009): used in book

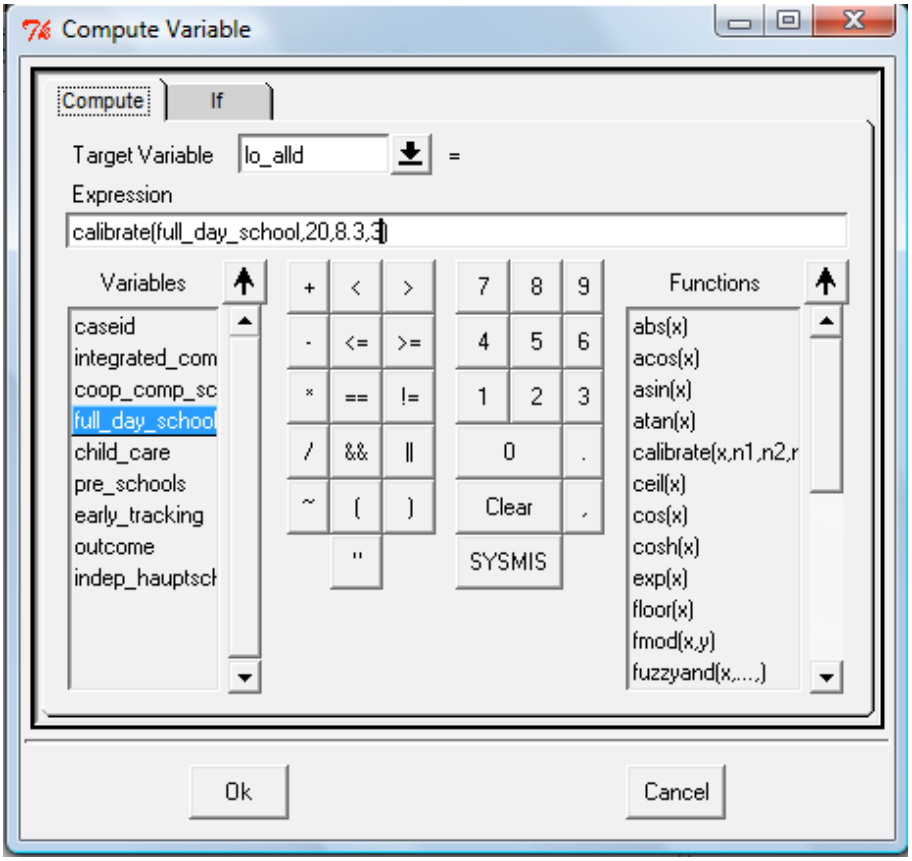
1.1 fsQCA 2.5

If data already exists in a suitable format for fsQCA 2.5, then it can be opened, clicking [file] [open] [data]. If, instead, a new data file has to be created, go to [file] [new] and then follow the self-explanatory steps. In order to get a grasp on the distribution of the data, click on [analyze] [statistics] and then either on [descriptives] or [frequencies]. A graphical overview is obtained by clicking [graphs] [fuzzy] [histogram], inserting the name of the condition or outcome for "x axis" and clicking on [plot].

fsQCA 2.5 supports the direct method of calibrating fuzzy sets. Let us take the example by Freitag and Schlicht (2009). The set to be calibrated is called "Län-

der with underdeveloped all–day school system", abbreviated lo_alld. The variable containing the empirical information is "percentage of all-day schools" (full_day_school). To perform the direct method of calibration, click [variables] [compute]. In "target variable" insert the name of the set you want to create, in our example "lo_alld". As a general advice, choose set labels in a way that it is clear what high set membership means. In the present example, the prefix "lo" makes clear that high membership scores denote those cases with few all-day schools. In the field [expression] type in [calibrate (x,n1,n2,n3)] or select it from the list of [Functions]. X stands for the name of the variable that contains the raw data, i.e. full_day_school, n1 for the 1-anchor, n2 for the 0.5-anchor, and n3 for the 0-anchor. Select the variable full_day_school and choose the following three anchors: 20 for 1-anchors; 8.3 for 0.5 anchor; 3 for 0-anchor. Click [ok]. A new column called "lo_alld" is added as the last column in the data sheet, containing the membership scores in the newly created fuzzy set. See figure 9.1.

Figure 1.1: Calibration using fsQCA 2.5.



The indirect method of calibration cannot be performed with fsQCA 2.5 (see R and Stata for this).

1.2 Tosmana 1.3.2

In Tosmana, raw data can be turned into crisp sets or multi-value sets using graphical representations of the data. To load the data by Freitag and Schlicht (2009), stored in a csv file format, go to [File] [Import] [Excel or fsqca (csv file)]. Select the file Freitag_Schlicht_2009.csv. Right below the variable names of the columns, we see both a tree icon and an exclamation mark. By clicking on the tree icon and choosing [edit variable] [setter], we see the distribution of the values and their median, as well as a proposal for a threshold, derived from an underlying cluster analysis. As should have become clear from our description of good set calibration in chapter 1.2 of SMSS, this proposed threshold for assigning cases to the membership values 0 and 1 should not simply be used by default. Nevertheless, the graphical representation provides a quick and sometimes useful overview of the raw data at hand. If we click on the exclamation mark and then [edit variable], we can also fill in our own theory-based threshold in the field "Thresholds." In order to see where this threshold is placed in the distribution of the data, we click [setter].

1.3 Stata

In Stata, both the direct and the indirect method of calibration and a dichotomization for crisp sets can be performed with the command `setgen`. The command line:

```
setgen CS = crisp(X), cutpt(n)
```

produces a crisp set of name CS based on the raw data variable X, with the threshold being at value n. The command line

```
setgen FS = drect(X), anchors(n1 n2 n3)
```

produces a fuzzy set of name FS using the direct method of calibration based on variable X and the three qualitative anchors n1, which stands for the 0-anchor, n2 (the 0.5 anchor), and n3 (the 1-anchor). Notice that the order in which the anchors are specified in Stata is from low to high, whereas in fsQCA 2.5 it is from high to low. Finally, the command line

```
setgen FS = ndrect(X), grpdvar(FSPRELIM)
```

creates a fuzzy set named FS based on the raw data variable X and the preliminary classification of cases into fuzzy set membership contained in FSPRELIM. The command line for reproducing the same fuzzy set "lo_alld" as described above for the fsQCA software, looks as follows:

```
setgen lo_alld = drect(full_day_school), anchors(3, 8.3, 20)
```

1.4 R

The first thing to do if calibration has to be done using R is to load three packages: `QCA`; `QCA3` and `SetMethods`. They contain the functions to calibrate the values and the data we will use. The code to load the packages is the following:

```
> library(QCA); library(QCA3); library(SetMethods)
```

It is useful to set your working directory:

```
> setwd("~/")
```

We use the data from Freitag and Schlicht (2010) that is called "FRSC" and from the crisp set data from Vis (2009) that is called "VisCS":

```
> data(FRSC)
> data(VisCS)
```

1.4.1 Data visualization

Some functions are useful to explore the data. `head()` shows the first lines of the dataset. `summary()` gives some descriptive statistics. `names()` gives the names of the columns (which are the conditions) and `rownames()` the names of the rows (which are the cases). So:

```
> head(FRSC)
.
.
.
> summary(FRSC)
.
.
.
> names(FRSC)
.
.
.
> rownames(FRSC)
.
```

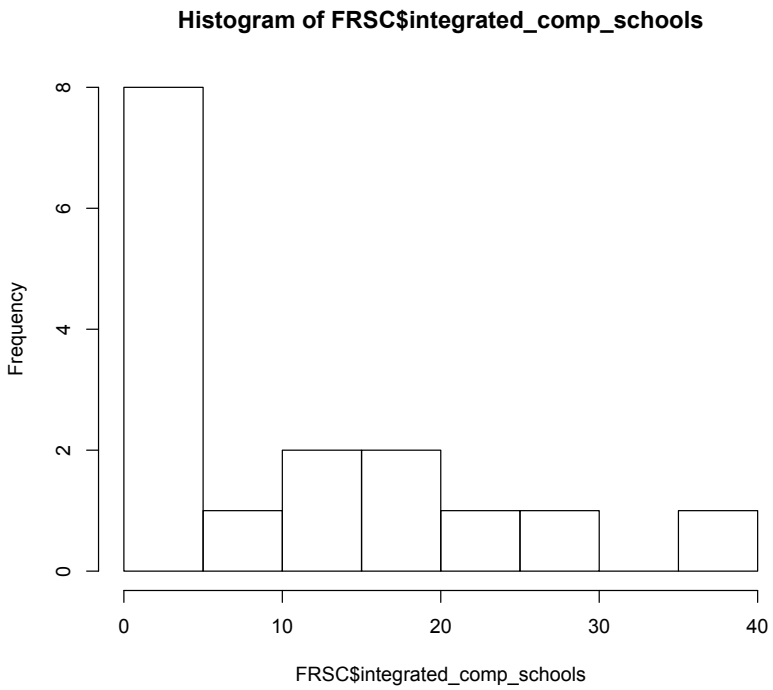
```
15 .
16 .
```

We can visualize the data using some graphics, such as histograms, with the following code:

```
1 > hist(FRSC$integrated_comp_schools, breaks = 6)
```

which gives the following figure:

Figure 1.2: Histogram of `integrated_comp_schools`

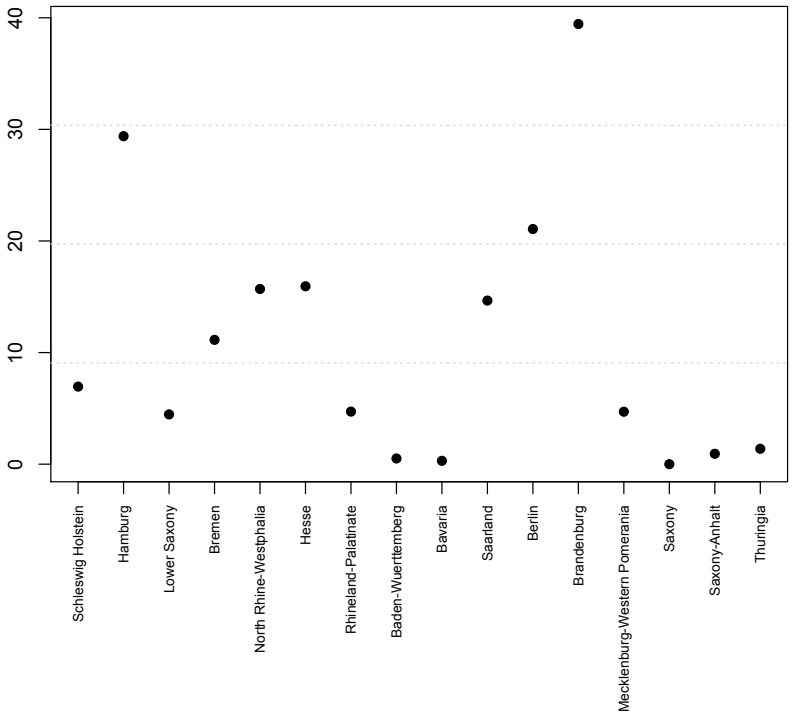


However, the use of histograms with a low N is questionable. Therefore, we can use a custom plot. Here is the code:

```
> par(mar = c(14, 3, 1, 1))  # this sets the graphical parameters
> plot(1:length(rownames(FRSC)), FRSC$integrated_comp_schools,
+ xaxt = "n", xlab = "", pch = 19)
> grid(ny = 4, nx = NA)
> axis(1, at = 1:length(rownames(FRSC)), labels = rownames(FRSC),
+ las = 2, cex.axis = .7)
```

which gives the following figure:

Figure 1.3: Plot of `integrated_comp_schools`

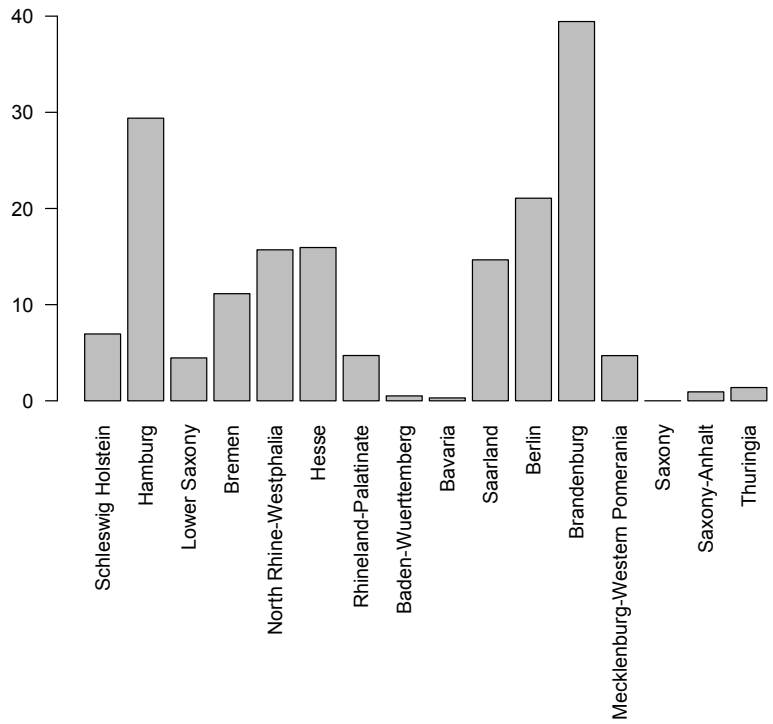


We can also use a barchart:

```
> par(mar = c(14, 3, 1, 1), las = 2)
> barplot(FRSC$integrated_comp_schools,
+   names.arg = rownames(FRSC), ylim = c(0, 40))
```

The code produces the following figure:

Figure 1.4: Barchart of `integrated_comp_schools`

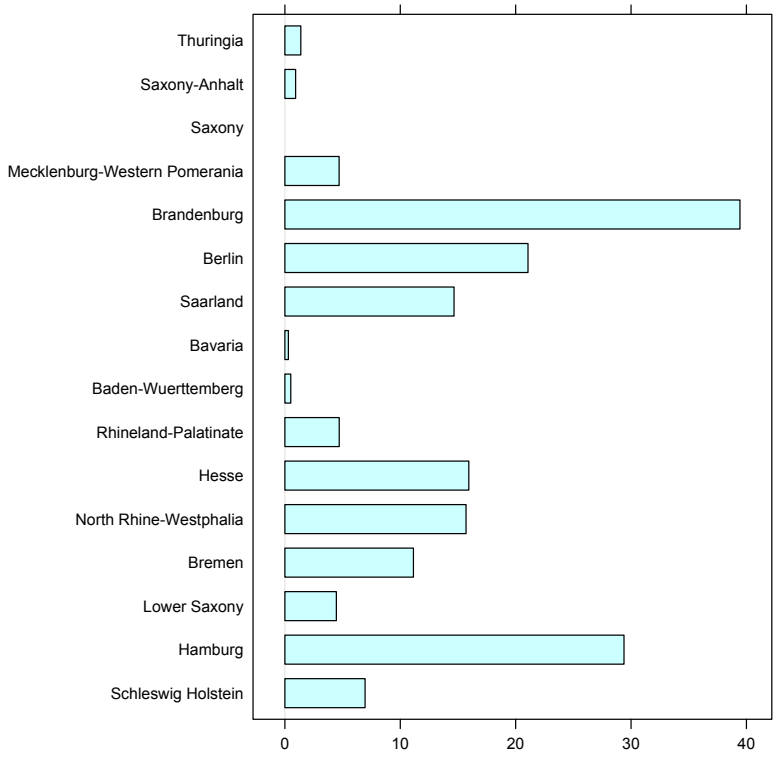


Using the `lattice` package we can produce a bar chart. The code:

```
> library(lattice)
> DTP <- matrix(FRSC$integrated_comp_schools,
+     dimnames = list(rownames(FRSC),
+       Integrated Comp Schools"))
> barchart(DTP, xlab = "")
```

gives the following figure:

Figure 1.5: Lattice barchart of `integrated_comp_schools`

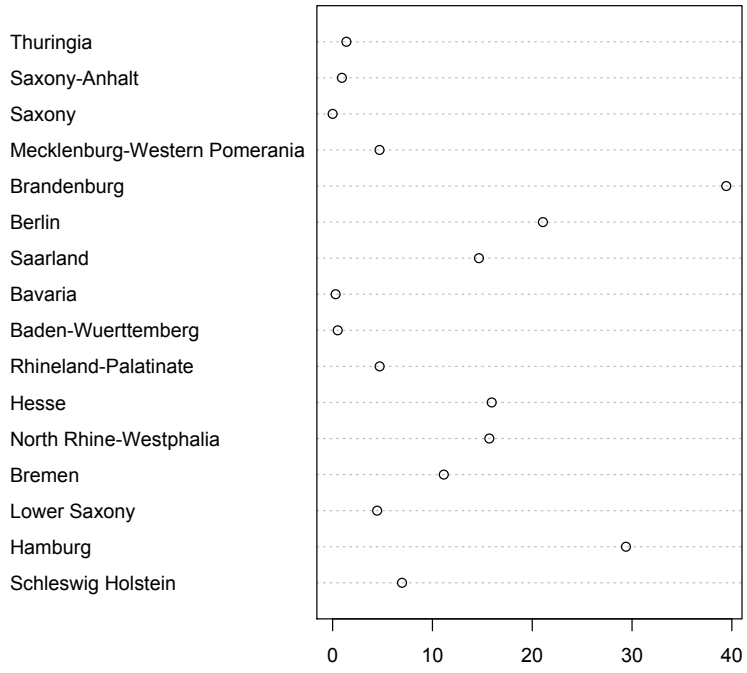


We can also use a dotchart. The code:

```
1 > dotchart(FRSC$integrated_comp_schools, labels = rownames(FRSC))
```

gives the following figure:

Figure 1.6: Dotchart of `integrated_comp_schools`

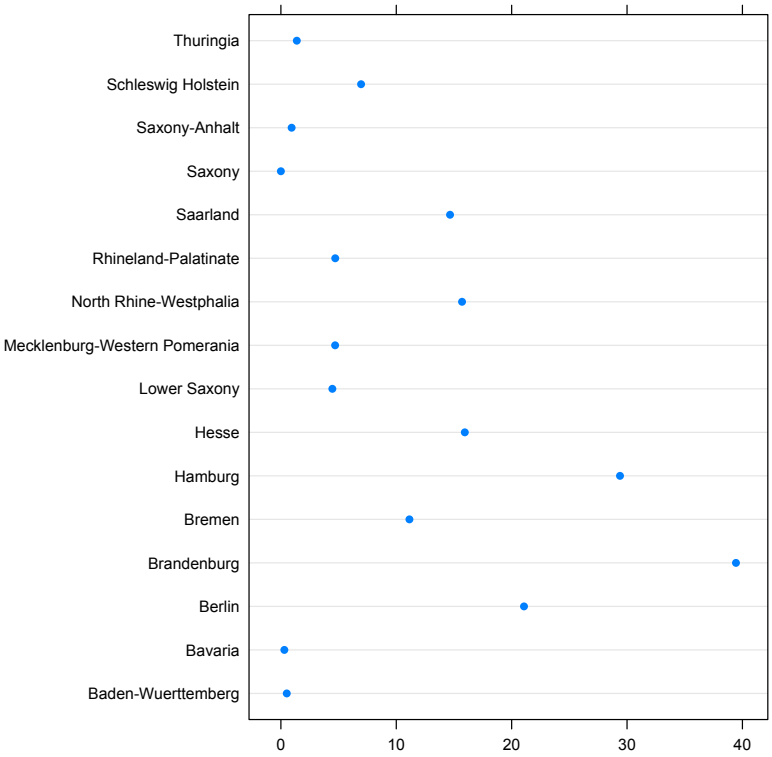


Or a dotplot. The code:

```
> dotplot(rownames(FRSC) ~ FRSC$integrated_comp_schools,
+ xlab = "")
```

gives the following figure:

Figure 1.7: Dotplot of `integrated_comp_schools`

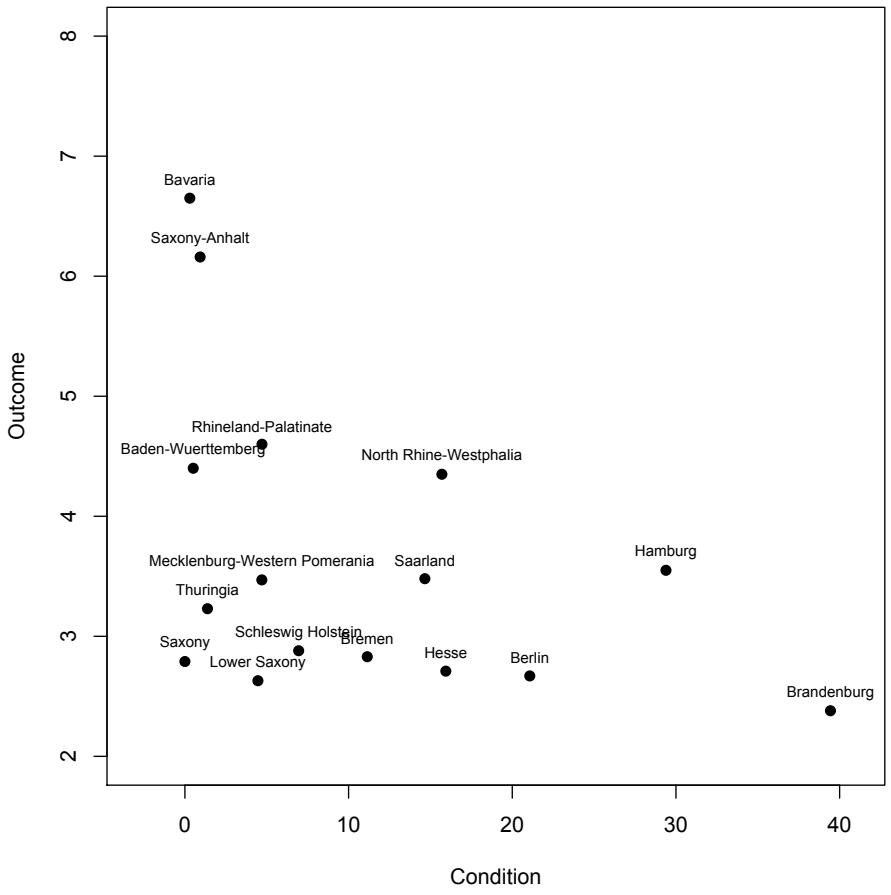


We can use scatterplots to check the association between two variables. The code:

```
> par(mar = c(4, 4, 1, 1))
> plot(FRSC$integrated_comp_schools, FRSC$outcome,
+ xlim = c(-3, 41), ylim = c(2, 8), pch = 19,
+ ylab = "Outcome", xlab = "Condition")
> text(FRSC$integrated_comp_schools, FRSC$outcome + .15,
+ labels=rownames(FRSC), cex = .7)
```

gives the following figure:

Figure 1.8: Scatterplot of `integrated_comp_schools` and outcome



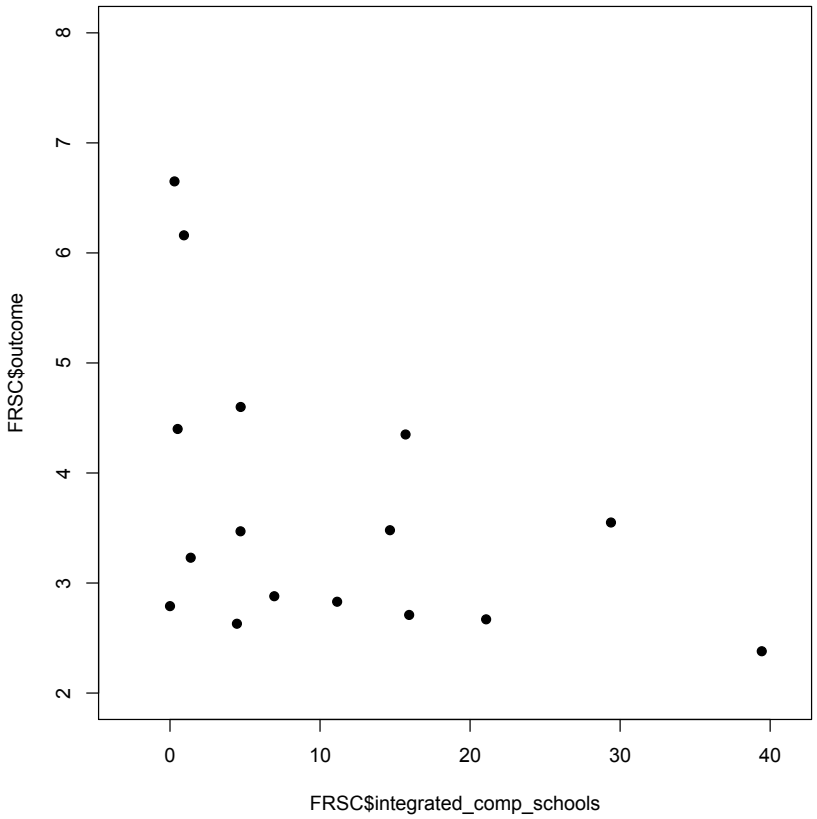
Instead of `text()` we can use `identify()` to manually label the cases. The code

```
> par(mar = c(4, 4, 1, 1))
> plot(FRSC$integrated_comp_schools, FRSC$outcome,
```

```
+ xlim = c(-3, 41), ylim = c(2, 8), pch = 19)
> identify(FRSC$integrated_comp_schools, FRSC$outcome,
+ labels = rownames(FRSC), cex = .7)
```

gives the following figure:

Figure 1.9: Scatterplot of `integrated_comp_schools` and outcome using `identify()`. The labels appear clicking on the cases

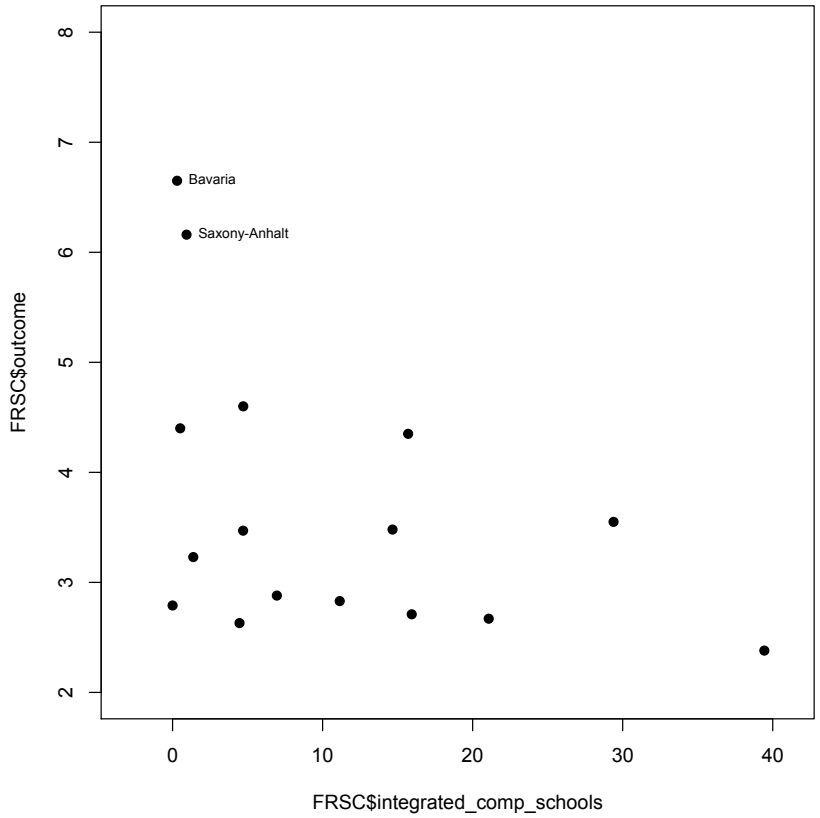


We can also define a condition for labeling cases. The code:

```
> par(mar = c(4, 4, 1, 1))
> plot(FRSC$integrated_comp_schools, FRSC$outcome,
+ xlim = c(-3, 41), ylim = c(2, 8), pch = 19)
> dots <- (FRSC$integrated_comp_schools < 10 & FRSC$outcome > 5)
> text(FRSC$integrated_comp_schools[dots], FRSC$outcome[dots],
+ labels = rownames(FRSC)[dots], pos = 4, cex = .7)
```

gives the following figure:

Figure 1.10: Scatterplot of `integrated_comp_schools` and outcome with conditioning

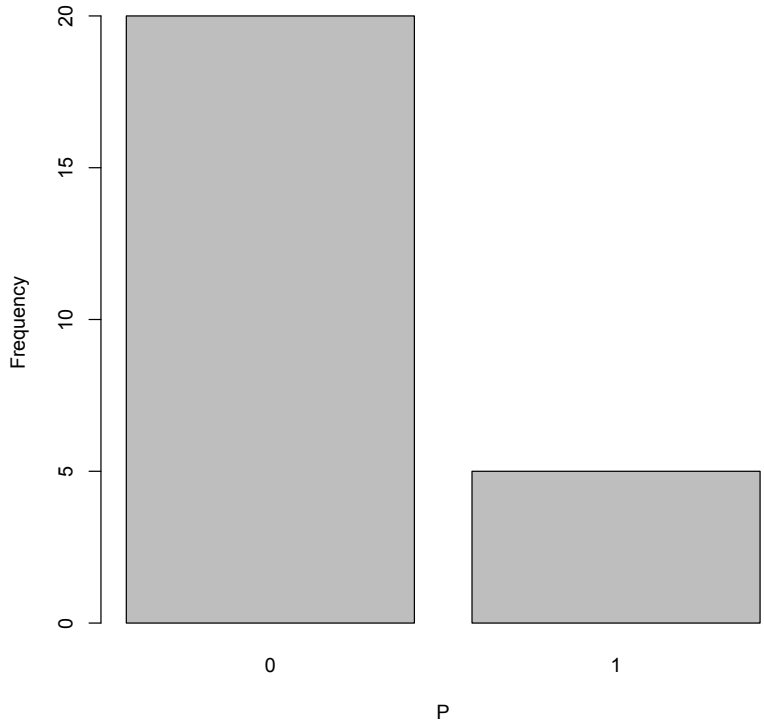


If we have crisp values we can use a barchart. The code:

```
1 > barplot(table(VisCS$P), xlab = "P", ylab = "Frequency")
```

gives the following figure:

Figure 1.11: Barchart of `P`



1.4.2 Calibration

1.4.2.1 Direct method

We use the `calibrate()` function from the `QCA` package with the option `type` set to `fuzzy`, to calibrate fuzzy sets. `thresholds` should indicate a vector with the thresholds. The code is the following:

```
> fulldayschools_fz <- calibrate(FRSC$full_day_schools,
+           type = "fuzzy", thresholds = c(3, 8.3, 20))
```

We can also use the `directCalibration()` function from `QCA3` package. `fullin`, `crossover` and `fullout` set the thresholds for full membership, the cross-over point and full non-membership. `infz` and `outfz` indicate fuzzy set score for full membership and full non-membership. The code is the following:

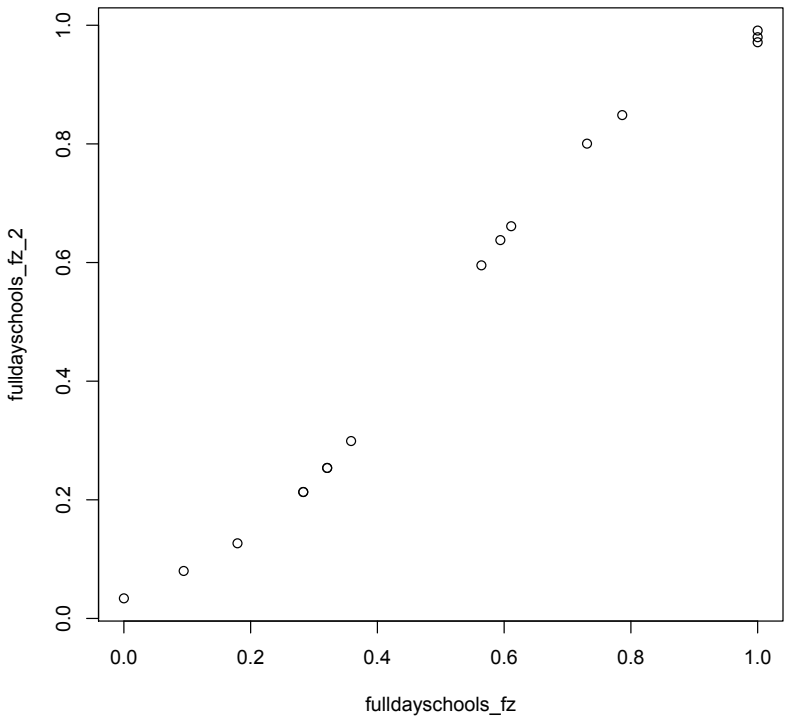
```
> fulldayschools_fz_2 <- directCalibration(FRSC$full_day_schools,
       fullin = 20, crossover = 8.3, fullout = 3)
```

The fuzzy scores are slightly different. See figure 1.12 to check the differences between `calibrate()` and `directCalibration()`. This depends on the choice of the function used for calibration. In fact, if we specify in `calibrate()` the option `logistic = TRUE` the fuzzy scores become identical. The code below specifies that the logistic function has to be used to calibrate the values:

```
> fulldayschools_fz_l <- calibrate(FRSC$full_day_schools,
+     type = "fuzzy",  thresholds = c(3, 8.3, 20),
+     logistic = TRUE)
```

Now, the scores are identical as shown in figure 1.13.

Figure 1.12: Scatterplot of fuzzy set calibration using `calibrate()` and `directCalibration()`



1.4.2.2 "Theoretical" calibration, crisp sets

We use the `calibrate()` function from the `QCA` package with the option `type` set to `crisp`, to calibrate crisp sets. The last vector sets the cross-over point. See the code below:

```
> fulldayschools_cs <- calibrate(FRSC$full_day_schools,
+   type = "crisp", thresholds = c(8.3))
```

As R is a statistical software, we can use it as such and calibrate the raw values in crisp set with a simple function: `ifelse()`. This performs a test on the vector of data. The function below means: if the value in vector `full_day_schools` is higher than 550 return 1, otherwise return 0. See the code below:

```
> fulldayschools_cs_2 <- ifelse(FRSC$full_day_schools > 8.3, 1, 0)
```

We get the same result. Check this building a table:

```
> table(fulldayschools_cs,fulldayschools_cs_2)
```

1.4.2.3 Indirect method

We assign membership values to cases, according to our substantive knowledge and then scores are tweaked using predictions, after a binomial or a beta regression, so to have predictions bounded between 0 and 1. Let us generate a variable `x` and find the quintiles:

```
> set.seed(123)
> x <- runif(20, 110, 40110)
> quant <- quantile(x, c(.2, .4, .5, .6, .8))
```

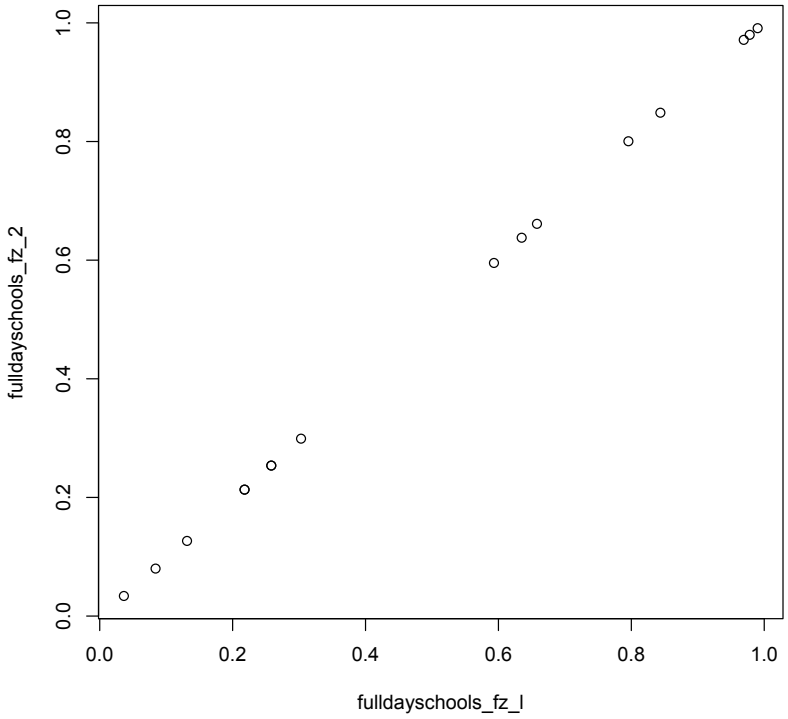
The we sort of recode the variable `x` to get the theoretically calibrated scores:

```
> x_cal <- NA  # Empty vector
> x_cal[x <= quant[1]] <- 0
> x_cal[x > quant[1] & x <= quant[2]] <- .2
> x_cal[x > quant[2] & x <= quant[3]] <- .4
> x_cal[x > quant[3] & x <= quant[4]] <- .6
> x_cal[x > quant[4] & x <= quant[5]] <- .8
> x_cal[x > quant[5]] <- 1
```

Then use function `indirectCalibration()` in `SetMethods` using the binomial regression (`binom` set as `TRUE`) or the the beta regression (`binom` set as `FALSE`):

```
> y_hat_bin <- indirectCalibration(x, x_cal, binom = TRUE)
> y_hat_bet <- indirectCalibration(x, x_cal, binom = FALSE)
```

Figure 1.13: Scatterplot of fuzzy set calibration using `calibrate()` with the `logistic` option set as `TRUE` and `directCalibration()`

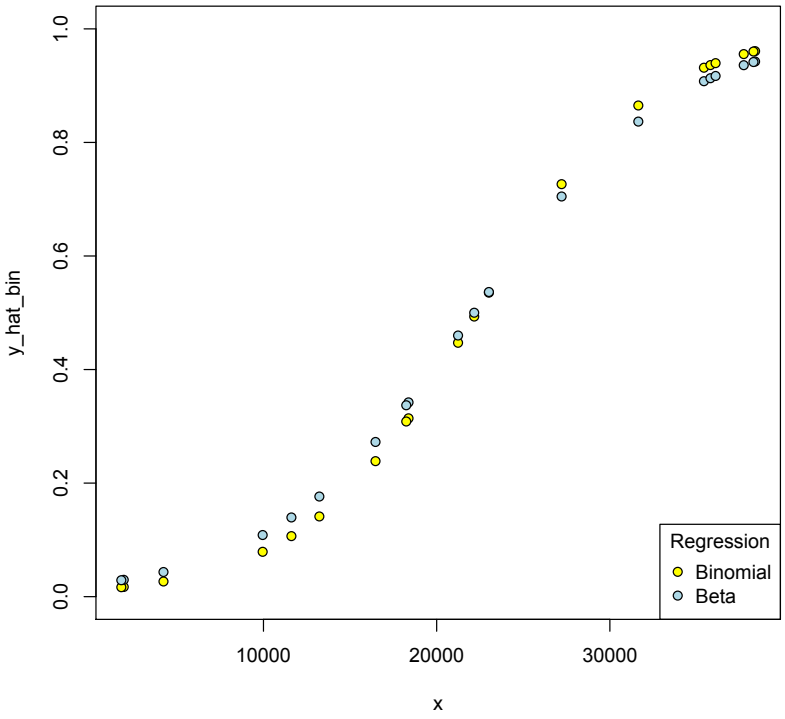


There are slight differences between the predictions. Check the differences using the following code:

```
> plot(x, y_hat_bin, ylim = c(0, 1), pch = 21,
+   col = "black", bg = "yellow")
> points(x, y_hat_bet, pch = 21, col = "black", bg = "lightblue")
> legend("bottomright", title = "Regression",
+   legend = c("Binomial", "Beta"),
+   pch = c(21, 21), col = c("black", "black"),
+   pt.bg = c("yellow", "lightblue"))
```

Which gives the following plot:

Figure 1.14: Scatterplot of indirect calibration using binomial and beta regression



Chapter 2

Notions and Operations in Set Theory: How to . . . Perform Operations on Sets

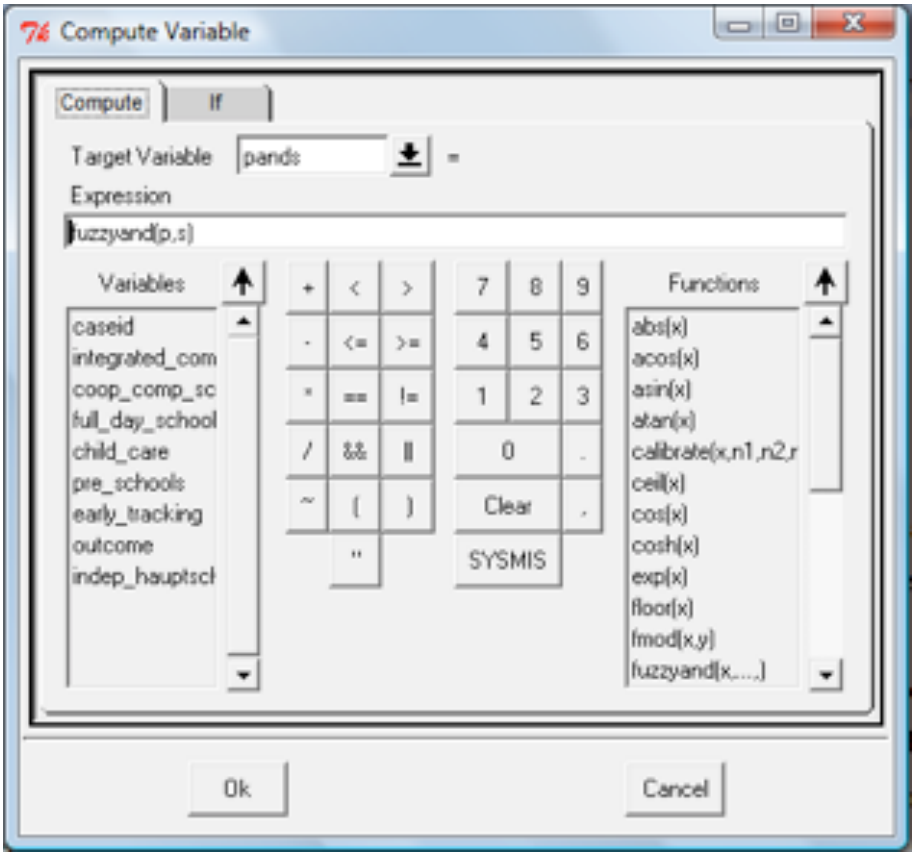
Overview:

- Calculating cases' membership scores in combined sets
  - Logical AND, logical OR, logical NOT
- Boolean calculator
- Empirical examples
  - Vis (2009): used in book
  - Ragin, Shulman, Weinberg, & Gran (2003): used in book

2.1 fsQCA 2.5

In fsQCA 2.5, routines exist for calculating each case's membership value in the intersection, union, or negation of more than one set. Click [File] [Open] [Data] and select file "Vis_2009_fs.csv", then [Open]. In order to generate each case's membership in the set of governments that were both in a difficult political (P) and socio-economic (S) situation click [Variables] [Compute], [Target Variable] "pands" - which is the name of the set to be added to the data sheet. In the field [Expression] type (or select from the list provided under [Functions]) "fuzzyand(P,S)" and click [Ok]. At the end of the data sheet, you find a new column labeled "pands". In order to create the union of sets (P + S), follow the same procedure but type "fuzzyor(P,S)" in the field [Expression]. The logical negation for a single set P is calculated by "fuzzynot(P)" in [Expression]. All three operations work both on crisp and fuzzy sets. See the figure 9.1 below.

Figure 2.1: Set theoretic operations using fsQCA 2.5.



The fsQCA 2.5 software only allows for one operation at a time. If the membership of cases is calculated for sets that include more than one logical operation (e.g. (P∗S) + R), then several steps are needed. This can be quite time-consuming and quickly reaches rather high levels of complexity. It is therefore often advisable to calculate the membership of cases in complex sets using off-the-shelf statistical packages, such as Stata, R, or Excel, using the appropriate syntax for determining the minimum and maximum values across a range of set membership scores.

2.2 Tosmana 1.3.2

There are two issues at stake: the calculation of a case's set membership scores in (complex) expressions based on its membership scores in the single constitutive sets; and the calculation of logical operations on complex expressions.

The Boolean calculator integrated in Tosmana offers a very useful tool for calculating intersections between and complements of complex logical expressions. Click [File] [Import] [Excel or fsQCA (csv file)], select "Ragin_etal_-2003.csv", then [Analysis] [Boolean Calculator]. Suppose, we are interested in the Boolean expression that describes those Indian villages that are simultaneously located on channels MN but not on the V (M∼V) or are they irrigated (I). In Boolean terms: M∼V + I. Suppose further, we are interested in all those villages that are not like these villages just described. Formally ∼(M∗∼V + I). In order to calculate the Boolean expression of this second expression, in window [1. Select Variable] select "M", in [2. Select Value] choose "1", then click [AND]. Now, in window [1. Select Variable] select "V", in [2. Select Value] choose "0", then click [OR]. Finally, in window [1. Select Variable] select "I", in [2. Select Value] choose "1", then click [Add expression to list: add]. The expression M∼V + I now appears in the window at the bottom of the interface, using Tosmana's notation "m{1}v{0}+i{0}". In order to negate this expression, point the cursor on it and click [Compute Complement]. This yields ∼I∼M + ∼IV. By factoring out ∼I, this can be rewritten as ∼I (M + ∼V). If you now highlight the two expressions in the window and click [Compute Intersection], the empty set results, as stated by the rule of the excluded middle.

2.3 Stata

In Stata, the command line for calculating each case's membership score in the complex set of, say P∗S + ∼R reads as follows:

```
1 gen pandsorr = max(min(P, S), R)
```

2.4 R

To show how to perform operation in set theory in R we generate some fake data. We set the state of RNG (random number generator) to be sure that when we draw numbers from a distribution we get always the same draws. This is standard procedure when using simulation or fake data, so it is possible to replicate it:

```
> set.seed(123)
```

In R we can also perform set theory operation, both for crisp and fuzzy sets. Here, we show that function `pmin()` takes the parallel minimum of the vectors included in between the brackets. This operation is needed when using the logical AND operator:

```
> x <- runif(10, 0, 1) # We just draw from a uniform distribution
> x

[1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673 0.0455565
    0.5281055 0.8924190
[9] 0.5514350 0.4566147

> y <- runif(10, 0, 1)
> y

[1] 0.95683335 0.45333416 0.67757064 0.57263340 0.10292468 0.89982497
     0.24608773
[8] 0.04205953 0.32792072 0.95450365

> pmin(x, y)

[1] 0.28757752 0.45333416 0.40897692 0.57263340 0.10292468 0.04555650
     0.24608773
[8] 0.04205953 0.32792072 0.45661474
```

The function `pmax()` takes the parallel maximum of the vectors included in between the brackets. This is needed when using the logical OR operator:

```
> pmax(x, y)

[1] 0.9568333 0.7883051 0.6775706 0.8830174 0.9404673 0.8998250
    0.5281055 0.8924190
[9] 0.5514350 0.9545036
```

Both functions work for crisp and fuzzy sets:

```
> x_c <- rbinom(20, 1, .4) # We draw from a binomial distribution
> x_c

[1] 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 0 0 0

```

```
> y_c <- rbinom(20, 1, .6)
> y_c

[1] 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 0 1

> pmin(x_c, y_c)

[1] 1 1 1 1 1 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0

> pmax(x_c, y_c)

[1] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1
```

And for more than two vectors:

```
> z <- runif(10, 0, 1)
> z

[1] 0.66511519 0.09484066 0.38396964 0.27438364 0.81464004 0.44851634
     0.81006435
[8] 0.81238951 0.79434232 0.43983169

> pmin(x, y, z)

[1] 0.28757752 0.09484066 0.38396964 0.27438364 0.10292468 0.04555650
     0.24608773
[8] 0.04205953 0.32792072 0.43983169

> pmax(x, y, z)

[1] 0.9568333 0.7883051 0.6775706 0.8830174 0.9404673 0.8998250
    0.8100644 0.8924190
[9] 0.7943423 0.9545036
```

For crisp sets we can also use logical operators: `|` for the OR operator and `&` for the AND operator:

```
> (x_c | y_c)

[1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
      TRUE  TRUE  TRUE
[15]  TRUE  TRUE  TRUE FALSE FALSE  TRUE

> (x_c & y_c)

[1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE
      TRUE FALSE  TRUE
[15] FALSE FALSE  TRUE FALSE FALSE FALSE
```

They can also be used in combination as well (just as using a calculator):

```
> pmax(pmin(x, y), pmin(z, y))

[1] 0.66511519 0.45333416 0.40897692 0.57263340 0.10292468 0.44851634
     0.24608773
```

```
4 [8] 0.04205953 0.32792072 0.45661474
```

The negation is calculated differently for crisp and fuzzy sets. For crisp sets we can use `!`, which means that we want as result something not equal to the object:

```
1 > n_x_c <- !x_c
2 > n_x_c
3
4 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
      FALSE FALSE FALSE
5 [15]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

Up here we obtained the a vector of values that are not those in x_c. We can also use the `ifelse()` function, as shown earlier:

```
1  > n_x_c <- ifelse(x_c == 0, 1, 0)
2  > n_x_c
3
4  [1] 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 1 1 1
5
6  > pmax(n_x_c, x)
7
8  [1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673 0.0455565
       1.0000000 1.0000000
9  [9] 1.0000000 1.0000000 0.2875775 0.7883051 0.4089769 0.8830174
       1.0000000 1.0000000
10 [17] 0.5281055 1.0000000 1.0000000 1.0000000
```

For fuzzy sets, it is simply a subtraction:

```
1 > n_x <- 1 - x
2 > n_x
3
4 [1] 0.71242248 0.21169486 0.59102308 0.11698260 0.05953272 0.95444350
      0.47189451
5 [8] 0.10758096 0.44856499 0.54338526
```

Chapter 3

Set Relations: How to . . . Use Graphs for Identifying Sufficiency and Necessity Relations

Overview:

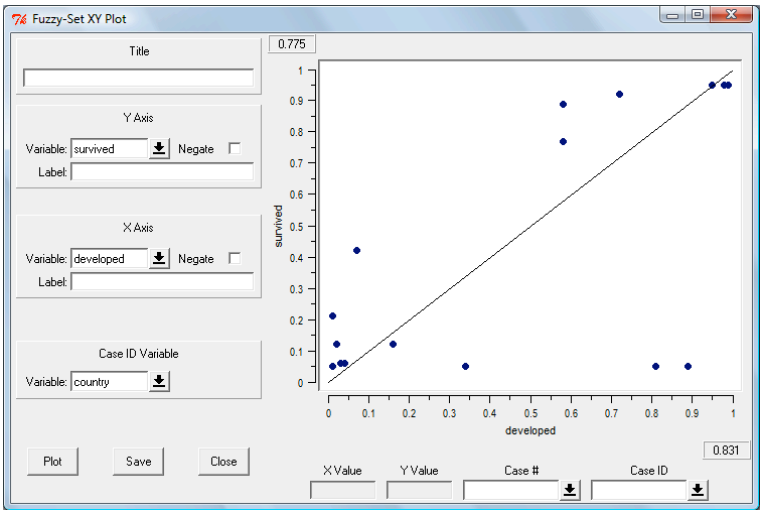
- Cross-tabs for crisp sets
- XY plots for fuzzy sets
- Empirical examples
  - Rihoux & De Meur (2009): see book
  - Ragin (2009): see book

3.1 fsQCA 2.5

Any software packages capable of producing two-by-two tables of binary data can be used for visualizing set relations involving crisp sets. In fsQCA 2.5, click [Analyze] [Statistics] [Crosstabs] and specify the outcome set under "outcome" and the conditions to be examined under "causal conditions". Separate two-by-two tables are created for each single condition. When interpreting the table, be aware that the 0,0 cell is in the upper left corner rather than the lower left, as it is in all examples throughout SMSS. Click [File] [Open [Data] "lipset_cs.csv". Under [Analyze] [Statistics] [Cross-tabs] specify as "outcome survived" and as condition "developed". One cell is empty: which type of set relation have you discovered? When using fuzzy sets, in fsQCA 2.5, click [Graphs] [Fuzzy] [XY Plot]. Specify the outcome under "Y axis" and the condition under "X axis". If you want to plot a combination of conditions, this combination must already exist in the data set. You can use the functions under [Variables] [Compute] in fsQCA 2.5 or any other appropriate software

for creating this new condition. If our dataset contains string variables with the case ID, then you can specify this under "Case ID variable." Click [Plot] to produce the XY plot. When clicking on one of the single dots in the plot, the case's membership in X and Y, its row in the dataset ("case #"), and the case's name is revealed below the plot. To practice, open file "lipset_fs.csv". Under [Graphs] [Fuzzy] [XY plot], specify as outcome "survived", as condition "developed", and as as Case ID Variable "country". Look at figure 9.1. Do you see a perfect set relation? If not, which cases are not in line with a perfect set relation?

Figure 3.1: XY plot using fsQCA 2.5.



3.2 Tosmana 1.3.2

Set relations cannot be visualized in Tosmana.

3.3 Stata

In Stata, use the command `fzplot`. It requires that both the outcome and the conditions have single letter names. Logical AND configuration can be plotted even if only the single conditions exist in the data. Simply write, e.g., `fzplot y aBc` if you want to create an XY plot with ∼AB∼C as the condition to be plotted against outcome Y.

Applied to the data set "lipset_fs.csv", the syntax for producing an XY plot for outcome "survived" and condition "developed" looks as follows:

```
insheet using "~/Lipset_fs.csv"
gen y = survived
gen x = developed
fzplot y x, mlabel(country)
```

3.4 R

Producing cross-tables in R is very simple (but at the same time time consuming). Let's load the Lipset crisp set in the crisp set version (this is exercise 9):

```
> data(LipsetCS)
> head(LipsetCS)
.
.
.
```

We use the function `table()`:

```
> tb <- table(LipsetCS$SURVIVED, LipsetCS$DEVELOPED)
> tb

  0 1
0 8 2
1 0 8
```

If we want the sum scores we have to:

```
> r_s <- margin.table(tb, margin = 1) # for the rows
> r_s

0  1
10 8

> c_s <- margin.table(tb, margin = 2) # for the columns
> c_s

0  1
8 10

> t_s <- margin.table(tb, margin = NULL) # for the total
> t_s

[1] 18
```

Then we build our table:

```
> csTable <- cbind(tb, c_s)
> csTable <- rbind(csTable, c(r_s, t_s))

# give some labels
> rownames(csTable) <- c(0, 1, "Tot.")
> colnames(csTable) <- c(0, 1, "Tot.")

# and we have our nice table
> csTable

      0 1 Tot.
0     8 2    8
1     0 8   10
Tot. 10 8   18
```

We can export it in LaTeX using `xtable()` and in excel/word:

```
> xtable(csTable)
> write.table(csTable, file = "table", sep = "\t",
+   row.names = TRUE)
> library(stable)
> xtable(csTable)
```

Table 3.1: Frequency table for crisp-sets.

	0	1	Tot.
0	8	2	8
1	0	8	10
Tot.	10	8	18

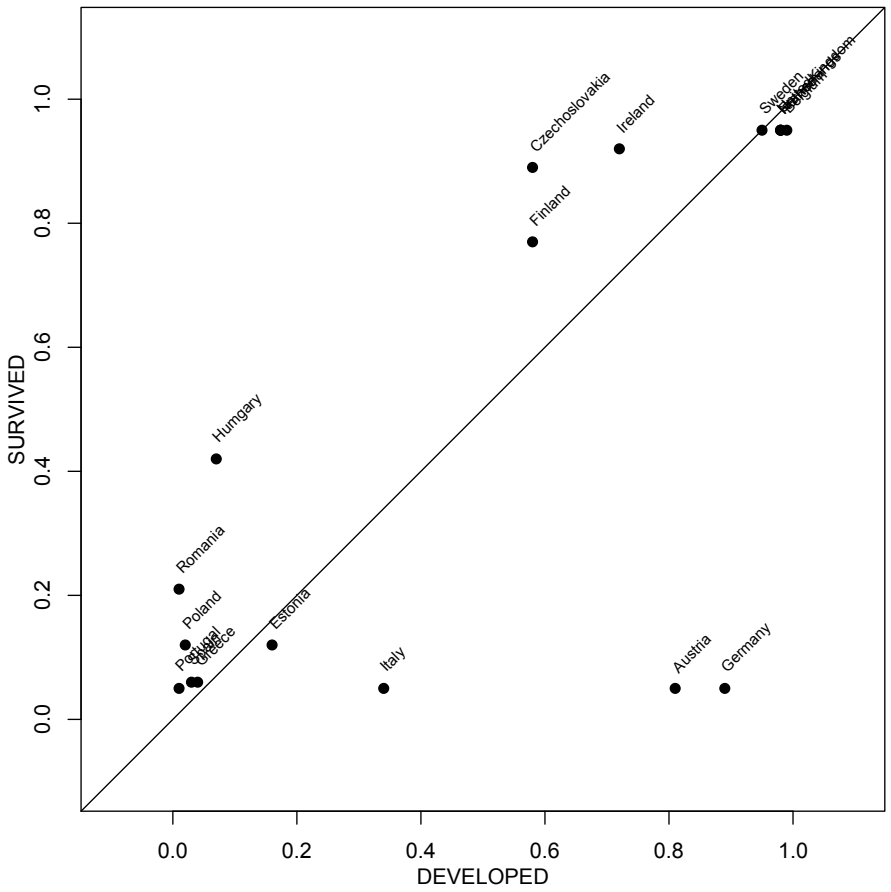
Making plots in R is also very easy and flexible (this is exercise 10). Load the Lipset dataset in fuzzy version:

```
> data(LipsetFS)
> head(LipsetFS)
.
.
.
```

To produce a xyplot we use the `plot()` function with some options and customization to make it nicer (see figure 9.2):

```
> par(mar = c(3, 3, 1, 1), mgp = c(2, 1, 0))
> plot(LipsetFS$DEVELOPED, LipsetFS$Survived,
+   xlab = "DEVELOPED", ylab = "SURVIVED",
+   xlim = c(-.1, 1.1), ylim = c(-.1, 1.1), pch = 19)
> abline(0, 1)
> text(LipsetFS$DEVELOPED, LipsetFS$Survived + .03,
+   labels = rownames(LipsetFS), cex =.7,
+   srt = 45, adj = 0)
```

Figure 3.2: XY plot using R



Remember that you can use `identify()`. We can also label cases using conditional statements (see figure 5.3):

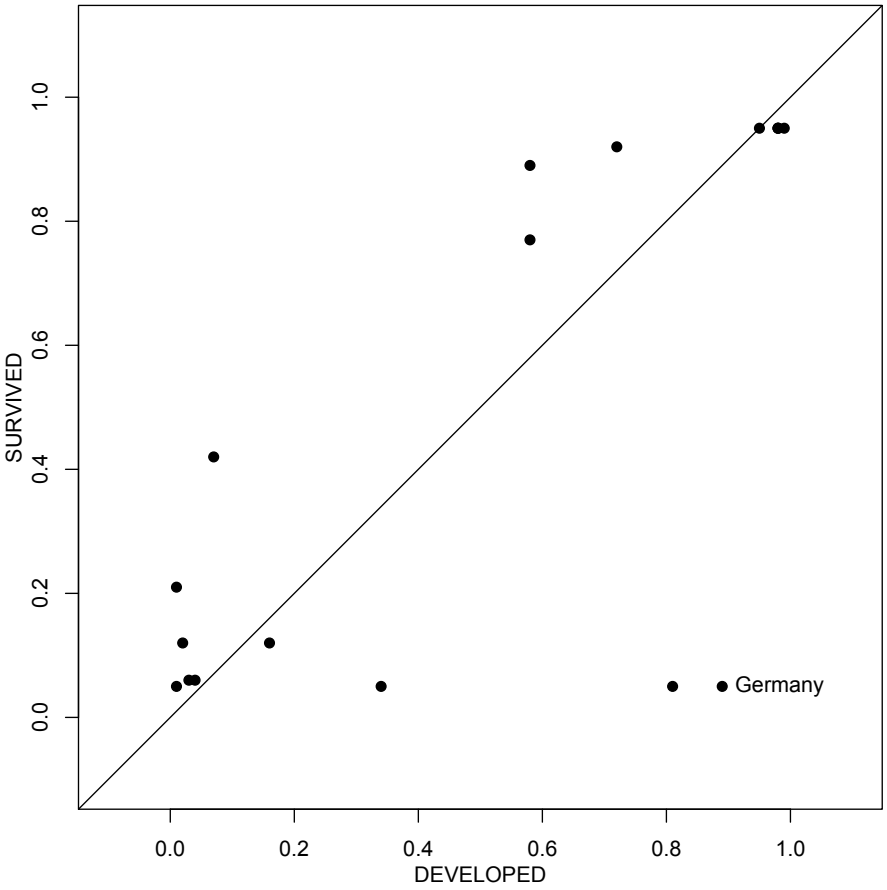
```
> par(mar = c(3, 3, 1, 1), mgp = c(2, 1, 0))
> plot(LipsetFS$DEVELOPED, LipsetFS$Survived,
+   xlab = "DEVELOPED", ylab = "SURVIVED",
+   xlim = c(-.1, 1.1), ylim = c(-.1, 1.1), pch = 19)
> abline(0, 1)
> dots <- (LipsetFS$DEVELOPED > .85 & LipsetFS$Survived < .2)
> text(LipsetFS$DEVELOPED[dots], LipsetFS$Survived[dots],
+   labels = rownames(LipsetFS)[dots], pos = 4, cex = 1)
```

We can make plots for all the conditions, one by one. For those who want to plunge into a little bit of code: there is a possibility of using a “loop" to produce all the plots in one shot (see figure 3.4):

```
> par(mfrow = c(2, 3), mar = c(3, 3, 1, 1), mgp = c(2, 1, 0))
```

```
> for (i in 2:6){
+ plot(LipsetFS[, i], LipsetFS$Survived, pch = 19,
+ xlim=c(-.1, 1.1), ylim=c(-.1, 1.1),
+ xlab = colnames(LipsetFS)[i], ylab = "SURVIVED")
+ abline(0, 1)
+ text(LipsetFS[, i], LipsetFS$Survived + .03,
+ labels = rownames(LipsetFS), cex = .7,)
+ }
```

Figure 3.3: XY plot with conditional statements using R

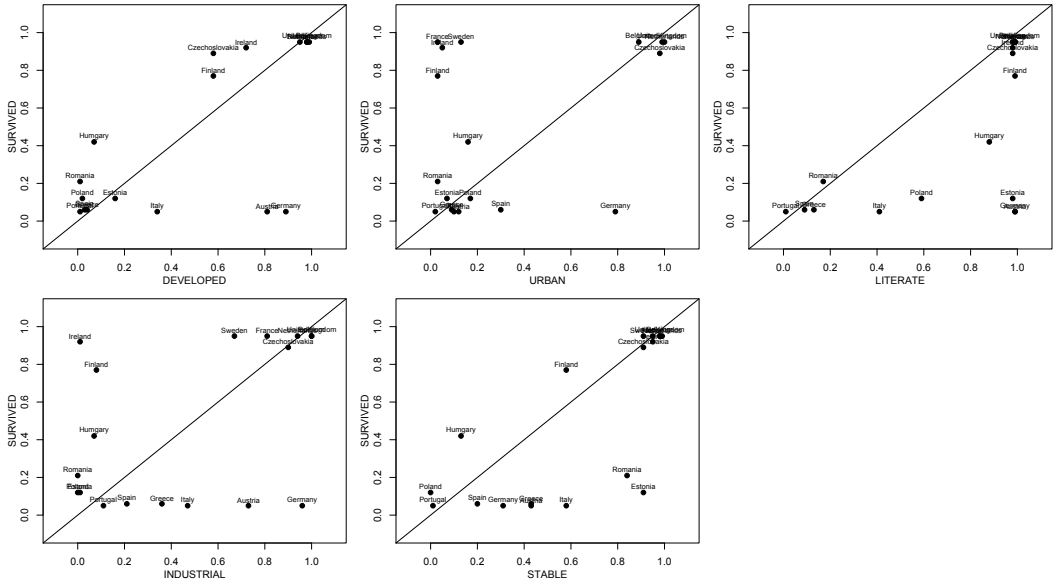


Similar (but better looking) plot can be made using `lattice`, but it needs some preparation (see figure 3.5):

```
> library(reshape) # this is needed for melt()
> Dat <- melt(LipsetFS[, 2:6])
# Using as id variables
> Dat$SURVIVED <- rep(LipsetFS[, 1], 5)
> xyplot(SURVIVED ~ value | variable, Dat,
+    ylim = c(-.1, 1.1), xlab = "", as.table = T,
```

```
+            layout = c(2, 3),
+       panel = function(x, y, ...){
+          panel.xyplot(x, y, ...)
+          panel.abline(0, 1)
+ })
```

Figure 3.4: Multiple XY plot using R



Without making all this effort, we have a function for making xyplots: `fsplot()` (see figure 3.6), but it is not very flexible, later we present a function for plotting xyplots:

```
fsplot(Survived ~ DEVELOPED, data = LipsetFS)
```

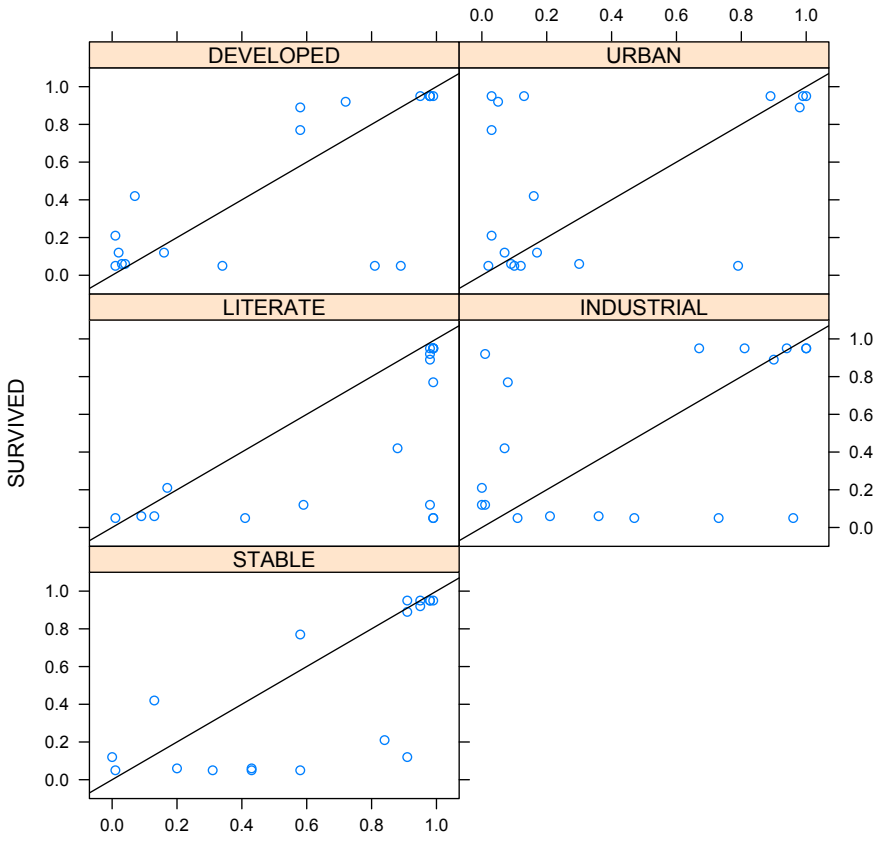
It can be included in the previous loop to generate all the plot in once, making a slight modification to `fsplot()` (see figure 3.7):

```
> par(mfrow = c(2, 3), mar = c(3, 4, 3, 1), mgp = c(2, 1, 0))
> for (i in 2:6){
+ fsplot(Survived ~ LipsetFS[,i], data = LipsetFS, main = "",
+ xlab = names(LipsetFS)[i])
+ }
```

But it does not look very nice. We can use the `xy.plot()` function in `SetMethods` (see figure 3.8):

```
> xy.plot(LipsetFS$DEVELOPED, LipsetFS$Survived,
+ labs = rownames(LipsetFS))
```

Figure 3.5: Multiple XY plot using R and `lattice`



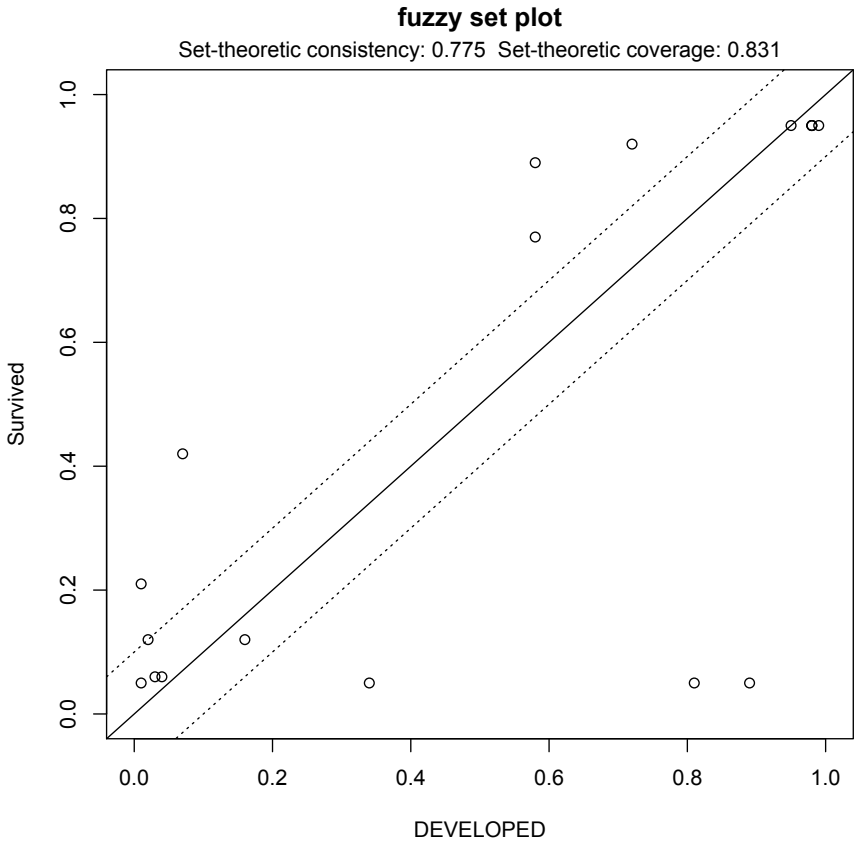
The function has several options. Type `?xy.plot` to look what they are: It can be used in a loop as well (see figure 3.9):

```
> par(mfrow = c(2, 3))
> for (i in 2:6){
+ xy.plot(LipsetFS[,i], LipsetFS$Survived,
+ main = paste("XY Plot for", names(LipsetFS)[i], sep=" "),
+ cex.axis = .7,
+ xlab = "Condition", cex.fit = .5, labs = rownames(LipsetFS))
+ }
```

In `SetMethods` there another function producing the same plot using the `lattice` environment called `xy.plot.lat()` (see figure 3.10):

```
> xy.plot.lat(LipsetFS$DEVELOPED, LipsetFS$Survived, case.lab = TRUE,
+ labs = rownames(LipsetFS))
```

Figure 3.6: Multiple XY plot using R and `fsplot()`



The following code shows how to do Exercise 11:

```
> data(VisFS)
> VisFS$n_r <- 1 - VisFS$r
> VisFS$p_s_nr <- VisFS(Vis$p, Vis$s, Vis$n_r)
> xy.plot(VisFS$p_s_nr, VisFS$u, labs = rownames(VisFS))
```

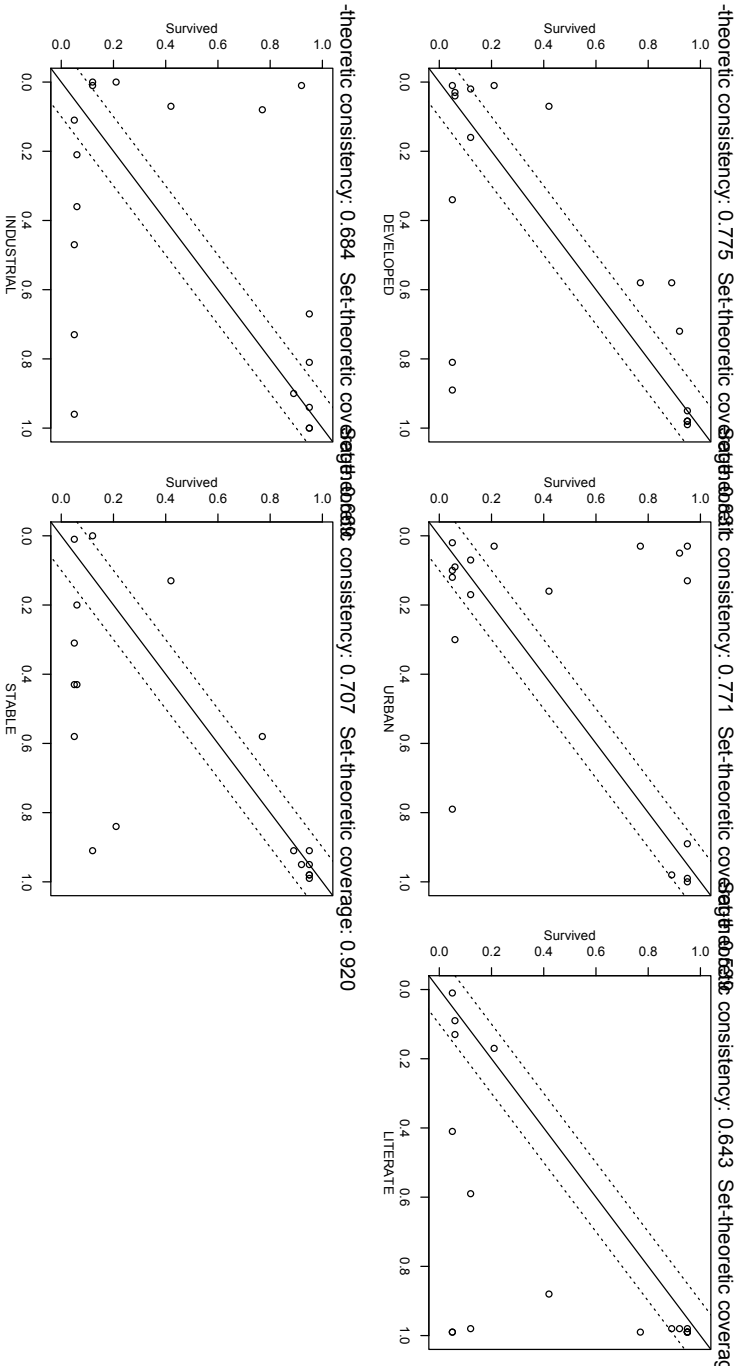
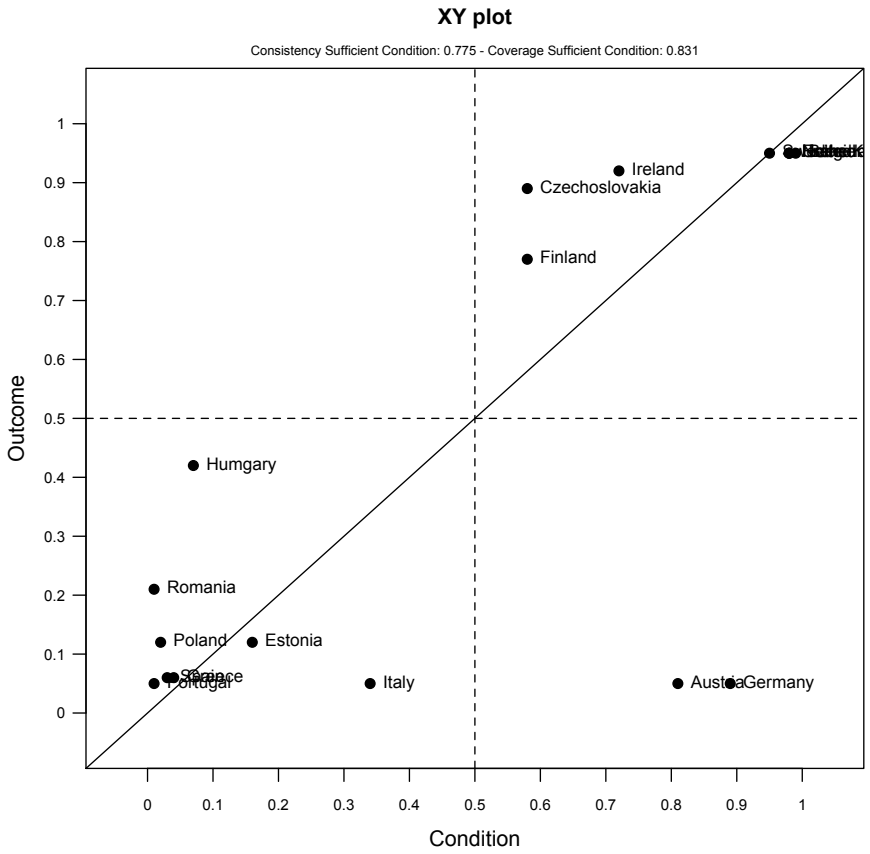


Figure 3.7: Multiple XY plot using R and `fsplot()` in a "loop"

Figure 3.8: XY plot using R and `xy.plot`



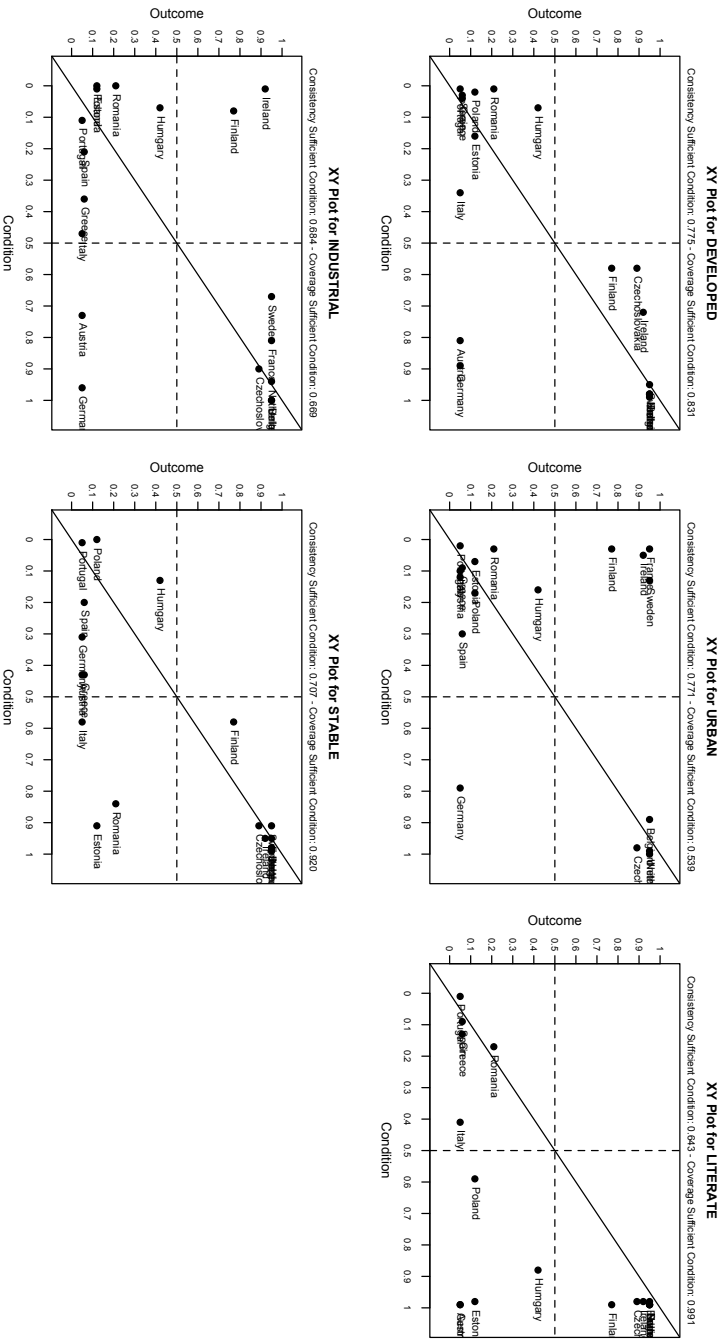
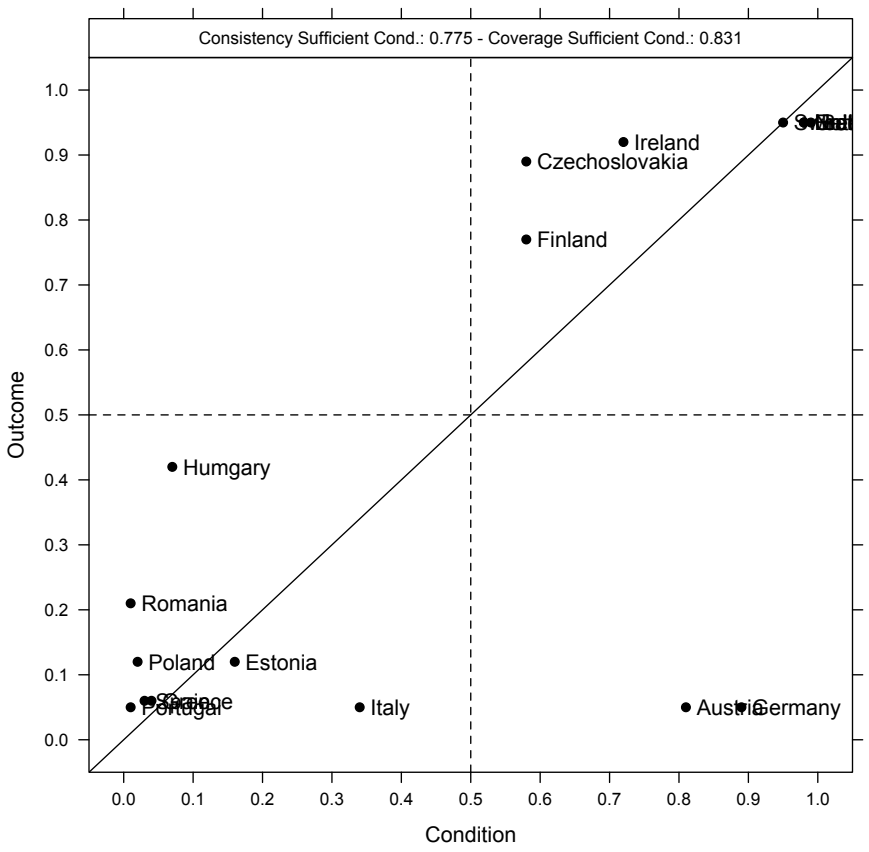


Figure 3.9: Multiple XY plot using R and `xy.plot()` in a "loop"

Figure 3.10: XY plot using R and `xy.plot.lat()`



Chapter 4

Truth Tables: How to . . . Create and Logically Minimize Truth Tables with the Appropriate Software

Overview:

- Transforming data sets with set membership scores into truth tables
- Logically minimizing truth tables
- Empirical examples
  - Fake data `Selbst.disappear` or "Selbst_1_data_matrix.csv"
  - Lipset (1956)

4.1 fsQCA 2.5

With crisp set data, click [Analyze] [Crisp Sets] [Truth Table Algorithm] and with fuzzy set data [Analyze] [Fuzzy Sets] [Truth Table Algorithm]. A new window opens in which we are asked to specify the outcome and the conditions. With [Set], we choose the occurrence of the set highlighted in the left panel as the outcome to be analyzed. With [Set Negated], we create the truth table with the non-occurrence of the set as the outcome to be explained. After clicking [Run], the truth table appears in a new window. It consists of 2k rows, k being the number of conditions. The first k columns represent the conditions specified in the previous step. Column "number" indicates the number of cases that have a membership greater than 0.5 in the given truth table row. By adding up the numbers appearing in column number, we thus obtain the number of cases under analysis (unless one or more cases have a fuzzy-set membership score of exactly 0.5 in one or more of the conditions; see chapter 4.2 of SMSS). The column for the outcome value is empty. It should

be filled based on the information of whether a given row can be considered a sufficient condition for the outcome of interest. Rows that contain enough cases (see column "number") and that are subsets of the outcome can be considered sufficient for that outcome. This information is provided in column "raw consistency." The precise meaning of this and the columns 'pre consist' and 'product' is explained in chapters 5 and 9. With fully specified truth tables, each row has enough cases and the values in column "raw consist" are either 1 or 0. Rows with raw consistency of 1 are sufficient and thus included in the logical minimization. We can either directly insert the values 1 or 0 in the outcome column or we click [Edit] [Delete and code . . .] [Delete rows with number less than "1"], [and set outcome to 1 for rows with consis>0.99].

We now have a truth table that can be subjected to logical minimization. In order to do this, we click [Specify Analysis]. A new window opens. The rows in this matrix list all the values that can be encountered in the outcome column of a truth table. In our simple examples so far, only the values of 1 and 0 occur. The columns indicate the options how to treat rows with outcome 1 or 0. Setting rows to "True" means that they enter the logical minimization procedure, "False" that they are not included. After clicking [Run], a new window opens with the solution formula for sufficiency. Here is an important practical note: when using the [Truth Table Algorithm] option in fsQCA 2.5, we always have to specify "Positive Cases (1)" as "True" and "Negative Cases (0)" as "False." If we want to analyze the sufficient condition for the non-occurrence of the outcome, then, as explained, we need to specify the negation of the outcome set at the beginning of the process of creating a truth table.

In order to practice, open file "Selbst_1_data_matrix.csv". Click [Analyze] [Crisp Sets] [Truth Table Algorithm] and specify as [Outcome] "Y" and as [Causal Conditions] all other three remaining sets in the data. Tick the box [show solution cases in output] with "label" specified as [variable name column]. Click [Run] [Edit] [Delete and Code], set the frequency threshold (upper box) to 1 and the consistency threshold (lower box) to 0.99, and click [Ok]. See figure 9.1.

Each truth table row now has either 1 or 0 in the outcome column. Click [Specify Analysis] and set [Positive Cases (1)] to [True] and [Negative Cases (0)] to [False]. Press [Run]. See figure 9.2.

This procedure yields the following solution formula:

$$\sim A + \sim B{*}C \rightarrow Y$$

Figure 4.1: Truth table fsQCA 2.5.

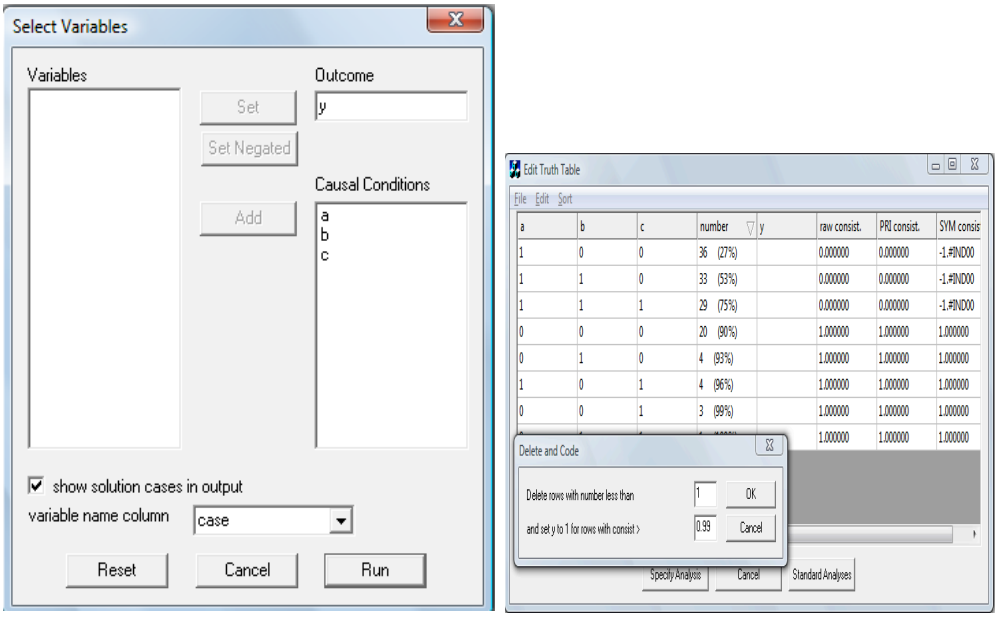
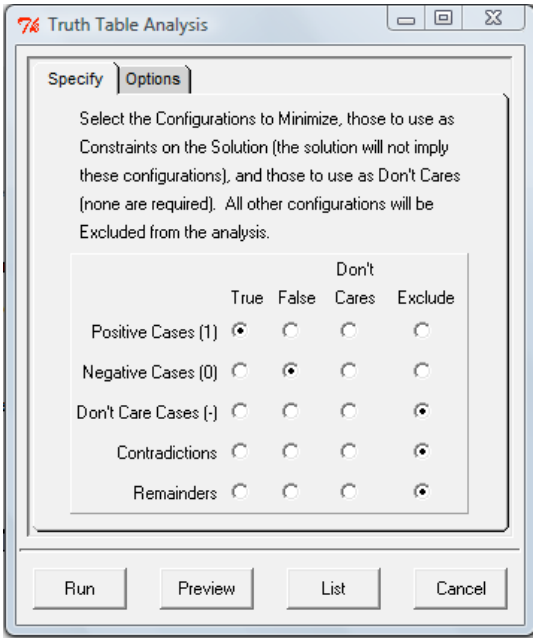


Figure 4.2: Specify analysis in fsQCA 2.5.



The software also reports consistency and coverage values (see chapter 5 of SMSS) and lists those cases that have membership in each of the sufficient

terms.

4.2 Tosmana 1.3.2

When using Tosmana, click [Analysis] [Start (MV)QCA]. A new window opens. From the panel on the left, we choose the outcome and conditions. If a "case descriptor" is also chosen, case labels will be displayed in their respective truth table row and the sufficiency solution term. Clicking [Truth Table], a new window opens that displays the truth table, with the first columns containing the conditions and the penultimate column indicating whether all the cases that are members of the respective row also members of the outcome (1) or not (0). The last columns shows the cases in each row.

The logical minimization of this truth table is performed by returning to the previous window. On the right-hand side we find a panel that looks similar to the one encountered in fsQCA 2.5. The rows list all values that could be encountered in the outcome column of a truth table. So far, we have only dealt with values of 1 and 0 but in the subsequent chapters, when we handle more complex (and realistic) data, contradictions and remainders will be the rule rather than the exception.[1] For the sufficiency analysis of the occurrence of the outcome, we set the marker at Outcome 1 = explain and Outcome 0 = exclude.

4.3 Stata

In Stata, the command `fuzzy settest( )` displays, for all logically possible combinations of conditions (a.k.a. truth table rows), the number of cases that are members and whether it is a subset of the outcome or the non-occurrence of the outcome. Unlike fsQCA 2.5, Stata does not produce a truth table in which the outcome value can be directly coded using the decision rules described above and further elaborated in chapter 7 of SMSS. For the logical minimization procedure, we use the command `reduce`.

4.4 R

To show how to create and minimize a truth table in R we use the dataset called `Selbst.disappear` available in `SetMethods`:

[1] Neither of the authors of this book have ever encountered a set-theoretic study where the outcome value was missing, the fifth option in the panel.

```
> data(Selbst.disappear)
> head(Selbst.disappear)

   A B C Y
A1 0 0 0 1
A2 0 0 0 1
A3 0 0 0 1
A4 0 0 0 1
A5 0 0 0 1
A6 0 0 0 1
```

We can build a truth table for crisp sets using both packages `QCA3` and `QCA`. In `QCA3` there is the function `cs_truthTable()` which produces a truth table for crisp set values:

```
> TTcs <- cs_truthTable(mydata = Selbst.disappear,
+          outcome = "Y",
+          conditions = c("A", "B", "C"))

# If we just need the truth table type:
> TTcs$truthTable

   A B C NCase freq1 freq0 OUT
14 0 0 0    20    20     0   1
15 1 0 0    36     0    36   0
17 0 1 0     4     4     0   1
18 1 1 0    33     0    33   0
23 0 0 1     3     3     0   1
24 1 0 1     4     4     0   1
26 0 1 1     1     1     0   1
27 1 1 1    29     0    29   0
```

To minimize the truth table we use `reduce()`:

```
> reduce(TTcs)

Call:
reduce(x = TTcs)

truthTable with 8 configuration(s)

   A B C NCase freq1 freq0 OUT
14 0 0 0    20    20     0   1
15 1 0 0    36     0    36   0
17 0 1 0     4     4     0   1
18 1 1 0    33     0    33   0
23 0 0 1     3     3     0   1
24 1 0 1     4     4     0   1
26 0 1 1     1     1     0   1
27 1 1 1    29     0    29   0


14 A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14,
  A15, A16, A17, A18, A19, A20
```

```
15 A29, A30, A31, A32, A33, A34, A35, A36, A37, A38, A39,
  A40, A41, A42, A43, A44, A45, A46, A47, A48, A49, A50, A51, A52,
      A53, A54, A55, A56, A57, A58, A59, A60, A61, A62, A63, A64
17  A24, A25, A26, A27
18  A69, A70, A71, A72, A73, A74, A75, A76, A77, A78, A79, A80,
  A81, A82, A83, A84, A85, A86, A87, A88, A89, A90, A91, A92, A93,
      A94, A95, A96, A97, A98, A99, A100, A101
23 A21, A22, A23
24 A65, A66, A67, A68
26 A28
27 A102, A103, A104, A105, A106, A107, A108, A109, A110, A111,
  A112, A113, A114, A115, A116, A117, A118, A119, A120, A121,
  A122, A123, A124, A125, A126, A127, A128, A129, A130

-----------------
Explaining 5 configuration(s)

-----------------
Prime implicant No. 1 with 2 implicant(s)

a + b*C

Common configuration: None
```

Instead, if we use `QCA` another function is available that is `truthTable()`:

```
> TTcs <- truthTable(Selbst, outcome = "Y", complete = TRUE,
+          show.cases = FALSE, sort.by = c("incl", "n"))
> TTcs

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

   A  B  C  OUT n  incl  PRI
1  0  0  0   1  20 1.000 1.000
3  0  1  0   1   4 1.000 1.000
6  1  0  1   1   4 1.000 1.000
2  0  0  1   1   3 1.000 1.000
4  0  1  1   1   1 1.000 1.000
5  1  0  0   0  36 0.000 0.000
7  1  1  0   0  33 0.000 0.000
8  1  1  1   0  29 0.000 0.000
```

To get only the truth table:

```
> TTcs$tt

  A B C OUT  n incl PRI
1 0 0 0   1 20    1   1
3 0 1 0   1  4    1   1
6 1 0 1   1  4    1   1
2 0 0 1   1  3    1   1
4 0 1 1   1  1    1   1
5 1 0 0   0 36    0   0
```

```
7 1 1 0   0 33    0   0
8 1 1 1   0 29    0   0
```

To minimize the truth table we use `eqmcc()`:

```
> eqmcc(TTcs)

S1: a + bC
```

We can build a truth table for fuzzy sets with these two packages. We show how to produces a truth table for fuzzy set values using the Lipset fuzzy data `LipsetFS`:

```
> data(LipsetFS)
> head(LipsetFS)
.
.
.
```

If we use `QCA3` we use the function `fs_truthTable()`:

```
> TTfs <- fs_truthTable(LipsetFS,
+       outcome = "Survived",
+       conditions = c("DEVELOPED", "URBAN",
+             "LITERATE", "INDUSTRIAL", "STABLE"),
+       consistency_cutoff = 0.8)

# To get the truth table only
> TTfs$truthTable

# To sort the table by consistency values, make an object
> TTfsSort <- TTfs$truthTable

# and use order()
> TTfsSort <- TTfsSort[order(TTfsSort$Consistency, decreasing = TRUE)
    ,]
> TTfsSort

    DEVELOPED URBAN LITERATE INDUSTRIAL STABLE OUT freq1 freq0 NCase
       Consistency
243         1     1        1          1      1   1     4     0     4
      0.9042056
213         1     0        1          0      1   1     2     0     2
      0.8042705
240         1     0        1          1      1   0     0     2     2
      0.7087719
212         0     0        1          0      1   0     0     1     1
      0.5285714
131         0     0        1          0      0   0     0     2     2
      0.5209003
162         1     1        1          1      0   0     0     1     1
      0.4452555
159         1     0        1          1      0   0     0     1     1
      0.3782051
```

```
25 203          0     0         0           0       1    0      0      2      2
       0.2780269
26 122          0     0         0           0       0    0      0      3      3
       0.2159763
27     priConsistency sqrtProduct                                        Cases
28 243      0.88579387  0.80093979 Belgium,Czechoslovakia,Netherlands,
       UnitedKingdom
29 213      0.71938776  0.57858232
       Finland,Ireland
30 240      0.63436123  0.44961744
       France,Sweden
31 212      0.22807018  0.12055138                                    Estonia
32 131      0.11309524  0.05891135
       Humgary,Poland
33 162      0.05000000  0.02226277                                    Germany
34 159      0.03960396  0.01497842                                    Austria
35 203      0.00000000  0.00000000
       Italy,Romania
36 122      0.00000000  0.00000000                                          Greece,
       Portugal,Spain
37
38 # For reduction we use, as before, reduce() and check the options
39 > reduce(TTfs) # using the truth table as object
40
41 Call:
42 reduce(x = TTfs)
43
44 truthTable with 9 configuration(s)
45
46     DEVELOPED URBAN LITERATE INDUSTRIAL STABLE OUT freq1 freq0 NCase
           Consistency
47 243         1     1        1          1      1   1     4     0     4
       0.9042056
48 240         1     0        1          1      1   0     0     2     2
       0.7087719
49 213         1     0        1          0      1   1     2     0     2
       0.8042705
50 212         0     0        1          0      1   0     0     1     1
       0.5285714
51 203         0     0        0          0      1   0     0     2     2
       0.2780269
52 162         1     1        1          1      0   0     0     1     1
       0.4452555
53 159         1     0        1          1      0   0     0     1     1
       0.3782051
54 131         0     0        1          0      0   0     0     2     2
       0.5209003
55 122         0     0        0          0      0   0     0     3     3
       0.2159763
56     priConsistency sqrtProduct                                        Cases
57 243      0.88579387  0.80093979 Belgium,Czechoslovakia,Netherlands,
       UnitedKingdom
```

```
240     0.63436123  0.44961744
    France,Sweden
213     0.71938776  0.57858232
    Finland,Ireland
212     0.22807018  0.12055138
                                                Estonia
203     0.00000000  0.00000000
    Italy,Romania
162     0.05000000  0.02226277
                                                Germany
159     0.03960396  0.01497842
                                                Austria
131     0.11309524  0.05891135
    Humgary,Poland
122     0.00000000  0.00000000                              Greece,
    Portugal,Spain

----------------
Explaining 2 configuration(s)

----------------
Prime implicant No. 1 with 2 implicant(s)

DEVELOPED*urban*LITERATE*industrial*STABLE +
DEVELOPED*URBAN*LITERATE*INDUSTRIAL*STABLE

Common configuration: DEVELOPED*LITERATE*STABLE

 # We can also use the dataset directly:
> reduce(LipsetFS,
+ outcome = "Survived",
+ conditions = c("DEVELOPED", "URBAN", "LITERATE",
+ "INDUSTRIAL", "STABLE"),
+ explain = "positive", remaind = "exclude", prepro = "fs",
+ consistency = 0.8)

Call:
reduce(x = LipsetFS, outcome = "Survived", conditions = c("DEVELOPED"
    ,     "URBAN", "LITERATE", "INDUSTRIAL", "STABLE"), explain = "
    positive",     remainders = "exclude", preprocess = "fs",
    consistency = 0.8)

truthTable with 9 configuration(s)

    DEVELOPED URBAN LITERATE INDUSTRIAL STABLE OUT freq1 freq0 NCase
       Consistency
243         1     1        1          1      1   1     4     0     4
      0.9042056
240         1     0        1          1      1   0     0     2     2
      0.7087719
213         1     0        1          0      1   1     2     0     2
      0.8042705
212         0     0        1          0      1   0     0     1     1
      0.5285714
203         0     0        0          0      1   0     0     2     2
      0.2780269
```

```
97  162          1     1         1           1       0   0      0      1      1
        0.4452555
98  159          1     0         1           1       0   0      0      1      1
        0.3782051
99  131          0     0         1           0       0   0      0      2      2
        0.5209003
100 122          0     0         0           0       0   0      0      3      3
        0.2159763
101      priConsistency sqrtProduct                                    Cases
102 243      0.88579387  0.80093979 Belgium,Czechoslovakia,Netherlands,
    UnitedKingdom
103 240      0.63436123  0.44961744
    France,Sweden
104 213      0.71938776  0.57858232
    Finland,Ireland
105 212      0.22807018  0.12055138
                                                 Estonia
106 203      0.00000000  0.00000000
    Italy,Romania
107 162      0.05000000  0.02226277
                                                 Germany
108 159      0.03960396  0.01497842
                                                 Austria
109 131      0.11309524  0.05891135
    Humgary,Poland
110 122      0.00000000  0.00000000                                Greece,
    Portugal,Spain
111
112 ----------------
113 Explaining 2 configuration(s)
114
115 ----------------
116 Prime implicant No. 1 with 2 implicant(s)
117
118 DEVELOPED*urban*LITERATE*industrial*STABLE +
119 DEVELOPED*URBAN*LITERATE*INDUSTRIAL*STABLE
120
121 Common configuration: DEVELOPED*LITERATE*STABLE
122
123  # To sort the table see the previous lines
```

If we use `QCA` we do the same as before, using `truthTable()`:

```
1  > TTfs <- truthTable(LipsetFS,
2  +        outcome = "Survived",
3  +        incl.cut1 = 0.8,
4  +        sort.by = "incl") # and we sort by "incl"
5  > TTfs
6
7   OUT: outcome value
8     n: number of cases in configuration
9  incl: sufficiency inclusion score
10  PRI: proportional reduction in inconsistency
11
12     DEVELOPED URBAN LITERATE INDUSTRIAL STABLE OUT n  incl  PRI
13 32       1       1       1          1       1     1  4  0.904 0.886
```

```
22      1        0       1          0         1      1  2  0.804 0.719
24      1        0       1          1         1      0  2  0.709 0.634
 6      0        0       1          0         1      0  1  0.529 0.228
 5      0        0       1          0         0      0  2  0.521 0.113
31      1        1       1          1         0      0  1  0.445 0.050
23      1        0       1          1         0      0  1  0.378 0.040
 2      0        0       0          0         1      0  2  0.278 0.000
 1      0        0       0          0         0      0  3  0.216 0.000


# With the following line we just get the truth table:
> TTfs$tt

   DEVELOPED URBAN LITERATE INDUSTRIAL STABLE OUT n                incl
                        PRI
32         1     1        1          1      1   1 4 0.904205607476635
      0.885793871866295
22         1     0        1          0      1   1 2 0.804270462633452
      0.719387755102041
24         1     0        1          1      1   0 2 0.708771929824561
      0.634361233480176
6          0     0        1          0      1   0 1 0.528571428571429
      0.228070175438597
5          0     0        1          0      0   0 2 0.520900321543408
      0.113095238095238
31         1     1        1          1      0   0 1 0.445255474452555
     0.0499999999999999
23         1     0        1          1      0   0 1 0.378205128205128
     0.0396039603960395
2          0     0        0          0      1   0 2 0.278026905829596
                       0
1          0     0        0          0      0   0 3 0.215976331360947
                       0

# And with the following we minimize the truth table:
> eqmcc(TTfs) # here we use as object the truth table

S1: DEVELOPED*urban*LITERATE*industrial*STABLE + DEVELOPED*URBAN*
    LITERATE*INDUSTRIAL*STABLE
```

Part II

Neat Formal Logic Meets Noisy Social Science Data

Chapter 5

Parameters of Fit: How To... Analyze Subset Relations With the Appropriate Software

Overview:

- Identification of contradictory truth table rows
- Identification of true logical contradictory cases
- Graphical display of set relations
- Empirical examples:
  - Vis (2009): used in book
  - "Selbst_1_data_matrix_with_contradictions.csv"

5.1 fsQCA 2.5

In order to identify necessary conditions, click [Analyze] [Necessary Conditions]. A new window opens and we specify the outcome (either the occurrence or non-occurrence of a set) in the appropriate box. In the window "Add condition" we select one condition at a time and send it into the field "Conditions" by pushing the arrow button. If we want to test the necessity of "functional equivalents," we first select two or more conditions and then send them into the field "Conditions." By clicking [Run], the output window opens and reports the consistency and coverage/relevance scores for each condition and functional equivalent specified. A second strategy for analyzing necessity in fsQCA 2.5 is to produce an XY plot by clicking [Graphs] [Fuzzy] [XY Plot] (see How To for chapter 3). The number on the lower right corner of the plot indicates consistency necessity and the number in the upper left corner cover-

age necessity. If a test of sufficiency is performed, then the upper left number indicates consistency of sufficiency and the lower right coverage of sufficiency.

In order to identify contradictory truth table rows, we follow the steps for representing a data matrix in a truth table as described above in chapter 4. In order to practice, open file "Selbst_1_data_matrix_with_contradictions.csv" [Analyze] [Crisp Sets] [Truth Table Algorithm]. Choose "y" as the outcome set, and "a", "b", "c" as the conditions, then click [Run]. A truth table appears with information on each row's consistency with the statement of being sufficient for outcome Y. Together with the information on how many cases are members in each row, we can calculate how many true logical contradictory cases there are in each row.[1] For example, row A∗B∗∼C contains 33 cases and its consistency is 0.939. This means that $33 - (33 * 0.939) = 2$ cases are inconsistent with the statement of sufficiency. That is, they are members of row A∗B∗∼C but not of Y. In order to identify which two cases these are, we need to turn to different software packages (see below).

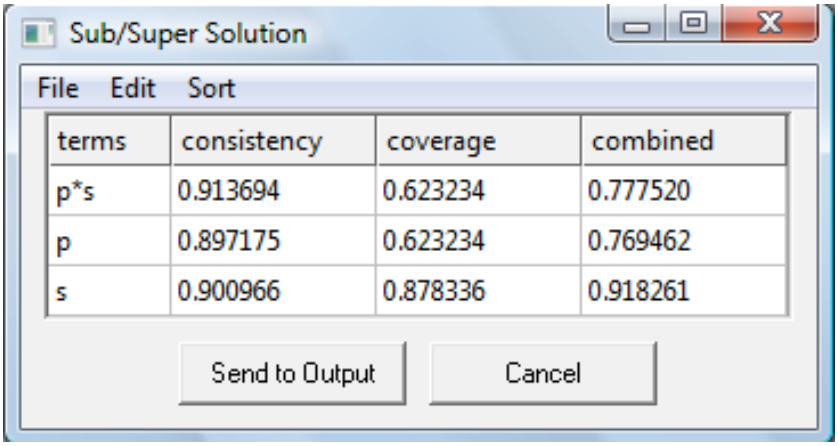
Rather than testing the sufficiency of all logically possible AND conjunctions – i.e. all truth table rows – fsQCA 2.5 offers an option to test the sufficiency of specific logical AND conjunctions and all their possible components. For illustration, open the data file "Vis_2009_fs.csv", click [Analyze] [Subset/Superset Analysis] and specify "u" as the outcome set, "p", "s", as the conditions, and click [Run]. A new window opens which reports the consistency and coverage of for expressions "P∗S", "P" and of "S". This function is useful when researchers have hunches about the sufficiency of a specific conjunction and want to know not only whether that hunch finds empirical support, but also if the conjunction could be shortened by dropping one or more constitutive elements without consistency becoming too low. See figure 9.1.

5.2 Tosmana 1.3.2

In Tosmana, no separate function for the analysis of necessity exists and no consistency and coverage scores are calculated. As described above in the How-To section for chapter 5, Tosmana reports which cases fall into each truth table row; however, it does not report each case's membership in the outcome set. Without this information, we cannot identify the true logical contradictory cases.

[1] Remember from chapter 5 of SMSS that with crisp sets all inconsistent cases are also true logical contradictory cases.

Figure 5.1: Sub/Super-set solution in fsQCA 2.5.



5.3 Stata

In Stata, the command `suffnec, altdisplay` is an easy way to calculate the consistency of a condition as either a necessary or a sufficient condition. The numbers in the lower right triangle of the table are the consistency of necessity scores and those in the upper left triangle the consistency sufficiency scores. Of course, in case a condition is identified as consistent enough as a necessary (or a sufficient) condition, the other value can be interpreted as its coverage score as a necessary (or sufficient) condition. The command `necessity` is less useful as it tests the necessity of logical AND configurations, which, for reasons explained in chapters 3ff. of SMSS, can only be necessary if each of the single conditions involved are necessary on their own.

5.4 R

To show how to calculate the parameters of fit concerning sufficiency and necessity in `R` we use the dataset called `LipsetFS` available in `SetMethods`:

```
1 > data(LipsetFS)
2 > head(LipsetFS)
3 .
4 .
5 .
```

`QCA3` has two functions to do get the parameters of fit with are `coverage()` and `consistency()`. The code below shows how to use them:

```
1 > coverage(LipsetFS$DEVELOPED,
```

```
+   LipsetFS$Survived, alternative = "greater")

[1] 0.775

> coverage(LipsetFS$DEVELOPED,
+   LipsetFS$Survived, alternative = "less")

[1] 0.831
```

The first entry in the functions is the condition, while the second entry is the outcome. The `alternative` option defines what parameter of fit is calculated: `greater` gives the coverage for necessary condition, while `less` gives the coverage for sufficient condition.

`QCA` has also a function to calculate consistency and coverage that is called `pof()`.[2] The following code shows how to use the function:

```
# Necessary condition
> pof(LipsetFS[, 2:6], LipsetFS,
+   outcome = "Survived", relation = "necessity")

                incl   PRI    cov.r
-----------------------------------
1  DEVELOPED    0.831  0.789  0.775
2  URBAN        0.539  0.474  0.771
3  LITERATE     0.991  0.990  0.643
4  INDUSTRIAL   0.669  0.617  0.684
5  STABLE       0.920  0.904  0.707
-----------------------------------

# Sufficient condition
> pof(LipsetFS[, 2:6], LipsetFS,
+   outcome = "Survived", relation = "sufficiency")

                incl   PRI    cov.r  cov.u
----------------------------------------
1  DEVELOPED    0.775  0.743  0.831  0.000
2  URBAN        0.771  0.720  0.539  0.000
3  LITERATE     0.643  0.587  0.991  0.053
4  INDUSTRIAL   0.684  0.648  0.669  0.004
5  STABLE       0.707  0.664  0.920  0.005
----------------------------------------
```

The first entry is the matrix including the conditions, the second entry is the dataset, the third entry is the outcome and the last entry defines the relation (sufficiency or necessity).

The parameters of fit can be calculated using the `QCAfit()` included in `SetMethods`. The following code shows how to use the function:

[2] `pof()` works if you input matrices, not single conditions. [check]

```
# Sufficiency
> QCAfit(LipsetFS$DEVELOPED, LipsetFS$Survived,
+ cond.lab = "Developed")

          Cons. Suf. Cov. Suf.   PRI PRODUCT
Developed       0.775     0.831 0.743   0.576

# Necessity
> QCAfit(LipsetFS$DEVELOPED, LipsetFS$Survived,
+ cond.lab = "Developed", necessity = TRUE)

          Cons. Nec. Cov. Nec.   RoN
Developed       0.831     0.775 0.811
```

The first entry is the condition, the second entry is the outcome, in the third entry we include the conditions' labels (if they need to be changed). To get the parameters of fit for necessity the `necessity` option must be set as `TRUE`. The default is `FALSE`.

QCAfit() also tests multiple conditions:

```
# Sufficiency
> QCAfit(LipsetFS[, 2:6], LipsetFS$Survived,
+ names(LipsetFS[, 2:6]))

            Cons. Suf. Cov. Suf.   PRI PRODUCT
DEVELOPED       0.775     0.831 0.743   0.576
URBAN           0.771     0.539 0.720   0.556
LITERATE        0.643     0.991 0.587   0.377
INDUSTRIAL      0.684     0.669 0.648   0.444
STABLE          0.707     0.920 0.664   0.470

# Necessity
> QCAfit(LipsetFS[, 2:6], LipsetFS$Survived,
+ names(LipsetFS[, 2:6]),
+ necessity = TRUE)

            Cons. Nec. Cov. Nec.   RoN
DEVELOPED       0.831     0.775 0.811
URBAN           0.539     0.771 0.899
LITERATE        0.991     0.643 0.509
INDUSTRIAL      0.669     0.684 0.786
STABLE          0.920     0.707 0.680
```

The entry `names(LipsetFS[, 2:6])` collects the conditions' labels from the data frame and uses it the output table.

If the parameters have to be computed for the negative outcome, the `negative` option have to be set as `TRUE`:

```
# Sufficiency
> QCAfit(LipsetFS[, 2:6], LipsetFS$Survived,
+ names(LipsetFS[, 2:6]), negation = FALSE)
.
.
.
```

If the parameters have to be computed for a negative condition just subtract 1 from the conditions:

```
# Sufficiency
> QCAfit(1 - LipsetFS[, 2:6], LipsetFS$Survived,
+ names(LipsetFS[, 2:6]))
.
.
.
```

As already shown we can visualize set-relations use `xy.plot()` from `SetMethods`. See figure 9.2.

```
> xy.plot(LipsetFS$DEVELOPED, LipsetFS$Survived,
+   labs = rownames(LipsetFS), srt = 45)
```

Configuration of conditions consistency and coverage are easily obtained getting, first, the scores of the configuration and, second, using the functions described above. We now use the `VisFS` dataset included in `SetMethods`. We load the data and get the minimum of two conditions. See the code below:

```
> data(VisFS)

# We get the minimum
> VisFS$p_s <- pmin(VisFS$p, VisFS$s)
> VisFS$p_s

 [1] 0.33 0.17 0.33 0.17 0.33 0.67 0.17 0.17 0.17 0.67 0.33 0.83 0.33
     0.33 0.33 0.60 0.17
[18] 0.60 0.33 0.17 0.33 0.33 0.33 0.17 0.33
```

The we use the "new" condition, which is a configuration of conditions, in the functions described above:

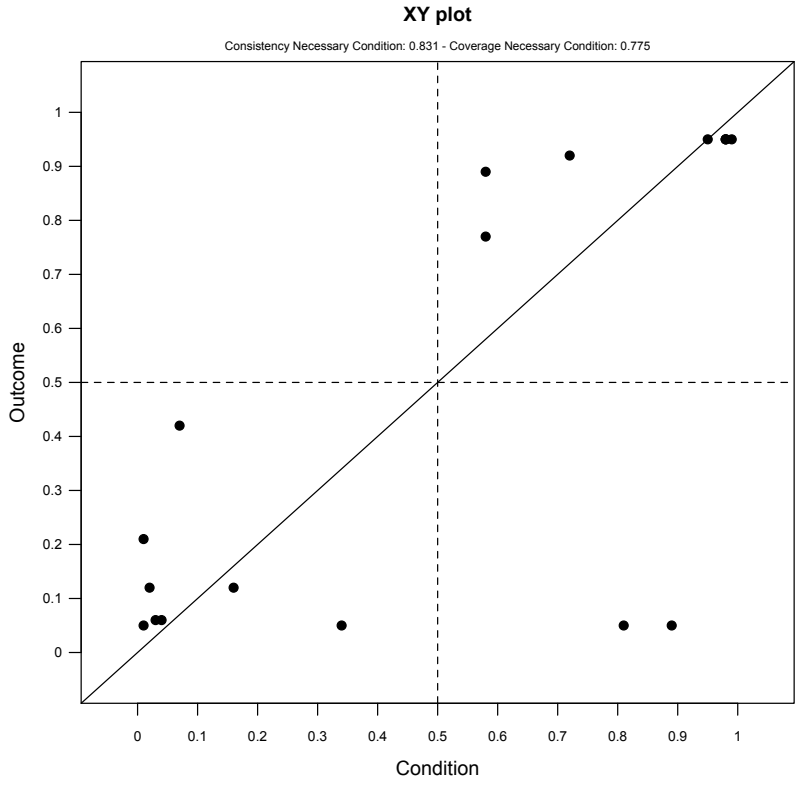
```
# Sufficiency
> QCAfit(VisFS$p_s, VisFS$ru, cond.lab = "ru")

   Cons. Suf. Cov. Suf.   PRI PRODUCT
ru       0.759      0.538 0.264   0.201

# Necessity
> QCAfit(VisFS$p_s, VisFS$ru, cond.lab = "ru", necessity = TRUE)

   Cons. Nec. Cov. Nec.   RoN
ru       0.538      0.759 0.886
```

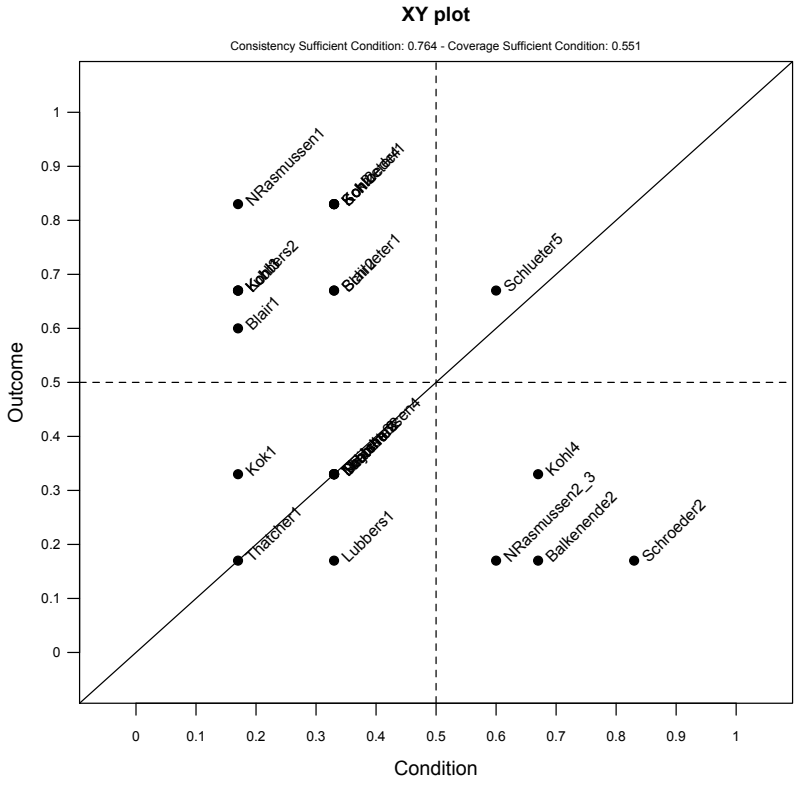
Figure 5.2: XY plot of Lipset data using "developed" as condition.



And we can use the configuration of condition to get an XY plot (see figure 5.3):

```
1 > xy.plot.lat(VisFS$p_s, VisFS$ru, labs = rownames(VisFS), srt = 45)
```

Figure 5.3: XY plot of Vis data using a configuration of conditions



Chapter 6

Limited Diversity and Logical Remainders: How To . . . Identify Assumptions Made on Remainders with the Appropriate Software

Overview:

- Identification of logical remainders rows
- Standard Analysis: conservative, most parsimonious, and intermediate solution term
- Identification of assumptions on remainders made for producing the most parsimonious and the intermediate solution terms
- Empirical examples:
  - Vis (2009): used in book

6.1 fsQCA 2.5

In fsQCA 2.5, the identification of logical remainders is easy. We simply convert our data set into a truth table using the Truth Table Algorithm command as described above (chapters 4 and 5). By default, the software displays all truth table rows, including those with not enough empirical evidence, a.k.a logical remainder rows. In order to get a summary of all remainder rows in the form of a Boolean expression, we can use the following trick. In the empty column for the outcome, assign the value 1 to all logical remainders and 0 to all other rows. Click [Run] and in the new window set "Positive cases true" and "Negative cases false" (see chapter 4), then click [Run]. The output window opens and displays the Boolean expression of the logical remainders (disregard the parameters of fit, since they do not have any meaning in this

specific procedure).

In order to practice, open file "Vis_2009_fs.csv". After producing the truth table following the steps outlined above, we see that there are three remainder rows: ∼PS∼R + P∼S∼R + P∼SR. In order to have the software summarize limited diversity in the form of a Boolean expression, code these three rows with 1 and all others with 0, click [Specify Analysis] [positive cases (1)] [True] and [RUN]. In the output window, we see that limited diversity in Vis' data is described by P∼S + ∼PS∼R.

In fsQCA 2.5, it is also easy to obtain the most parsimonious and the intermediate solution term. Again, we use the function of the (crisp or fuzzy) Truth Table Algorithm, specify the outcome and the conditions and click [Run]. In the window with the truth table, we, define rows as logical remainders according to the amount of empirical evidence they contain (as expressed in column "number") and classify all non-remainder rows as either sufficient for the outcome or not sufficient for the outcome according to their consistency value reported in column "raw consist".[1] After that, click [Standard Analyses]. This routine produces the three solution terms that should always be produced and reported: the conservative solution, the most parsimonious solution, and the intermediate solution. For the latter, we need to define the directional expectations. This is done in the new window that appears after having clicked [Standard Analyses]. For each single condition, we need to decide whether it contributes to the outcome when it is present, absent, or both. After having specified directional expectations for each condition, click [OK]. The three solution formulas appear in the output window. The fsQCA 2.5 software does not report which counterfactuals went into the most parsimonious and intermediate solution. In order to have this information, we have to use other software packages (see below).

6.2 Tosmana 1.3.2

Tosmana does produce truth tables (based on crisp but not fuzzy sets), but only without the logical remainder rows. Also, Tosmana produces the most parsimonious solution term but has no routine for specifying directional expectations and thus for producing the intermediate solution term. In order to produce the most parsimonious solution term, click [Analysis] [Start (MV)QCA]

[1] As explained above (chapter 5), we can use the shortcut [Edit] [Delete and code ...] for the task of classifying truth table rows as either sufficient for the outcome, not sufficient, or as logical remainders.

and specify the outcome, conditions, and case descriptor in the new window (see chapter 4). Set 'Outcome 1 Explain' and 'Outcome 0 Exclude'. In addition, set 'Remainders Include for Reduction' and select 'Compute Simplifying Assumptions'. With this setup, the software reports the simplifying assumptions that went into the most parsimonious solution term. After clicking [go], the output window appears and reports the most parsimonious solution term together with the simplifying assumptions that went into it.[2]

The Boolean Calculator tool in Tosmana is useful whenever more complex Boolean operations need to be performed. It can also be fruitfully used when trying to identify the assumptions that went into the intermediate solution term. For this, we need the intermediate solution term and the Boolean expression of all logical remainders. Both can be obtained with fsQCA 2.5 (see above). Then in Tosmana click [Analysis] [Boolean Calculator]. In the new window, create the Boolean expressions of the intermediate solution and for all logical remainder, respectively. This is done by first selecting a condition from window "Select Variable", then specify whether it appears in its presence or negation in the Boolean expression, then specify whether it is combined by logical AND or logical OR. Once ready with one Boolean expression, click [add] and the expression appears in the lower window. Once both the intermediate solution and the expression for all logical remainders have been specified, highlight both and click [Compute Intersection]. The expression that appears in the lower window describes all those logical remainders that are implied by the intermediate solution term, i.e., all easy counterfactuals.

6.3 Stata

In Stata, logical remainder rows can be identified by using the command `fuzzy settest( )`, which displays for all logically possible combinations of conditions (a.k.a. truth table rows), the number of cases that are members and whether it is a subset of the outcome or the non-occurrence of the outcome (see chapter 4). The command `truthtab` saves a truth table, which, however, does not contain enough information as to which rows are logical remainders. The most parsimonious and the conservative solution can be produced by using the commands `reduce` and `remainders(#)` combined with command `settest()`. There is no routine, however, for implementing directional expectations and

[2] Note the different notation form in Tosmana: A1 is equal to the condition A, whereas A0 denotes the complement ∼A. An expression such as A1B0C0 therefore has to be read as A∼B∼C.

thus producing the intermediate solution term. Stata also does not report the assumptions that went into the most parsimonious solution term.

6.4 R

We now show how to deal with logical remainders and limited diversity in `R`. As before we run the analysis with `QCA`. Let us load some data:

```
> data(VisFS)
> head(Vis)

                p    s   r    u
Lubbers1     0.33 0.83 1.0 0.83
Lubbers2     0.17 0.33 1.0 0.33
Lubbers3     0.33 0.67 0.6 0.67
Kok1         0.17 0.40 0.4 0.67
Kok2         0.33 0.33 0.4 0.17
Balkenende2 0.67 0.67 1.0 0.83

> data(LipsetFS)
> head(LipsetFS)
.
.
.

> data(LipsetCS)
> head(LipsetCS)
.
.
.
```

We get all the logical remainders in a truth table specifying the `complete` option as `TRUE` using the `truthTable()` function. See the code below:

```
> TTfs <- truthTable(LipsetFS, outcome = "Survived",
+     conditions = names(LipsetFS[,2:6]),
+   incl.cut1 = .8, complete = TRUE)
> TTfs

OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    DEVELOPED URBAN LITERATE INDUSTRIAL STABLE OUT n  incl  PRI
 1  0 0 0 0 0 0 3 0.216 0.000
 2  0 0 0 0 1 0 2 0.278 0.000
 3  0 0 0 1 0 ? 0 0.312 0.000
 4  0 0 0 1 1 ? 0 0.295 0.000
 5  0 0 1 0 0 0 2 0.521 0.113
.
.
.
```

```
20 .
```

Then, we can subset the table and just get the remainders:

```
> REMfs <- subset(TTfs$tt, OUT == "?")
> REMfs

DEVELOPED URBAN LITERATE INDUSTRIAL STABLE OUT n  incl  PRI
3   0 0 0 1 0 ? 0 0.3125  0
4   0 0 0 1 1 ? 0 0.2950  0
7   0 0 1 1 0 ? 0 0.4590  0
8   0 0 1 1 1 ? 0 0.4590  0
9   0 1 0 0 0 ? 0 0.6105  0
10  0 1 0 0 1 ? 0 0.6716  0
.
.
.
.
```

With `subset` we only select those rows (configurations of conditions) in which the outcome is not present in the data, and that are indicate by `?`. The same procedure is applicable to crisp sets:

```
> TTcs <- truthTable(LipsetCS, outcome = "SURVIVED",
+     conditions = names(LipsetCS[1:5]),
+     complete = TRUE)
> REMcs <- subset(TTcs$tt, OUT == "?")
```

We know show how to obtain the solutions using both `QCA` and `QCA3`. The complex solution is obtained excluding logical remainders, which is done by default by the function `eqmcc`.

```
> CompSolFS <- eqmcc(TTcs, details = TRUE)
> CompSolFS

n OUT = 1/0/C: 8/10/0
  Total      : 18

S1: DEVELOPED*LITERATE*INDUSTRIAL*GOVSTAB + DEVELOPED*urban*LITERATE*
    GOVSTAB

                                           incl   PRI    cov.r  cov.u
-------------------------------------------------------------------
1  DEVELOPED*LITERATE*INDUSTRIAL*GOVSTAB  1.000  1.000  0.750  0.500
2  DEVELOPED*urban*LITERATE*GOVSTAB       1.000  1.000  0.500  0.250
-------------------------------------------------------------------
   S1                                     1.000  1.000  1.000
```

The first entry in `eqmcc` is the trust table obtained with `truthTable()`. But we can also use a different function called `fs_truthTable()` and then `reduce`:

```
> TTfs2 <- fs_truthTable(LipsetFS, outcome = "Survived",
+     conditions = names(LipsetFS[2:6]))
> CompSolFS2 <- reduce(TTfs2)
> CompSolFS2

Call:
reduce(x = TTfs2)

truthTable with 9 configuration(s)

    DEVELOPED URBAN LITERATE INDUSTRIAL STABLE OUT freq1 freq0 NCase
        Consistency
243 1 1 1 1 1 1 4     0     4       0.904
240 1 0 1 1 1 0 0     2     2       0.709
213 1 0 1 0 1 1 2     0     2       0.804
212 0 0 1 0 1 0 0     1     1       0.529
203 0 0 0 0 1 0 0     2     2       0.278
162 1 1 1 1 0 0 0     1     1       0.445
159 1 0 1 1 0 0 0     1     1       0.378
131 0 0 1 0 0 0 0     2     2       0.521
122 0 0 0 0 0 0 0     3     3       0.216
    priConsistency sqrtProduct  Cases
243         0.8858      0.8009 Belgium,Czechoslovakia,Netherlands,
    UnitedKingdom
240         0.6344      0.4496  France,Sweden
213         0.7194      0.5786  Finland,Ireland
212         0.2281      0.1206  Estonia
203         0.0000      0.0000  Italy,Romania
162         0.0500      0.0223  Germany
159         0.0396      0.0150  Austria
131         0.1131      0.0589  Humgary,Poland
122         0.0000      0.0000  Greece,Portugal,Spain

----------------
Explaining 2 configuration(s)

----------------
Prime implicant No. 1 with 2 implicant(s)

DEVELOPED*urban*LITERATE*industrial*STABLE +
DEVELOPED*URBAN*LITERATE*INDUSTRIAL*STABLE

Common configuration: DEVELOPED*LITERATE*STABLE
```

The parsimonious solution is obtained including logical remainders with the option `include` using `eqmcc`:

```
> ParsSol <- eqmcc(TTfs, include = "?", details = TRUE)
> ParsSol

n OUT = 1/0/C: 6/12/0
  Total      : 18

S1: DEVELOPED*industrial + URBAN*STABLE

```

```
                          incl   PRI    cov.r  cov.u
-------------------------------------------------------
1  DEVELOPED*industrial   0.815  0.721  0.284  0.194
2  URBAN*STABLE           0.874  0.845  0.520  0.430
-------------------------------------------------------
   S1                     0.850  0.819  0.714
```

The intermediate solution is obtained specifying directional expectations. Be careful, the elements of the vector of directional expectations, `direxp`, must follow the order of column names:

```
> IntSol <- eqmcc(TTfs, include = "?",
+   direxp = c(1, 0, 0, 1, 1), details = TRUE)
> IntSol

n OUT = 1/0/C: 6/12/0
  Total      : 18

p.sol: DEVELOPED*industrial + URBAN*STABLE

S1:    DEVELOPED*urban*industrial*STABLE + DEVELOPED*URBAN*INDUSTRIAL
    *STABLE

                                       incl   PRI    cov.r  cov.u
-------------------------------------------------------------------
1  DEVELOPED*urban*industrial*STABLE   0.804  0.719  0.265  0.204
2  DEVELOPED*URBAN*INDUSTRIAL*STABLE   0.904  0.886  0.454  0.393
-------------------------------------------------------------------
   S1                                  0.870  0.843  0.658
```

`direxp` tells the function that the first condition must be present, the second and third must be absent, and the last two must be present.

What about crisp sets? It is exactly the same:

```
> CompSolCS <- eqmcc(TTcs, details = TRUE)
> CompSolCS

n OUT = 1/0/C: 8/10/0
  Total      : 18

S1: DEVELOPED*LITERATE*INDUSTRIAL*GOVSTAB + DEVELOPED*urban*LITERATE*
    GOVSTAB

                                            incl   PRI    cov.r  cov.u
-----------------------------------------------------------------------
1  DEVELOPED*LITERATE*INDUSTRIAL*GOVSTAB    1.000  1.000  0.750  0.500
2  DEVELOPED*urban*LITERATE*GOVSTAB         1.000  1.000  0.500  0.250
-----------------------------------------------------------------------
   S1                                       1.000  1.000  1.000


> ParsSolCS <- eqmcc(TTcs, include="?", details = TRUE)
> ParsSolCS

```

```
n OUT = 1/0/C: 8/10/0
  Total       : 18

S1: DEVELOPED*GOVSTAB

                       incl   PRI    cov.r  cov.u
-------------------------------------------------
1  DEVELOPED*GOVSTAB  1.000  1.000  1.000  1.000
-------------------------------------------------
   S1                 1.000  1.000  1.000
```

The `details` option sets as `TRUE` tells the function to provide all the possible information.

If you want or need to analyze the negative outcome set the option `neg.out` as `TRUE` and the function will automatically gives you the solution leading to the negative outcome.

If you just want check the remainder, follow this simple procedure:

1. get a truth table using `truthTable()`, with the `complete` option as `TRUE`
2. get the truth table, after creating a new object
3. select the logical remainders with `subset`
4. sort by inclusion

See the code below:

```
> TTfs <- truthTable(LipsetFS, outcome = "Survived",
+     conditions = names(LipsetFS[,2:6]),
+   incl.cut1 = .8, complete = TRUE)

> TTfsRem <- TTfs$tt

> TTfsRem <- subset(TTfsRem, OUT == "?")

> TTfsRem <- TTfsRem[order(TTfsRem$incl, decreasing = TRUE), ]

> TTfsRem

   DEVELOPED URBAN LITERATE INDUSTRIAL STABLE OUT n incl  PRI
25  1 1 0 0 0 ? 0 0.875 0
26  1 1 0 0 1 ? 0 0.868 0
27  1 1 0 1 0 ? 0 0.868 0
.
.
.
.
```

Chapter 7

The Truth Table Algorithm: How to . . . Perform the Truth Table Algorithm with the Appropriate Software

Overview:

- Implementation of Truth Table Algorithm
- Empirical examples:
  - Freitag & Schlicht (2009): used in book

7.1 fsQCA 2.5

Open file "FreitagSchlicht_2009_fs.csv" and click [Analyze] [Fuzzy Sets] [Truth Table Algorithm]. Then specify "socunequal" as the outcome, "lateeduc", "hdayschool", "earlytrack", "strongtripart" as conditions, "caseid" as the variable name column, then [Run].[1] In the new window that opens, first identify the logical remainders. As explained, these are the rows whose value in column "number" is smaller than the pre-established frequency threshold. In our example, the threshold is > 0. Point the cursor into the the first row that displays the value 0 in column "number, then [Edit] [Delete current to last row]. This visually eliminates the six logical remainder rows from the truth table. In order to get an overview of the consistency levels of all the non-remainder rows, move the cursor to the column "raw consist" and then click [Sort] [Descending]. Now, the highest raw consistency value (0.977 in our example) is at the top of the column. We notice a large gap in consistency between row 5 (0.861) and row 6 (0.789). This suggests a consistency threshold

[1] Notice that the label of the sets should be chosen such that it is clear what high membership in the set signifies.

of 0.86 or higher.[2] All rows above this threshold are coded 1 in the hitherto empty column for the outcome, those below are coded 0. This can be done by hand or by clicking [Edit] [Delete and code ...] and then inserting the value of 1 for the frequency threshold in the field "Delete rows with number less than" and the consistency threshold of 0.86 in the field "and set Y to 1 for rows with consist >". By clicking [OK], all rows receive a value (1, 0, or logical remainder) in the outcome column. In order to save this truth table, click [File] [Save As CSV File] "FreitagSchlicht_truthtab".

Click [Standard Analyses]. The software automatically produces the conservative (called "complex" by the software) and the most parsimonious solution term. Then a new window opens, asking you to specify for each single condition whether it should contribute to outcome "socunequal" when the condition is "Present", "Absent", or "Present or Absent". Clicking [Present or Absent] means that no directional expectation for a specific condition is formulate. We choose the following directional expectation "strongtripart (present)", "early-track (present)", "∼hdayschool (absent)", and "lateeduc (present)". This produces the following output:

```
1  **********************
2  *TRUTH TABLE ANALYSIS*
3  **********************
4
5  File: FreitagSchlicht_2009_fs.csv
6  Model: socunequal = f(lateeduc, hdayschool, earlytrack, strongtripart
       )
7
8   Rows:       10
9
10  Algorithm: Quine-McCluskey
11       True: 1
12
13 --- COMPLEX SOLUTION ---
14 frequency cutoff: 1.000
15 consistency cutoff: 0.860
16
17                                  raw        unique
18                                  coverage   coverage   consistency
19 lateeduc*earlytrack*~strongtripart  0.285 0.083 0.838
20 hdayschool*earlytrack*~strongtripart  0.355 0.153 0.882
21 lateeduc*hdayschool*strongtripart 0.448 0.305 0.944
22 solution coverage: 0.744
23 solution consistency: 0.892
24
```

[2]There is another gap between row 6 (0.789) and row 7 (0.703). As explained in chapter chapter 11.2 of SMSS, researchers are encouraged to test the robustness of their findings by performing the same analysis with a different raw consistency threshold, in the present example of 0.789 or higher, thus including row 6 into the logical minimization.

```
Cases with greater than 0.5 membership in term lateeduc*earlytrack*~
    strongtripart: SL (0.8,0.63),  NW (0.65,0.83)
Cases with greater than 0.5 membership in term hdayschool*earlytrack*
    ~strongtripart: SL (0.8,0.63), HH (0.79,0.65)
Cases with greater than 0.5 membership in term lateeduc*hdayschool*
    strongtripart: BY (0.92,1), RP (0.75,0.87), BW (0.71,0.84)

**********************
*TRUTH TABLE ANALYSIS*
**********************

File: FreitagSchlicht_2009_fs.csv
Model: socunequal = f(lateeduc, hdayschool, earlytrack, strongtripart
    )
Rows:      10

Algorithm: Quine-McCluskey
True: 1-L

--- PARSIMONIOUS SOLUTION ---
frequency cutoff: 1.000
consistency cutoff: 0.860

                                   raw       unique
                                coverage  coverage  consistency
lateeduc                           0.600 0.399 0.876
hdayschool*earlytrack*~strongtripart  0.355 0.153 0.882
solution coverage: 0.754
solution consistency: 0.880

Cases with greater than 0.5 membership in term lateeduc: BY (0.92,1),
     RP (0.83,0.87), SL (0.83,0.63), BW (0.71,0.84), NW (0.65,0.83)
Cases with greater than 0.5 membership in term hdayschool*earlytrack*
    ~strongtripart: SL (0.8,0.63), HH (0.79,0.65)

**********************
*TRUTH TABLE ANALYSIS*
**********************

File:  FreitagSchlicht_2009_fs.csv
Model: socunequal = f(strongtripart, earlytrack, hdayschool, lateeduc
    )
Rows:      10

Algorithm: Quine-McCluskey
True: 1
0 Matrix: 0L
Don't Care: -

--- INTERMEDIATE SOLUTION ---
frequency cutoff: 1.000
consistency cutoff: 0.860
Assumptions:
strongtripart (present)
earlytrack (present)
~hdayschool (absent)
lateeduc (present)
```

```
75
76                                    raw      unique
77                                  coverage   coverage   consistency
78 strongtripart*lateeduc                  0.467  0.083  0.925
79 earlytrack*lateeduc                     0.507  0.064  0.902
80 ~strongtripart*earlytrack*hdayschool    0.355  0.153  0.882
81 solution coverage: 0.744389
82 solution consistency: 0.892377
83
84
85 Cases with greater than 0.5 membership in term strongtripart*lateeduc
       : BY (0.92,1), RP (0.75,0.87), BW (0.71,0.84)
86 Cases with greater than 0.5 membership in term earlytrack*lateeduc:
       BY (0.92,1), SL (0.8,0.63), BW (0.71,0.84), NW (0.65,0.83)
87 Cases with greater than 0.5 membership in term ~strongtripart*
       earlytrack*hdayschool: SL (0.8,0.63), HH (0.79,0.65)
```

These results should be further examined. XY plots should be produced for single paths and the entire solution terms; uncovered cases and true logical contradictory cases should be identified; and the assumptions made for the most parsimonious and the intermediate solution should be made explicit. For this task, other software packages should be chosen.

7.2 Tosmana 1.3.2

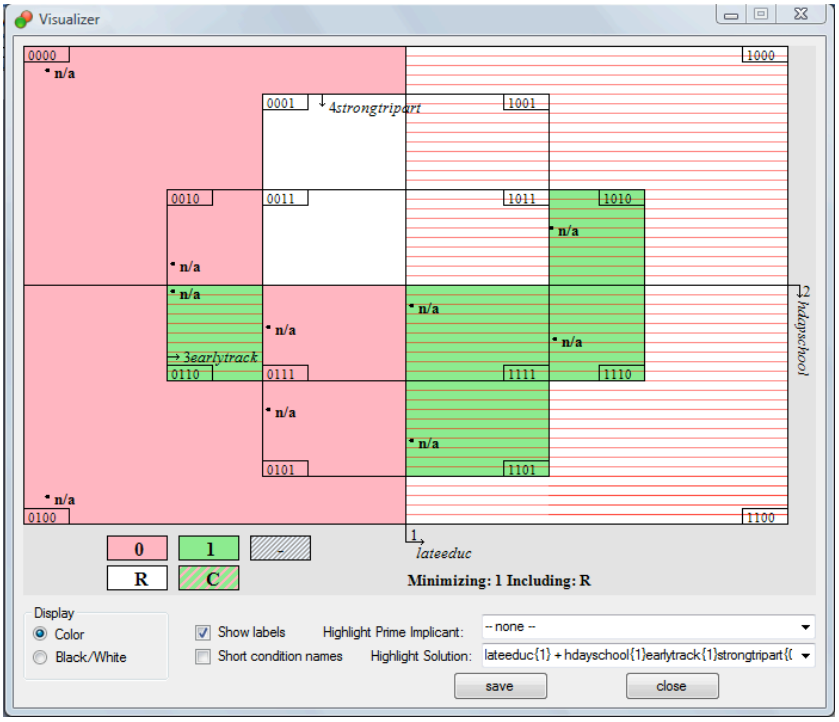
Tosmana 1.3.2 does not process fuzzy set data. Researchers can, however, use the software to identify the simplifying assumptions that went into the most parsimonious solution. In order to do so, save the truth table produced by fsQCA 2.5 (see above) and import it into Tosmana 1.3.2, clicking [File] [Import] [Excel or fsqca (csv file)] "FreitagSchlicht_truthtab". Click [Analysis] [Start (mvQCA)], specify "socunequal" as the outcome and the four other sets as condition. Tosmana 1.3.2 requests a case descriptor, so choose "number" as a fake descriptor. Then set "Outcome 1" to 'explain" and "Remainders" to "Include for Reduction". Also tick [Compute Simplifying Assumptions] and click [Go]. A new window opens that reports the same most parsimonious solution as fsQCA 2.5. In addition, the following simplifying assumptions are reported:

```
1 lateeduc{1}hdayschool{0}earlytrack{0}strongtripart{0} +
2 lateeduc{1}hdayschool{0}earlytrack{0}strongtripart{1} +
3 lateeduc{1}hdayschool{0}earlytrack{1}strongtripart{1} +
4 lateeduc{1}hdayschool{1}earlytrack{0}strongtripart{0}
```

It shows that four of the six logical remainders are assumed to be sufficient for "socunequal". The same information – the most parsimonious solution term and the simplifying assumptions that went into it – can be graphically

displayed by clicking [Visualize]. A new window opens that displays a Venn diagram (see figure 9.1). With four conditions, there are 16 different areas or truth table rows. Those colored in green are sufficient for outcome "socuneqal", those in red are not. The white areas are logical remainders. If we choose [Highlight solution], vertical lines are added to the graph, indicating which areas are covered by the most parsimonious solution. We see that not only all five green areas are covered, but also four out of the six white areas for logical remainders. These represent the simplifying assumptions that went into the most parsimonious solution.

Figure 7.1: Venn diagram produced by TOSMANA



Since Tosmana 1.3.2 does not have a routine for producing the intermediate solution term, this procedure of identifying simplifying assumption only works for the most parsimonious solution term.

7.3 Stata

Stata calculates the parameters of fit and can sort the data into truth tables. One cannot specify directional expectations, and there is therefore no convenient procedure for producing the intermediate solution term.

The following command line creates a truth table with information for each row's consistency level and the number of cases with membership higher than 0.5. In addition, the F distribution and the p values are reported, expressing whether each row's consistency threshold significantly differs from the pre-established threshold of 0.8:

```
fuzzy socunequal lateeduc hdayschool earlytrack ///
  strongtripart, label(Y A B C D) settest(yvv) conval(0.8)
```

7.4 R

We show how to the truth table algorithm in the previous chapter. However, we used R functions. In this section we show what is behind the truth table algorithm performing it step-by-step by "hand" with R. We use data from Pennings (2003):

```
> data(Pennings)
> head(Pennings)

      K    C    P    N    R
AU 0.33 0.33 0.00 0.33 0.83
AT 0.33 0.50 0.50 0.33 0.67
BD 0.33 0.17 0.17 1.00 0.67
BE 0.17 1.00 0.00 0.17 0.67
RB 0.00 0.00 1.00 0.67 0.17
BG 0.67 0.50 0.00 1.00 0.17

summary(Pennings) # check the data
.
.
.
```

There is a missing value so we drop it:

```
Pennings <- na.omit(Pennings)
```

We can construct the truth table by hand. First , find all the possible logical configurations: it is easy, they are 16 (4^2). We can do this by hand:

```
> LogConf <- data.frame(rbind(
+                             c(0, 0, 0, 0),
+                               c(0, 0, 0, 1),
+                               c(0, 0, 1, 0),
+                               c(0, 0, 1, 1),
+                               c(0, 1, 0, 0),
+                               c(0, 1, 0, 1),
+                               c(0, 1, 1, 0),
+                               c(0, 1, 1, 1),
+                               c(1, 0, 0, 0),
+                               c(1, 0, 0, 1),
```

```
+                               c(1, 0, 1, 0),
+                               c(1, 0, 1, 1),
+                               c(1, 1, 0, 0),
+                               c(1, 1, 0, 1),
+                               c(1, 1, 1, 0),
+                               c(1, 1, 1, 1)))
> colnames(LogConf) <- c("C", "P", "N", "R")  # Columns names
> rownames(LogConf) <- 1:16  # Row names
```

Below is the matrix of logical configurations:

```
> LogConf

   C P N R
1  0 0 0 0
2  0 0 0 1
3  0 0 1 0
4  0 0 1 1
5  0 1 0 0
6  0 1 0 1
7  0 1 1 0
8  0 1 1 1
9  1 0 0 0
10 1 0 0 1
11 1 0 1 0
12 1 0 1 1
13 1 1 0 0
14 1 1 0 1
15 1 1 1 0
16 1 1 1 1
```

We now calculate cases score memberships for each logical combination, we can follow the the matrix `LogConf` and we use `attach()` so we do not have to use the `$` sign and we include them in the dataset:

```
> attach(Pennings)
> Pennings$cpnr <- pmin(1-C, 1-P, 1-N, 1-R) # 1
> Pennings$cpnR <- pmin(1-C, 1-P, 1-N, R)   # 2
> Pennings$cpNr <- pmin(1-C, 1-P, N, 1-R)   # 3
> Pennings$cpNR <- pmin(1-C, 1-P, N, R)     # 4
> Pennings$cPnr <- pmin(1-C, P, 1-N, 1-R)   # 5
> Pennings$cPnR <- pmin(1-C, P, 1-N, R)     # 6
> Pennings$cPNr <- pmin(1-C, P, N, 1-R)     # 7
> Pennings$cPNR <- pmin(1-C, P, N, R)       # 8
> Pennings$Cpnr <- pmin(C, 1-P, 1-N, 1-R)   # 9
> Pennings$CpnR <- pmin(C, 1-P, 1-N, R)     # 10
> Pennings$CpNr <- pmin(C, 1-P, N, 1-R)     # 11
> Pennings$CpNR <- pmin(C, 1-P, N, R)       # 12
> Pennings$CPnr <- pmin(C, P, 1-N, 1-R)     # 13
> Pennings$CPnR <- pmin(C, P, 1-N, R)       # 14
> Pennings$CPNr <- pmin(C, P, N, 1- R)      # 15
> Pennings$CPNR <- pmin(C, P, N, R)         # 16
> detach(Pennings)
```

We extract the membership scores and create a new matrix:

```
> MemScores <- Pennings[, 6:ncol(Pennings)]
> head(MemScores)
.
.
.
```

and we exclude them from the data, dusk taking the first five columns:

```
> Keeps <- names(Pennings[, 1:5])
> Pennings <- Pennings[Keeps]
```

We look for the cases having a score higher than .5:

```
> TFtab <- MemScores > .5
> head(TFtab)
.
.
.
```

We see that the only case member of the combination `cpnr` is the United Kingdom, the cases member of the combination `cpnR` are Australia, Canada and Ireland, etc., etc. This procedure is long and we could miss some cases. So, we can make a loop and store the cases in a matrix called `Comb`:

```
> Comb <- matrix(NA, 16, 2)
> for(i in 1:16){
+ Comb[i, 1] <- length(rownames(MemScores[MemScores[, i] > .5,]))
+ Comb[i, 2] <- paste(rownames(MemScores[MemScores[, i] > .5,]),
+                     collapse="; ")
+ }
```

We get a matrix with cases that have membership $> .5$ in each logical configuration:

```
> Comb

      [,1] [,2]
 [1,] "1"  "GB"
 [2,] "2"  "AU; CA"
 [3,] "3"  "LT; MT; ES"
 [4,] "3"  "BD; GY; JA"
 [5,] "0"  ""
 [6,] "0"  ""
 [7,] "3"  "RB; NA; CL"
 [8,] "3"  "PK; PT; ZA"
 [9,] "3"  "LU; NZ; SE"
[10,] "3"  "BE; NL; NO"
[11,] "2"  "ET; HU"
[12,] "2"  "LV; RO"
[13,] "1"  "IS"
[14,] "1"  "FI"
[15,] "1"  "SK"
[16,] "0"  ""
```

Then we bind it with the matrix `LogConf` and we have the distribution of cases to ideal types (see chapter in the book):

```
> IdTypes <- data.frame(LogConf, data.frame(Comb))
> colnames(IdTypes) <- c("C", "P", "N", "R", "No.", "Cases")
> IdTypes

   C P N R No.       Cases
1  0 0 0 0   1          GB
2  0 0 0 1   2      AU; CA
3  0 0 1 0   3 LT; MT; ES
4  0 0 1 1   3 BD; GY; JA
5  0 1 0 0   0
6  0 1 0 1   0
7  0 1 1 0   3 RB; NA; CL
8  0 1 1 1   3 PK; PT; ZA
9  1 0 0 0   3 LU; NZ; SE
10 1 0 0 1   3 BE; NL; NO
11 1 0 1 0   2      ET; HU
12 1 0 1 1   2      LV; RO
13 1 1 0 0   1          IS
14 1 1 0 1   1          FI
15 1 1 1 0   1          SK
16 1 1 1 1   0
```

Now we can identify logical remainders (the rows with no cases) and perform the test of sufficiency of these rows. This means that for each combination we have to calculate the consistency. So, get `QCAfit()` or the functions in `QCA3` or in `QCA` we presented in the previous chapters. We get a vector of consistency scores for each logical combination:

```
> SufTest

 [1] 0.77 0.74 0.85 0.78 0.67 0.63 0.67 0.67 0.79 0.78 0.90 0.86 0.69
     0.63 0.78 0.78
```

We bind it to `IdTypes` and we make some substitutions:

```
> IdTypesCons <- data.frame(IdTypes, "Consistency" = SufTest)
> IdTypesCons

   C P N R No.       Cases Consistency
1  0 0 0 0   1          GB        0.77
2  0 0 0 1   2      AU; CA        0.74
3  0 0 1 0   3 LT; MT; ES        0.85
4  0 0 1 1   3 BD; GY; JA        0.78
5  0 1 0 0   0                   0.67
6  0 1 0 1   0                   0.63
7  0 1 1 0   3 RB; NA; CL        0.67
8  0 1 1 1   3 PK; PT; ZA        0.67
9  1 0 0 0   3 LU; NZ; SE        0.79
10 1 0 0 1   3 BE; NL; NO        0.78
11 1 0 1 0   2      ET; HU        0.90
```

```
16 12  1  0  1  1    2      LV; RO          0.86
17 13  1  1  0  0    1          IS          0.69
18 14  1  1  0  1    1          FI          0.63
19 15  1  1  1  0    1          SK          0.78
20 16  1  1  1  1    0                      0.78
```

Then we sort the table by consistency:

```
1  > IdTypesCons <- IdTypesCons[order(IdTypesCons[, "Consistency"],
2  +   decreasing = T),]
3  > IdTypesCons
4
5     C  P  N  R  No.       Cases  Consistency
6  11  1  0  1  0    2      ET; HU          0.90
7  12  1  0  1  1    2      LV; RO          0.86
8  3   0  0  1  0    3 LT; MT; ES          0.85
9  9   1  0  0  0    3 LU; NZ; SE          0.79
10 4   0  0  1  1    3 BD; GY; JA          0.78
11 10  1  0  0  1    3 BE; NL; NO          0.78
12 15  1  1  1  0    1          SK          0.78
13 16  1  1  1  1    0                      0.78
14 1   0  0  0  0    1          GB          0.77
15 2   0  0  0  1    2      AU; CA          0.74
16 13  1  1  0  0    1          IS          0.69
17 5   0  1  0  0    0                      0.67
18 7   0  1  1  0    3 RB; NA; CL          0.67
19 8   0  1  1  1    3 PK; PT; ZA          0.67
20 6   0  1  0  1    0                      0.63
21 14  1  1  0  1    1          FI          0.63
```

Decide the threshold for consistency (.79, for rounding reasons) and create a vector with values (1) indicating whether the row is sufficient for the outcome and we bind it to the table:

```
1  > Thres <- ifelse(IdTypesCons[,"Consistency"] >= .79, 1, 0)
2  > TT <- cbind(IdTypesCons, "Sufficiency ?" = Thres)
3  > TT[c(8, 12, 15), 8] <- c("?", "?", "?")
4  > TT
5
6     C  P  N  R  No.       Cases  Consistency  Sufficiency ?
7  11  1  0  1  0    2      ET; HU          0.90              1
8  12  1  0  1  1    2      LV; RO          0.86              1
9  3   0  0  1  0    3 LT; MT; ES          0.85              1
10 9   1  0  0  0    3 LU; NZ; SE          0.79              1
11 4   0  0  1  1    3 BD; GY; JA          0.78              0
12 10  1  0  0  1    3 BE; NL; NO          0.78              0
13 15  1  1  1  0    1          SK          0.78              0
14 16  1  1  1  1    0                      0.78              ?
15 1   0  0  0  0    1          GB          0.77              0
16 2   0  0  0  1    2      AU; CA          0.74              0
17 13  1  1  0  0    1          IS          0.69              0
18 5   0  1  0  0    0                      0.67              ?
19 7   0  1  1  0    3 RB; NA; CL          0.67              0
20 8   0  1  1  1    3 PK; PT; ZA          0.67              0
21 6   0  1  0  1    0                      0.63              ?
```

```
22 14 1 1 0 1   1          FI         0.63              0
```

By looking at `TT` we see that we have 5 rows that are sufficient for the outcome, therefore we have our complex solution: $CpNr + CpNR + cpNr + Cpnr + CpnR \rightarrow K$ However, it should be minimized. So we get: $Cp + pNr \rightarrow K$.

We know compute fit measures:

```
1  # Scores
2  > Cp <- pmin(Pennings$C, 1 - Pennings$P)
3  > Cp
4
5   [1] 0.33 0.50 0.17 1.00 0.00 0.50 0.17 0.17 0.83 0.67 0.33 0.50 0.50
        0.33 0.00 0.67 0.17
6  [18] 0.33 0.33 1.00 0.33 0.00 0.17 0.83 0.33 0.67 0.50 0.00 0.00 0.83
       0.67 0.67 0.00 0.50
7  [35] 0.17 0.67 0.17 0.50 0.00 0.33 0.00 0.67 0.50 0.00
8
9  > pNr <- pmin(1 - Pennings$P, Pennings$N, 1 - Pennings$R)
10 > pNr
11
12  [1] 0.17 0.33 0.33 0.17 0.00 0.83 0.17 0.17 0.17 0.67 0.33 0.50 0.17
        0.33 0.33 0.67 0.17
13 [18] 0.33 0.33 0.50 0.33 0.33 0.00 0.33 0.67 0.17 0.33 0.67 0.17 0.17
       0.17 0.17 0.00 0.50
14 [35] 0.33 0.00 0.17 0.50 0.33 0.67 0.00 0.17 0.00 0.00
15
16 > solution <- pmax(Cp, pNr)
17 > solution
18
19  [1] 0.33 0.50 0.33 1.00 0.00 0.83 0.17 0.17 0.83 0.67 0.33 0.50 0.50
        0.33 0.33 0.67 0.17
20 [18] 0.33 0.33 1.00 0.33 0.33 0.17 0.83 0.67 0.67 0.50 0.67 0.17 0.83
       0.67 0.67 0.00 0.50
21 [35] 0.33 0.67 0.17 0.50 0.33 0.67 0.00 0.67 0.50 0.00
22
23 # Raw coverages
24 > Cp_rc <- QCAfit(Cp, Pennings$K)[2]
25 > Cp_rc
26
27 [1] 0.697
28
29 > pNr_rc <- QCAfit(pNr, Pennings$K)[2]
30 > pNr_rc
31
32 [1] 0.588
33
34 # Consistencies
35 > Cp_co <- QCAfit(Cp, Pennings$K)[1]
36 > Cp_co
37
38 [1] 0.745
39
40 > pNr_co <- QCAfit(pNr, Pennings$K)[1]
41 > pNr_co
42
```

```
[1] 0.832

# Solution coverage and consistency
> sol_cov <- QCAfit(solution, Pennings$K)[2] # Solution coverage
> sol_cov

[1] 0.807

> sol_con <- QCAfit(solution, Pennings$K)[1] # Solution consist.
> sol_con

[1] 0.727

# Unique coverages
> pNr_u_cov <- sol_cov - Cp_rc
> pNr_u_cov

[1] 0.11

> Cp_u_cov <- sol_cov - pNr_rc
> Cp_u_cov

[1] 0.219
```

Check the cases covered by the solution. Before, we make a data frame with solution scores:

```
> SolScores <- data.frame(Cp, pNr, solution, "K" = Pennings$K)
> rownames(SolScores) <- rownames(Pennings)
> SolScores

     Cp  pNr solution    K
AU 0.33 0.17     0.33 0.33
AT 0.50 0.33     0.50 0.33
BD 0.17 0.33     0.33 0.33
BE 1.00 0.17     1.00 0.17
RB 0.00 0.00     0.00 0.00
BG 0.50 0.83     0.83 0.67
CA 0.17 0.17     0.17 0.33
CZ 0.17 0.17     0.17 0.50
DK 0.83 0.17     0.83 0.50
ET 0.67 0.67     0.67 0.67
FI 0.33 0.33     0.33 0.17
FR 0.50 0.50     0.50 0.00
DE 0.50 0.17     0.50 0.67
GR 0.33 0.33     0.33 0.50
GY 0.00 0.33     0.33 0.17
HU 0.67 0.67     0.67 1.00
IS 0.17 0.17     0.17 0.17
IN 0.33 0.33     0.33 0.33
IE 0.33 0.33     0.33 0.50
IL 1.00 0.50     1.00 0.83
IT 0.33 0.33     0.33 0.67
JA 0.00 0.33     0.33 0.33
JP 0.17 0.00     0.17 0.33
LV 0.83 0.33     0.83 0.83
```

```
LT 0.33 0.67      0.67 0.17
LU 0.67 0.17      0.67 0.50
MK 0.50 0.33      0.50 1.00
MT 0.00 0.67      0.67 0.33
NA 0.00 0.17      0.17 0.17
NL 0.83 0.17      0.83 0.50
NZ 0.67 0.17      0.67 0.17
NO 0.67 0.17      0.67 0.50
PK 0.00 0.00      0.00 0.00
PL 0.50 0.50      0.50 0.50
PT 0.17 0.33      0.33 0.50
RO 0.67 0.00      0.67 0.50
SK 0.17 0.17      0.17 0.67
SL 0.50 0.50      0.50 0.33
ZA 0.00 0.33      0.33 0.50
ES 0.33 0.67      0.67 0.50
CL 0.00 0.00      0.00 0.17
SE 0.67 0.17      0.67 0.50
TR 0.50 0.00      0.50 0.17
GB 0.00 0.00      0.00 0.17
```

Then we look for the cases that are members of the sets:

```
> CovCases <- NA
> for(i in 1:2){
+ CovCases[i] <- paste(rownames(SolScores[SolScores[, i] > .5, ]),
+           collapse = "; ")
+ }

> CovCases

[1] "BE; DK; ET; HU; IL; LV; LU; NL; NZ; NO; RO; SE"
[2] "BG; ET; HU; LT; MT; ES"
```

The first scalar indicates cases member of solution `Cp` the second scalar indicates cases member of solution `pNr`:

```
# Get uncovered cases
> UncovCases <- paste(rownames(SolScores[
+            SolScores$solution <= .5 & SolScores$K >= .5, ]),
+            collapse = "; ")

> UncovCases

[1] "CZ; DE; GR; IE; IT; MK; PL; PT; SK; ZA"
```

Now we build the output table:

```
> Output <- rbind(round(c(Cp_rc, pNr_rc), 3),
+          round(c(Cp_u_cov, pNr_u_cov), 3),
+          CovCases,
+          round(c(Cp_co, pNr_co), 3),
+          c(round(sol_cov, 3), ""),
+          c(round(sol_con, 3), ""),
+          c(UncovCases, ""))
```

```
8
9  > rownames(Output) <- c("Raw coverage", "Unique Coverage",
10 +     "Covered cases",  "Consistency", "Sol. coverege",
11 +     "Sol. consistency", "Uncovered cases")
12 > colnames(Output) <- c("Cp", "pNr")
13 > Output <- data.frame(Output)
14 > Output # the table
15
16                   Cp    pNr
17 Raw coverage       0.697 0.588
18 Unique Coverage    0.219 0.110
19 Covered cases      BE; DK; ET; HU; IL; LV; LU; NL; NZ; NO; RO;
20                   SE BG; ET; HU; LT; MT; ES
21 Consistency        0.745 0.832
22 Sol. coverege      0.807
23 Sol. consistency   0.727
24 Uncovered cases    CZ; DE; GR; IE; IT; MK; PL; PT; SK; ZA
```

We also make an xyplot. Before we apply some jittering so the cases slighly change position and the plot looks better:

```
1 > x <- jitter(Cp, 1.8)
2 > y <- jitter(Pennings$K, 1.7)
3 > xy.plot(x, y, labs = rownames(Pennings))
```

We get the following xyplot:

Figure 7.2: XY plot of the analysis of Pennings' data

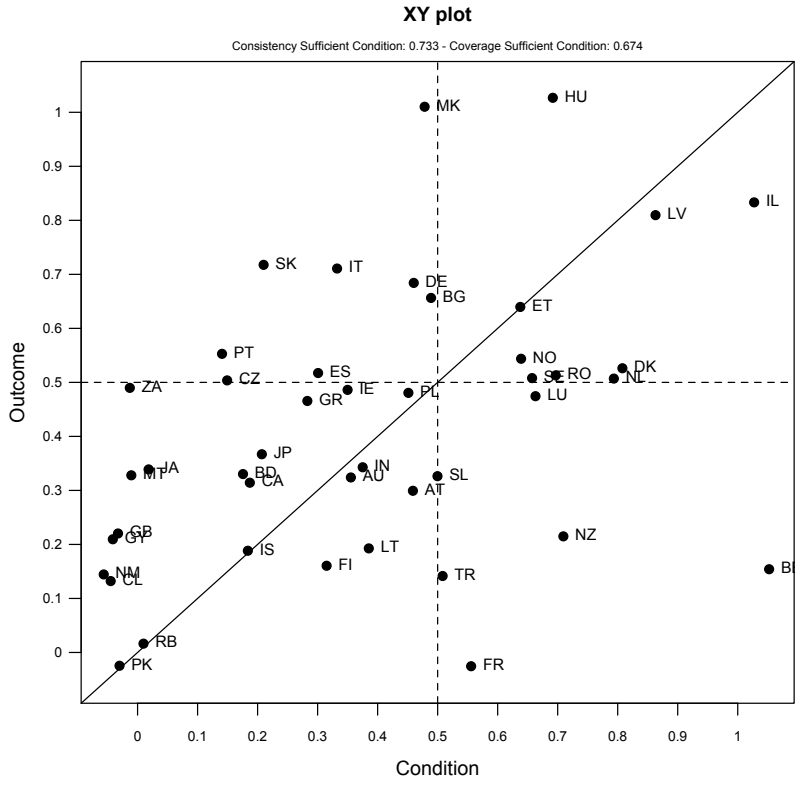


Table 7.1: Output table for the Pennings data.

	Cp	pNr
Raw coverage	0.697	0.588
Unique Coverage	0.219	0.11
Covered cases	BE; DK; ET; HU; IL; LV; LU; NL; NZ; NO; RO; SE	BG; ET; HU; LT; MT; ES
Consistency	0.745	0.832
Sol. coverege	0.807	
Sol. consistency	0.727	
Uncovered cases	CZ; DE; GR; IE; IT; MK; PL; PT; SK; ZA	

Part III

Potential Pitfalls and Suggestions for Solutions

Chapter 8

Potential Pitfalls in the Standard Analysis Procedure and Suggestions for Improvement: How to... Use the Software to Perform the Enhanced Standard Analysis and the Theory-Guided Analysis

Overview:

- Avoiding untenable assumptions
- Including good (non-simplifying) assumptions
- Empirical examples:
  - Vis (2009): used in book
  - Koenig-Archibugi (2004): used in book

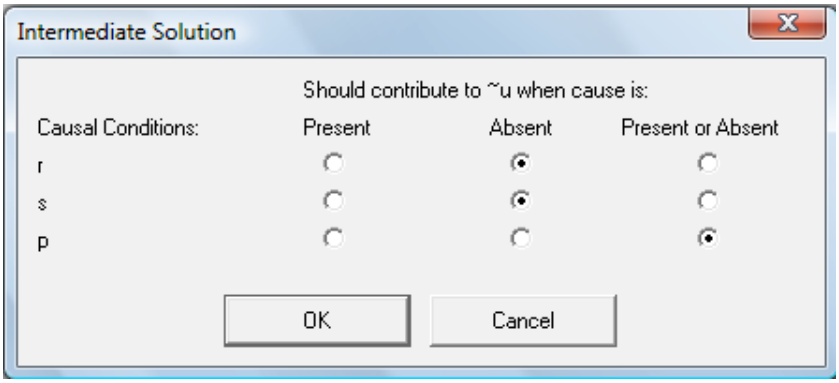
8.1 fsQCA 2.5

In order to avoid untenable assumptions, researchers must know which logical remainder rows should not be included into the logical minimization. This is a conceptual task. Once the non-eligible remainder rows are identified, it is easy in fsQCA 2.5 to bar them from being included into the logical minimization process.

For illustration, open file "Vis_2009_fs.csv" and produce the truth table for outcome "∼U" following the steps described above [Analyze] [Fuzzy Sets] [Truth Table Algorithm], then highlight "u" and click [Set Negated] and highlight "p", "s", "r" and click [Add], then [Run]. As explained in chapter 8 of SMSS, condition ∼P is necessary for outcome ∼U. Therefore, any logical remainder row containing condition P cannot be included into the logical minimization during the analysis of sufficiency. The two remainders rows –

P∼S∼R and P∼SR – must therefore be bared from the logical minimization process. This is done by coding them with the value 0 in column "∼u" of the truth table and to delete all remaining remainder rows (just one in this example) by moving the cursor into them and then [Edit] [Delete current row]. All remaining rows are coded 1 if their consistency is above 0.8 and 0 if below. Then click [Standard Analysis].

Figure 8.1: Standard analysis



In the window for directional expectations "S (absent)" and R (absent)" and [Ok]. In the output window you find the result of the enhanced Standard Analysis: all three solution formulas are identical: ∼S∼P + SPR → ∼U. This is because the only eligible remainder row– ∼PS∼R – does not contribute to parsimony and is therefore ruled out as both difficult or easy counterfactual.

TESA requires the inclusion of good but non-simplifying counterfactuals into the logical minimization procedure. In order to demonstrate how this is done, open file "Koenig_Archibugi_2004.csv" and produce the truth table following the procedure described with "supranat" as outcome and the remaining four sets as conditions. In order to allow only one counterfactual - namely, that countries with a European identity that do not expect conformity, who have a domestic multilevel governance structure and no high power capabilities are in favor of supranational foreign and security policy – we need to identify which logical remainder row in the truth table describes this hypothetical case. We can force the inclusion into the logical minimization of this row by inserting the value 1 into the oucome column "supranat" for it. All other remainder rows are deleted from the truth table using [Edit] [Delete current row]. All non-remainder rows receive the value of 1 if they reach the consistency threshold of 1 and the value of 0 otherwise. Click [Specify Analysis] and in the new window set [Positive cases (1)] to [True], [Negative Cases (0)] to [False], and

[Remainders] to [False] as well. Then click run. This produces the TESA solution term:

```
                                raw        unique
                             coverage    coverage   consistency
                            ----------  ----------  ----------
conform*region              0.494292    0.211187    1.000000
identmass*region*~capab     0.289954    0.006849    1.000000
solution coverage: 0.501141
solution consistency: 1.000000
```

Cases with greater than 0.5 membership in term conform*region: Germany (0.88,0.92), Belgium (0.8,1), Austria (0.7,0.92), Italy (0.6,0.92), Spain (0.59,0.83) Cases with greater than 0.5 membership in term identmass*region*∼capab: Belgium (0.63,1), Spain (0.55,0.83)

As explained in chapter 8 of SMSS, choosing entire truth table rows can be seen as an extreme form of conjunctural directional expectations. If researchers do not have expectations on entire truth table rows but only on the conjunction of, say, two conditions, the implementation of these assumptions is similar to the one just described for choosing entire truth table rows.

Continuing with the example from Koenig-Archibugi (2004), suppose, the expectation is that the joint presence of CONFORMITY and CAPABILITIES should produce the outcome. We now need to identify all those remainder rows that contain both conditions. There are two such logical remainder rows. Once identified, there outcome value must be coded as 1 and all other remainder rows be coded as 0. All non-remainder rows are coded as 1 if their consistency is 1, otherwise 0. Click [Specify Analysis] and in the new window set [Positive cases (1)] to [True] and [Negative Cases (0)] to [False], then [Run]. This produces the Theory-Guided Enhanced Solution term as reported in chapter 8 of SMSS.

```
                      raw        unique
                   coverage    coverage   consistency
                  ----------  ----------  ----------
conform*region    0.494292    0.184931    1.000000
conform*capab     0.334475    0.025114    1.000000
solution coverage: 0.519406
solution consistency: 1.000000
```

Cases with greater than 0.5 membership in term conform*region: Germany (0.88,0.92), Belgium (0.8,1), Austria (0.7,0.92), Italy (0.6,0.92), Spain (0.59,0.83) Cases with greater than 0.5 membership in term conform*capab: Germany (0.88,0.92), Italy (0.64,0.92)

8.2 Tosmana 1.3.2

Before untenable assumptions can be avoided or good assumptions be included, researcher must know which of the logical remainders would give rise to untenable assumptions and which of them can count as good assumptions. The most efficient way of performing this task is to represent all logical remainders, untenable assumptions, and good assumptions, respectively in the form of Boolean expressions. By calculating the intersection between all logical remainders and the untenable assumptions, we obtain a Boolean expression of all those remainders that must be excluded from the logical minimization. Likewise, by calculating the intersection between all logical remadiners and all good assumptions, we obtain a Boolean expression of all those truth table rows that must be included into the logical minimization.

We can use Tosmana's 1.3.2 Boolean calculator already presented in chapter 2 above for this task. Click [Analysis] [Boolean Calculator] and then create the logical expressions which have to be intersected. Calculate the intersection between two or more logical expressions by first highlighting the expression(s) and then clicking [Compute Intersection].

8.3 Stata

In Stata, the commands `remainders(#)` and `dnc` can be used in order to specify truth table rows about which the computer is allowed to make assumptions. By simply not listing remainder rows that would give rise to untenable assumptions, they are excluded from the logical minimization process.

8.4 R

In this section we see how to test unteneable assumptions using R. Get the data from Koenig-Archibugi (2004):

```
> data(KA
> head(KA)

         supranat identmass conform region capab
Austria      0.92      0.18    0.70    0.8  0.09
Belgium      1.00      0.63    0.97    0.8  0.14
Denmark      0.25      0.21    0.96    0.0  0.05
Finland      0.25      0.19    0.80    0.0  0.09
France       0.33      0.84    0.26    0.4  0.68
Germany      0.92      0.47    0.88    1.0  1.00
```

We look for logical remainders, we need to set `compete` as `TRUE` so we have all the combinations:

```
> TTka <- truthTable(data = KA, outcome = "supranat",
+          complete = T)
> TTka

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    IDENTMASS CONFORM REGION CAPAB OUT n  incl  PRI
 1      0        0      0      0    0  1  0.921 0.833
 2      0        0      0      1    0  1  0.547 0.277
 3      0        0      1      0    ?  0  0.916 0.846
 4      0        0      1      1    ?  0  0.909 0.792
 5      0        1      0      0    0  3  0.737 0.581
 6      0        1      0      1    ?  0  1.000 1.000
 7      0        1      1      0    1  1  1.000 1.000
 8      0        1      1      1    1  1  1.000 1.000
 9      1        0      0      0    ?  0  1.000 1.000
10      1        0      0      1    0  1  0.818 0.550
11      1        0      1      0    ?  0  1.000 1.000
12      1        0      1      1    ?  0  0.944 0.844
13      1        1      0      0    0  2  0.981 0.967
14      1        1      0      1    ?  0  1.000 1.000
15      1        1      1      0    1  2  1.000 1.000
16      1        1      1      1    1  1  1.000 1.000
```

We look for good counterfactual. We choose a specific configuration of condition, that is:

- eu identity, no conformity, multilevel governance and
- no power capabilities

We look for this configuration in the truth table and this is row (11) that is the 5th row. We can substitute the `?` sign indicating a logical remainder with a 1, so we force the software to think there is a case with this combination of condition also showing the presence of the outcome:

```
> TTka$tt[11, 5] <- 1
> TTka

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    IDENTMASS CONFORM REGION CAPAB OUT n  incl  PRI
 1      0        0      0      0    0  1  0.921 0.833
```

```
 2      0        0       0       1     0   1   0.547  0.277
 3      0        0       1       0     ?   0   0.916  0.846
 4      0        0       1       1     ?   0   0.909  0.792
 5      0        1       0       0     0   3   0.737  0.581
 6      0        1       0       1     ?   0   1.000  1.000
 7      0        1       1       0     1   1   1.000  1.000
 8      0        1       1       1     1   1   1.000  1.000
 9      1        0       0       0     ?   0   1.000  1.000
10      1        0       0       1     0   1   0.818  0.550
11      1        0       1       0     1   0   1.000  1.000
12      1        0       1       1     ?   0   0.944  0.844
13      1        1       0       0     0   2   0.981  0.967
14      1        1       0       1     ?   0   1.000  1.000
15      1        1       1       0     1   2   1.000  1.000
16      1        1       1       1     1   1   1.000  1.000
```

We minimize the truth table excluding logical remainders:

```
> Sol <- eqmcc(TTka, details = T)
> Sol

n OUT = 1/0/C: 5/8/0
  Total      : 13

S1: CONFORM*REGION + IDENTMASS*REGION*capab

                            incl   PRI    cov.r  cov.u
-----------------------------------------------------------
1  CONFORM*REGION           1.000  1.000  0.494  0.211
2  IDENTMASS*REGION*capab   1.000  1.000  0.290  0.007
-----------------------------------------------------------
   S1                       1.000  1.000  0.501
```

The same procedure applies to csQCA.

Chapter 9

Potential Pitfalls in the Analysis of Necessity and Sufficiency and Suggestions for Avoiding Them: How To... Avoid Wrong Inferences on Necessary Conditions and How To... Detect Skewed Set Membership Scores

Overview:

- Avoiding the appearance of false necessary conditions
- Avoiding the disappearance of necessary conditions
- Detecting skewed set membership scores
- PRI for sufficient condition
- Relevance parameter for necessary conditions
- Visual inspection for both
- Empirical examples:
  - Fake data Selbst: used in book
- For appearance of false necessary conditions
- For skewed membership in analysis of sufficiency
  - Schneider, Schulze-Bentrop, & Paunescu (2010): used in book
- For skewed membership in analysis of necessity

As explained in chapter 9 of SMSS, necessary conditions can disappear from the sufficiency solution term for two reasons. First, researchers make incoherent assumptions, i.e., assumptions on logical remainders that contradict the statement of necessity. In the How To section to chapter 8, we have explained how to bar specific remainders from the logical minimization process. The same commands and strategies equally apply to barring remainders that would produce incoherent assumptions. So here we do not demonstrate this remedy again. Second, necessary conditions might disappear if researchers include inconsistent truth table rows into the logical minimization.

9.1 fsQCA 2.5

In order to replicate the analysis on the disappearance of necessary conditions due to the inclusion of inconsistent truth table rows as discussed in chapter 9.1.1.2 in SMSS, load data file "Selbst_1_disappear_nec_incons.csv". First we test the necessity of condition ∼B for outcome Y. Click [Analyze] [Necessary Conditions] and specify "y" as [Outcome] and "∼b" from the drop-down menu under [Add Condition], then send it over by clicking [– >] and [Run]. In the output window we see that ∼B passes the conventional threshold for consistency and can therefore be considered as a necessary condition for Y.

```
1 Analysis of Necessary Conditions
2
3 Outcome variable: y
4
5 Conditions tested:
6                       Consistency          Coverage
7 ~b                    0.918367             0.978261
```

In order to perform the analysis of sufficiency, follow the procedure as described above: [Analyze] [Crisp Sets] [Truth Table Algorithm] "y" as [Outcome], "a", "b", "c" [Add]. As we can see from the truth table, there are no logical remainders and all but one row is either fully consistent or fully inconsistent with the statement of being sufficient for Y. Only row AB∼C shows a consistency value of 0.8. Let's impose a consistency threshold of 0.75, thus including row AB∼C into the logical minimization. Click [Specify Analysis] and set [Positive cases (1)] to [True] and [Negative cases (0)] to [False], then [Run]. This yields the following solution term:

```
1             raw        unique
2           coverage    coverage   consistency
3          ----------  ----------  ----------
4 a*~c     0.285714    0.285714    0.933333
5 ~b*c     0.714286    0.714286    1.000000
```

```
6 solution coverage: 1.000000
7 solution consistency: 0.980000
```

There are two sufficient paths but only one contains the necessary condition ∼B. As explained in chapter 9 of SMSS, this happens because by the inconsistent row AB∼C is combined with row A∼B∼C into A∼C. How can researchers avoid this pitfall. Apart from only accepting perfectly consistent set relations, which is not really a solution in applied set-theoretic research, researchers should check to which truth table rows those cases belong that contradict the statement of necessity. Those truth table rows should not be included into the logical minimization during the analysis of sufficiency.

As table 9.2 in chapter 9 of SMSS shows, row AB∼C contains five cases. None of them should be a member of outcome Y if ∼B was a fully consistent necessary condition. However, four cases are members of Y. In this example, all cases contradicting the statement of necessity are falling into the same truth table but this does not have to be like this. In other data sets, they could be located in different rows. It therefore is important to know the names of the cases contradicting the statement of necessity and to then locate them in the truth table.

For an illustration of the appearance of false necessary conditions, open file "Vis_2009_cs.csv", click [Analyze] [Crisp Sets] [Truth Table Algorithm], "u" as [Outcome], "p", "s', "r" [Add] as conditions, [Run], [Edit] [Delete andcode ...] [[Delete rows with number less than "1"] [and set u to 1 for rows with consist > "0.85"] [OK] [Specify Analysis] [Positive cases (1)] as [True], [Negative Cases (0)] as [False] and [Remainders] as [False] as well. This produces the conservative solution, which looks as follows:

```
1                raw        unique
2              coverage     coverage   consistency
3             ----------  ----------  ----------
4 p*s*~r      0.153846    0.153846    1.000000
5 ~p*s*r      0.461538    0.461538    0.857143
6 solution coverage: 0.615385
7 solution consistency: 0.888889
```

There are two sufficient paths. Factoring out condition S, which appears in both, we can rewrite the solution term as S (P∼R + ∼PR) → U. Since condition S appears in both paths, it might seem that Y only occurs in the presence of condition S and that therefore S is also necessary for U. it is not, though. To see this, click [Analyze] [Necessary Conditions] "u" as [Outcome], "s" [Add] [Run].

```
1 Analysis of Necessary Conditions
2
3 Outcome variable: u
4
5 Conditions tested:
6                       Consistency          Coverage
7 s                     0.769231             0.833333
```

The consistency score of S as a necessary condition for Y is very low (0.769). How can this pitfall be avoided? As explained in chapter 9 of SMSS, the most straightforward strategy is to simply perform a separate test of necessity and to do so prior to the analysis of sufficiency. Even if S is contained in each path, it would not be interpreted as a necessary condition.

Skewed set membership scores in either X and/or Y can lead to two analytic pitfalls. In the analysis of necessity, a trivial condition is declared as necessary. And in the analysis of sufficiency, a condition is declared sufficient both for outcome Y and ∼Y.

For the problem of trivial necessary conditions, in chapter 9 of SMSS, we offer a formula for calculating the relevance of a necessary condition, that is a condition that has already passed the threshold of consistency. Currently, none of the available software packages is calculating this relevance score by default. With the syntax-based programs (Stata and R) the formula can be implemented, though (see below). Researchers should, however, graphically display their claim of necessity. With fuzzy sets, the most useful way of doing this is via an XY plot. For illustration, load file "Samford_2010_only_nec_cond.csv". Click [Graphs] [Fuzzy] [XY Plot], then "y" on [Y Axis] and tick the box for [Negate], because it is the non-occurrence of rapid liberalization that we are interested in. on [X Axis] choose "horg", which is the union of the two sets H and G. After clicking [Plot], we see that the consistency with the statement of necessity of the distribution is 0.904 (the number in the lower right corner) and that its "coverage" score is rather high (0.875), thus signaling that it is a relevant necessary condition. However, as the graphical display shows, with a few exceptions, all cases are (almost) full members in condition H+G. The hogh coverage score is simply a the result of the fact that also many cases have high membership in outcome Y.

To illustrate the problem of simultaneous set relations in the analysis of sufficiency, open file "Vis_2009_fs.csv". Create the truth tables once for outcome U and once for outcome ∼U. Remember, for the latter you need to highlight "u" and click [Set Negated] in the "Select Variables" window. We see that truth table row PSR passes a consistency threshold of 0.8 as a sufficient condition

for both outcomes. In addition to the consistency score, we therefore should also take into account the PRI measure (see chapter 9 of SMSS), which is reported by the fsQCA 2.5 software.

Since row PSR has a low PRI score for outcome ∼U, we should not declare it as a sufficient condition for ∼U. In addition, researchers should check, which of the two sufficiency statements (PSR → U and PSR → ∼U) rests on true logical contradictory cases. Here, again, an XY plot is an efficient way of doing this. In fsQCA, 2.5, we must first create the logical AND combination PSR. Click [Variables] [Compute] [Target Variable] "psr" [Expression] "fuzzyand(p,s,r)", [Ok]. This adds a new column to the data called "psr" which contains the minimum set membership score across the three conditions P, S, and R. Then produce the XY plot with condition PSR and outcome U and ∼U, respectively, both times specfifying "government" as [Case ID Variable]. We see that there is one true logical contradictory case for outcome u and two for outcome ∼U, thus providing further arguments not to refute the claim that PSR → ∼U.

The parameters PRI and PRODUCT guard against the pitfall of simultaneous subset relation when membership in X is small. Click [Analyze] [Fuzzy Sets] [Truth Table Algorithm], then select the outcome and conditions and click [Run]. A new window with a truth table appears. The last three columns report, for each row, the consistency as a sufficient condition for the outcome, PRI, and PRODUCT. Only rows with high values in PRODUCT should be considered as sufficient conditions for the outcome.

9.2 Tosmana 1.3.2

Not available.

9.3 Stata

Regarding the problem of simultaneous subset relations, in Stata, the command `settest(yvn)` reports each truth table row's consistency as a sufficient condition for Y and for ∼Y, respectively. While mathematically not the same as the PRI parameter, this information is also useful for detecting simultaneous subset relations. Furthermore, the command `fzplot` (see chapter 3 above) produces XY plots that are useful in detecting simultaneous subset relations and true logical contradictory cases.

9.4 R

To show the disappearance of necessary condition to test the inclusion of inconsistent truth table rows. We use Selbst's data.

```
> data(Selbst)
> head(Selbst)

   A B C Y
A1 0 0 0 0
A2 0 0 1 1
A3 0 0 1 1
A4 0 0 1 1
A5 0 0 1 1
A6 0 0 1 1
```

We test the necessity of the condition ∼b for the outcome Y. We see that it is a necessary condition. This is just an alternative way of calculating the negation of a crisp set condition. It is exactly the same as `1 - condition`.

```
> Selbst$b <- ifelse(Selbst$B == 1, 0, 1)
> Selbst$b

 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0
[44] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
    1 1 1 1 1 1 1 1 1 1 1 1
[87] 1 1 1 1 1 1 0 0 0 0 0 0 0
```

We use `QCAfit()` with option `necessity` as `TRUE`:

```
> QCAfit(Selbst$b, Selbst$Y, cond.lab = "b",
+     necessity = TRUE)

  Cons. Nec. Cov. Nec.   RoN
b      0.918     0.978 0.981
```

Consistency is quite high, so it may be a necessary condition. We look at the truth table. We see that there are no logical remainders:

```
> TT <- truthTable(Selbst, outcome = "Y",
      conditions = c("A", "B", "C"), complete = T)

> TT

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

   A  B  C  OUT n  incl  PRI
1  0  0  0   0   1 0.000 0.000
2  0  0  1   1  20 1.000 1.000
3  0  1  0   0  30 0.000 0.000
4  0  1  1   0  15 0.000 0.000
5  1  0  0   1  10 1.000 1.000
6  1  0  1   1  15 1.000 1.000
7  1  1  0   0   5 0.800 0.800
8  1  1  1   0   2 0.000 0.000
```

So, we compute the solution setting the inclusion threshold to .7:

```
> Sol <- eqmcc(Selbst, outcome = "Y",
      conditions = c("A", "B", "C"), incl.cut1 = .7, details = T)
> Sol

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

   A  B  C  OUT n  incl  PRI
1  0  0  0   0   1 0.000 0.000
2  0  0  1   1  20 1.000 1.000
3  0  1  0   0  30 0.000 0.000
4  0  1  1   0  15 0.000 0.000
5  1  0  0   1  10 1.000 1.000
6  1  0  1   1  15 1.000 1.000
7  1  1  0   1   5 0.800 0.800
8  1  1  1   0   2 0.000 0.000

n OUT = 1/0/C: 50/48/0
  Total      : 98

S1: Ac + bC

```

```
       incl   PRI    cov.r  cov.u
---------------------------------
1  Ac  0.933  0.933  0.286  0.286
2  bC  1.000  1.000  0.714  0.714
---------------------------------
   S1  0.980  0.980  1.000
```

To show the appearance of false necessary conditions we use the Vis (2009) data:

```
> data(VisCS)
> head(Vis)

              P S R U
Lubbers I     0 1 1 1
Lubbers II    0 0 1 0
Lubbers III   0 1 1 1
Kok I         0 0 0 1
Kok II        0 0 0 0
Balkenende II 1 1 1 1
```

Then find the truth table and look for solution setting the consistency threshold to .85:

```
> TT <- truthTable(Vis, outcome = "U", complete = T)
> TT

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

   P  S  R  OUT n  incl  PRI
1  0  0  0   0  7  0.286 0.286
2  0  0  1   0  6  0.167 0.167
3  0  1  0   ?  0    -     -
4  0  1  1   0  7  0.857 0.857
5  1  0  0   ?  0    -     -
6  1  0  1   ?  0    -     -
7  1  1  0   1  2  1.000 1.000
8  1  1  1   0  3  0.667 0.667


> Sol <- eqmcc(Vis, outcome = "U", incl.cut1 = .85, details = T)
> Sol

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

   P  S  R  OUT n  incl  PRI
1  0  0  0   0  7  0.286 0.286
2  0  0  1   0  6  0.167 0.167
4  0  1  1   1  7  0.857 0.857
```

```
7  1  1  0   1  2  1.000 1.000
8  1  1  1   0  3  0.667 0.667

n OUT = 1/0/C: 9/16/0
  Total      : 25

S1: pSR + PSr

        incl   PRI    cov.r  cov.u
-----------------------------------
1  pSR  0.857  0.857  0.462  0.462
2  PSr  1.000  1.000  0.154  0.154
-----------------------------------
   S1   0.889  0.889  0.615
```

We see two sufficient paths and that S is in two solutions. We can factorize the formula using `factorize()`:

```
> FSol <- factorize(Sol)
> FSol

S: pSR + PSr

F1: S(pR + Pr)
```

It might be possible that S is a necessary condition as it appears in the two terms, so we test its necessity:

```
> QCAfit(Vis$S, Vis$U, cond.lab = "S",
+     necessity = TRUE)

  Cons. Nec. Cov. Nec.   RoN
S      0.769     0.833 0.867
```

However, S is not necessary. This can be avoided performing the tests before the analysis of sufficiency.

In order to show the problem of set memebership scores skeweness we use the Samford data:

```
> data(Samford)
> head(Samford)

              Y    G    H HorG
arg 73-74 0.01 1.00 0.28 1.00
arg 74-76 0.01 1.00 0.81 1.00
arg 76-83 1.00 0.26 0.00 0.26
arg 83-89 0.07 1.00 0.00 1.00
arg 89-95 0.17 0.00 0.00 0.00
arg 95-99 0.03 1.00 0.98 1.00
```

We can use a "relevance of necessity" test. To test the presence of an "actual" necessary condition we use the function `QCAfit()` with the option `necessity` is set as `TRUE`:

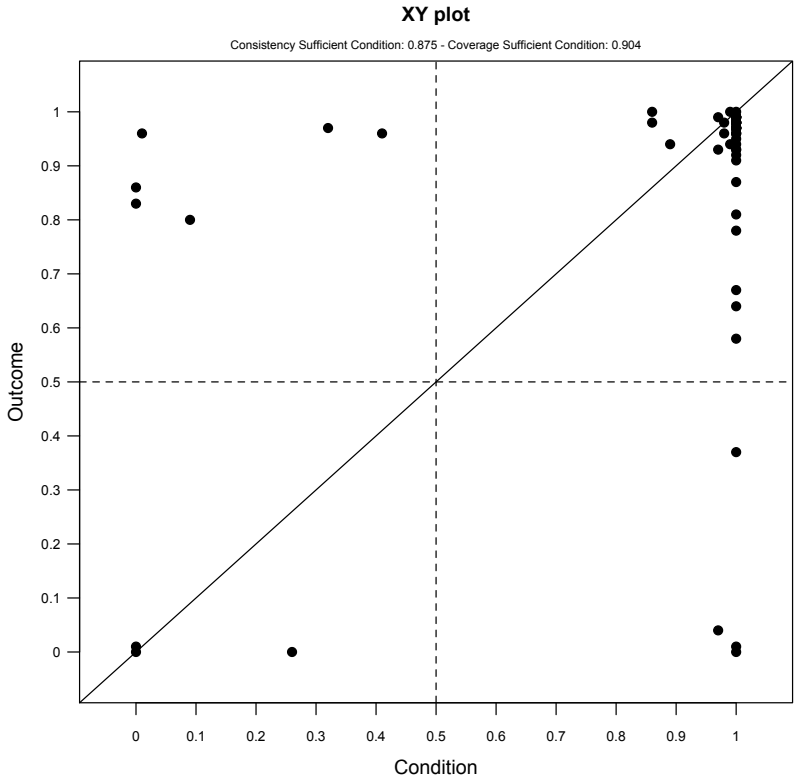
```
> QCAfit(Samford$HorG, 1 - Samford$Y, cond.lab = "HorG",
+     necessity = TRUE)

     Cons. Nec. Cov. Nec.   RoN
HorG       0.904      0.875 0.562
```

From the set-relational fit measures it may seem that the condition is necessary. But from the plot we notice that the condition might be a trivial one:

```
xy.plot(Samford$HorG, 1 - Samford$Y)
```

Figure 9.1: XY plot of a trivial condition



To show the problem of simultaneous set relations in the analysis of sufficiency we use the Vis (2009) data:

```
> data(VisFS)
> head(VisFS)

                p    s   r    u
Lubbers1     0.33 0.83 1.0 0.83
Lubbers2     0.17 0.33 1.0 0.33
Lubbers3     0.33 0.67 0.6 0.67
Kok1         0.17 0.40 0.4 0.67
Kok2         0.33 0.33 0.4 0.17
Balkenende2 0.67 0.67 1.0 0.83
```

We build the truth tables for positive and negative outcomes:

```
> TTpos <- truthTable(VisFS, outcome = "u",
+       sort.by = "incl", incl.cut1 = .8)
> TTpos

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

   P  S  R  OUT n  incl  PRI
4  0  1  1   1  7  0.918 0.782
7  1  1  0   1  2  0.911 0.773
8  1  1  1   1  3  0.911 0.647
2  0  0  1   0  6  0.719 0.242
1  0  0  0   0  7  0.642 0.307

> TTneg <- truthTable(VisFS, outcome = "u",
+       neg.out = TRUE, sort.by = "incl", incl.cut1 = .8)
> TTneg

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

   P  S  R  OUT n  incl  PRI
2  0  0  1   1  6  0.911 0.758
8  1  1  1   1  3  0.836 0.353
1  0  0  0   1  7  0.829 0.668
4  0  1  1   0  7  0.706 0.218
7  1  1  0   0  2  0.696 0.227
```

The row PSR has a consistency threshold of > 0.8 as a sufficient condition for both outcomes. So we take into account the PRI measure. It is possible to get PRI in two ways. Using `QCA`:

```
> TTpos

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
```

```
 PRI: proportional reduction in inconsistency

   P  S  R  OUT n  incl  PRI
4  0  1  1   1  7  0.918 0.782
7  1  1  0   1  2  0.911 0.773
8  1  1  1   1  3  0.911 0.647
2  0  0  1   0  6  0.719 0.242
1  0  0  0   0  7  0.642 0.307
```

Or it is found after minimization with the function `eqmcc()` set `details` as `TRUE`. This gives the PRI for each configuration in the solution formula:

```
> eqmcc(TTpos, details = TRUE)

n OUT = 1/0/C: 12/13/0
  Total      : 25

S1: PS + SR

       incl   PRI    cov.r  cov.u
---------------------------------
1  PS  0.914  0.736  0.623  0.151
2  SR  0.900  0.776  0.709  0.237
---------------------------------
   S1  0.904  0.794  0.860
```

Otherwise, you can use `QCAfit()` to get PRI:

```
> QCAfit(pmin(VisFS$p, VisFS$s, VisFS$r), 1 - VisFS$u, "psr")

    Cons. Suf. Cov. Suf.   PRI PRODUCT
psr       0.836       0.45 0.353   0.295
```

We can notice the the PRI score is low for for $\sim$u. This is confirmed by the xyplot (figure 9.2):

```
> xy.plot(pmin(VisFS$p, VisFS$s, VisFS$r), 1 - VisFS$u)
```

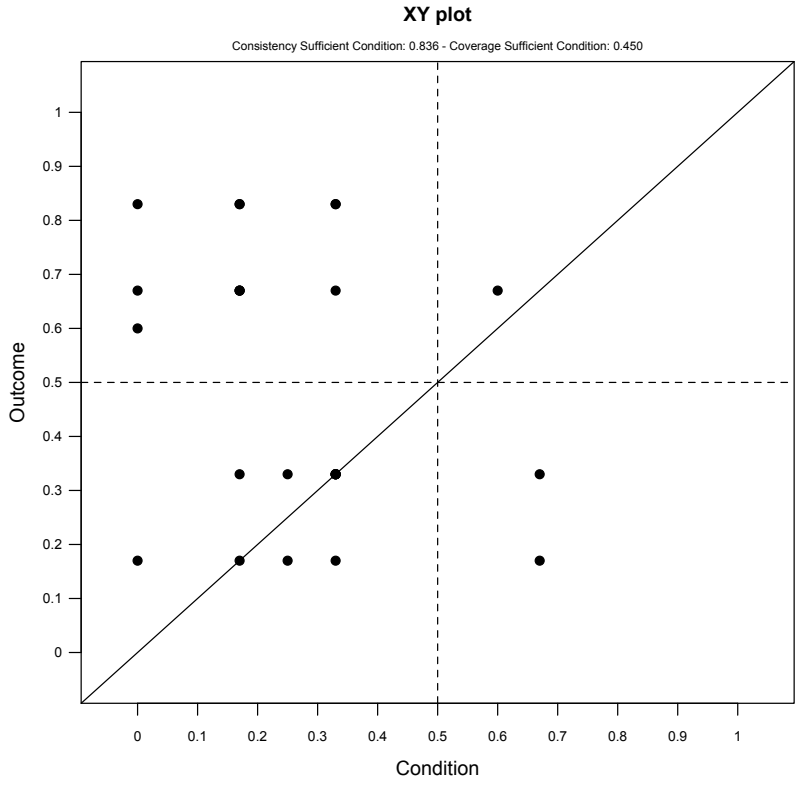
`QCAfit()` works for multiple conditions as well:

```
> QCAfit(cbind(VisFS$p, VisFS$s, VisFS$r), 1 - VisFS$u,
+     cond.lab = c("p", "s", "r"))

  Cons. Suf. Cov. Suf.   PRI PRODUCT
p       0.764      0.551 0.303   0.232
s       0.634      0.643 0.213   0.135
r       0.523      0.738 0.297   0.155
```

PRODUCT is another fit measure combining consistency and it can be obtained using `QCAfit()`:

Figure 9.2: XY plot of a low PRI score condition




```
> QCAfit(pmin(VisFS$p, VisFS$s, VisFS$r), VisFS$u,
+      cond.lab = "PSR")

    Cons. Suf. Cov. Suf.   PRI PRODUCT
PSR       0.911     0.472 0.647   0.589

> QCAfit(cbind(VisFS$p, VisFS$s, VisFS$r), 1 - VisFS$u,
+      cond.lab = c("P", "S", "R"))

  Cons. Suf. Cov. Suf.   PRI PRODUCT
P       0.764     0.551 0.303   0.232
S       0.634     0.643 0.213   0.135
R       0.523     0.738 0.297   0.155
```

Part IV

Variants of QCA as a Technique Meet QCA as an Approach

Chapter 10

Variants of QCA: How To... Perform Two-Step QCA, mvQCA and tQCA

Overview:

- Two-step QCA
- mvQCA
- tQCA
- Empirical examples:
  - Cronqvist & Berg-Schlosser (2009)
  - Caren & Panofsky (2005) and Ragin & Strand (2008): used in book

10.1 fsQCA 2.5

Not available.

10.2 Tosmana 1.3.2

Not available.

10.3 Stata

Not available.

10.4 R

10.4.1 Two-step QCA

Let us load the "Consolidation of Democracy" data (Schneider 2009) from `QCA3`:

```
data(CoD)
.
.
.
```

It is assumed that the output is the product of two sets of conditions: remote and proximate. So we perform two analyses (or steps). The first analyzes remote conditions. This produces a solution that outlines "outcome-enabling conditions". In a way, they are pre-requisites of the outcome.

The hypothesis of the first step is that: $econdev * eduhi * ethlihom * close * demex * nocom \rightarrow cod$. Therefore we look for the configurations of conditions sufficient for the outcome

```
# We label rownames
rownames(CoD) <- CoD$label

# Truth table
> TT <- truthTable(CoD, outcome = "cod",
+      conditions = c("econdev", "eduhi", "ethlihom",
+       "close", "demex", "nocom"),
+       sort.by = "incl", incl.cut1 = .8, show.cases = T)

> TT

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    ECONDEV EDUHI ETHLIHOM CLOSE DEMEX NOCOM OUT n  incl  PRI   cases
28      0      1       1     0     1     1    1  1  1.000 1.000 BR
50      1      1       0     0     0     1    1  1  1.000 1.000 AR
56      1      1       0     1     1     1    1  1  1.000 1.000 SP
58      1      1       1     0     0     1    1  1  1.000 1.000 MX
60      1      1       1     0     1     1    1  2  1.000 1.000 CH,UR
63      1      1       1     1     1     0    1  2  1.000 1.000 CR,SK
64      1      1       1     1     1     1    1  2  1.000 1.000 GR,PO
61      1      1       1     1     0     0    1  3  0.946 0.897 HU,PL
    ,SL
12      0      0       1     0     1     1    1  1  0.930 0.585 HO
20      0      1       0     0     1     1    1  1  0.921 0.683 PE
29      0      1       1     1     0     0    1  1  0.917 0.800 BU
13      0      0       1     1     0     0    1  2  0.915 0.721 RO,AL
25      0      1       1     0     0     0    1  1  0.899 0.667 MO
10      0      0       1     0     0     1    1  2  0.873 0.419 NI,PA
```

```
 2     0      0      0       0     0     1    1  2  0.862 0.185 BO,
     GUA
18     0      1      0       0     0     1    1  1  0.836 0.386 EC
 6     0      0      0       1     0     1    1  1  0.827 0.151 TU
53     1      1      0       1     0     0    1  1  0.813 0.441 EST
17     0      1      0       0     0     0    0  1  0.701 0.000 GE
21     0      1      0       1     0     0    0  5  0.680 0.355 BE,RU
    ,UK,LAT,LIT

# Parsimoniuos solution:
# econdev + ethlihom + nocom -> cod
> ParsSol <- eqmcc(TT, include = "?", details = T)
> ParsSol

n OUT = 1/0/C: 26/6/0
  Total      : 32

S1: ECONDEV + ETHLIHOM + NOCOM

             incl   PRI    cov.r  cov.u
----------------------------------------
1  ECONDEV   0.945  0.913  0.621  0.062
2  ETHLIHOM  0.807  0.721  0.689  0.124
3  NOCOM     0.605  0.442  0.563  0.082
----------------------------------------
   S1        0.668  0.543  0.910
```

The second step consists in running the analysis for the proximate factors in combination with one remote condition one by one. So we look for sufficient paths leading to CoD:

```
# Complex solutions

# with econdev
> TTecondev <- truthTable(CoD, outcome = "cod",
+   conditions = c("parlia", "efpahi", "decent", "econdev"),
+   sort.by = "incl", incl.cut1 = .8, show.cases = T)
> TTecondev

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    PARLIA EFPAHI DECENT ECONDEV OUT n  incl  PRI   cases
14     1      1      0       1     1  5  0.991 0.983 CR,HU,PO,SL,EST
10     1      0      0       1     1  1  0.986 0.967 GR
16     1      1      1       1     1  2  0.976 0.943 SK,SP
11     1      0      1       0     1  1  0.974 0.911 BU
13     1      1      0       0     1  2  0.966 0.880 RO,LAT
 3     0      0      1       0     1  2  0.930 0.748 NI,LIT
 6     0      1      0       1     1  2  0.927 0.800 CH,PL
 8     0      1      1       1     1  3  0.904 0.746 AR,MX,UR
 7     0      1      1       0     1  2  0.885 0.657 BR,RU
 9     1      0      0       0     1  3  0.816 0.461 MO,TU,AL
```

```
 1    0      0      0       0     1  2  0.808 0.385 HO,PA
 5    0      1      0       0     0  7  0.675 0.260 BE,BO,GE,GUA,PE,
     UK,EC


> ParsSolecondev <- eqmcc(TTecondev, details = T)
> ParsSolecondev

n OUT = 1/0/C: 25/7/0
  Total      : 32

S1: PARLIA*decent + efpahi*econdev + EFPAHI*ECONDEV + (parlia*DECENT*
    econdev)
S2: PARLIA*decent + efpahi*econdev + EFPAHI*ECONDEV + (parlia*EFPAHI*
    DECENT)

                                                  ------------------
                           incl   PRI    cov.r  cov.u  (S1)   (S2)
-------------------------------------------------------------------
1  efpahi*econdev          0.752  0.450  0.378  0.044  0.044  0.062
2  EFPAHI*ECONDEV          0.937  0.888  0.531  0.066  0.084  0.066
3  PARLIA*decent           0.907  0.840  0.436  0.055  0.055  0.055
-------------------------------------------------------------------
4  parlia*DECENT*econdev   0.872  0.642  0.392  0.004  0.050
5  parlia*EFPAHI*DECENT    0.847  0.630  0.381  0.007         0.054
-------------------------------------------------------------------
   S1                      0.824  0.707  0.819
   S2                      0.806  0.682  0.823


# with ethlihom
> TTethlihom <- truthTable(CoD, outcome = "cod",
+   conditions = c("parlia", "efpahi", "decent", "ethlihom"),
+   sort.by = "incl", incl.cut1 = .8, show.cases = T)
> TTethlihom

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    PARLIA EFPAHI DECENT ETHLIHOM OUT n  incl  PRI   cases
 3    0      0      1       0      1  1  0.979 0.918 LIT
12    1      0      1       1      1  1  0.974 0.931 BU
14    1      1      0       1      1  5  0.973 0.952 CR,HU,PO,RO,SL
15    1      1      1       0      1  1  0.963 0.887 SP
13    1      1      0       0      1  2  0.962 0.884 EST,LAT
16    1      1      1       1      1  1  0.962 0.925 SK
 6    0      1      0       1      1  2  0.957 0.888 CH,PL
 4    0      0      1       1      1  1  0.920 0.757 NI
 8    0      1      1       1      1  3  0.906 0.785 BR,MX,UR
 9    1      0      0       0      1  1  0.891 0.632 TU
10    1      0      0       1      1  3  0.873 0.730 GR,MO,AL
 7    0      1      1       0      1  2  0.838 0.492 AR,RU
 2    0      0      0       1      1  2  0.812 0.527 HO,PA
 5    0      1      0       0      0  7  0.616 0.122 BE,BO,GE,GUA,PE,
     UK,EC
```

```
78
79 > ParsSolethlihom <- eqmcc(TTethlihom, details = T)
80 > ParsSolethlihom
81
82 n OUT = 1/0/C: 25/7/0
83   Total      : 32
84
85 S1: ETHLIHOM + parlia*DECENT + PARLIA*decent + (EFPAHI*DECENT)
86 S2: ETHLIHOM + parlia*DECENT + PARLIA*decent + (PARLIA*EFPAHI)
87
88                                          --------------------
89                    incl   PRI    cov.r  cov.u  (S1)   (S2)
90 ----------------------------------------------------------
91 1  ETHLIHOM        0.807  0.721  0.689  0.103  0.124  0.103
92 2  parlia*DECENT   0.819  0.614  0.445  0.025  0.025  0.088
93 3  PARLIA*decent   0.907  0.840  0.436  0.000  0.023  0.000
94 ----------------------------------------------------------
95 4  EFPAHI*DECENT   0.873  0.740  0.570  0.000  0.016
96 5  PARLIA*EFPAHI   0.941  0.905  0.463  0.016         0.032
97 ----------------------------------------------------------
98    S1              0.787  0.682  0.904
99    S2              0.785  0.682  0.920
100
101 # with nocom
102 > TTnocom <- truthTable(CoD, outcome = "cod",
103 +   conditions = c("parlia", "efpahi", "decent", "nocom"),
104 +   sort.by = "incl", incl.cut1 = .8, show.cases = T)
105 > TTnocom
106
107  OUT: outcome value
108    n: number of cases in configuration
109 incl: sufficiency inclusion score
110  PRI: proportional reduction in inconsistency
111
112     PARLIA EFPAHI DECENT NOCOM OUT n  incl  PRI   cases
113  3     0      0      1      0    1  1  1.000 1.000 LIT
114 11     1      0      1      0    1  1  1.000 1.000 BU
115 13     1      1      0      0    1  6  1.000 1.000 CR,HU,RO,SL,EST,LAT
116 15     1      1      1      0    1  1  0.986 0.966 SK
117 16     1      1      1      1    1  1  0.967 0.919 SP
118 14     1      1      0      1    1  1  0.946 0.886 PO
119  8     0      1      1      1    1  4  0.940 0.844 AR,BR,MX,UR
120  4     0      0      1      1    1  1  0.913 0.719 NI
121 10     1      0      0      1    1  2  0.901 0.810 GR,TU
122  9     1      0      0      0    1  2  0.869 0.593 MO,AL
123  2     0      0      0      1    0  2  0.766 0.400 HO,PA
124  6     0      1      0      1    0  5  0.746 0.397 BO,CH,GUA,PE,EC
125  7     0      1      1      0    0  1  0.726 0.333 RU
126  5     0      1      0      0    0  4  0.648 0.295 BE,GE,PL,UK
127
128
129 > ParsSolnocom <- eqmcc(TTnocom, details = T)
130 > ParsSolnocom
131
132 n OUT = 1/0/C: 20/12/0
133   Total      : 32
134
```

```
S1: PARLIA*decent + (PARLIA*EFPAHI + efpahi*DECENT*nocom + parlia*
    DECENT*NOCOM)
S2: PARLIA*decent + (PARLIA*nocom + EFPAHI*DECENT*NOCOM + parlia*
    efpahi*DECENT)

                                                  ------------------
                           incl   PRI    cov.r  cov.u  (S1)   (S2)
-----------------------------------------------------------------------
1  PARLIA*decent           0.907  0.840  0.436  0.024  0.037  0.043
-----------------------------------------------------------------------
2  PARLIA*EFPAHI           0.941  0.905  0.463  0.007  0.078
3  PARLIA*nocom            0.904  0.847  0.436  0.020         0.099
4  efpahi*DECENT*nocom     1.000  1.000  0.158  0.000  0.014
5  EFPAHI*DECENT*NOCOM     0.943  0.862  0.375  0.000         0.106
6  parlia*DECENT*NOCOM     0.870  0.714  0.314  0.022  0.272
7  parlia*efpahi*DECENT    0.933  0.790  0.255  0.000         0.021
-----------------------------------------------------------------------
   S1                      0.878  0.797  0.810
   S2                      0.892  0.817  0.800
```

For further details see Schneider and Wagemann (2006).

10.4.2 mvQCA

It uses multi valued sets, meaning that the score do not range between 0 and 1, but are "categories". The analysis can be carried out in R as well and we use some fake data and, if you want to use real data you can use data included in the `QCA` package.

```
> data(DTmv)
> head(DTmv)

  Y A B C
1 1 1 1 0
2 1 0 0 2
3 0 0 0 2
4 1 0 0 2
5 0 2 0 0
6 0 0 1 1

# Truth table
> TTfake <- truthTable(DTmv, outcome = "Y", complete = T)
> TTfake <- truthTable(DTmv, outcome = "Y",
+     incl.cut1 = 0.8, complete = T)
> TTfake

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    A  B  C  OUT n  incl  PRI
 1  0  0  0   ?  0    -     -
 2  0  0  1   ?  0    -     -
```

```
 3  0  0  2   1  6  0.833 0.833
 4  0  1  0   ?  0    -     -
 5  0  1  1   0  6  0.667 0.667
 6  0  1  2   ?  0    -     -
 7  0  2  0   0  2  0.500 0.500
 8  0  2  1   ?  0    -     -
 9  0  2  2   ?  0    -     -
10  1  0  0   ?  0    -     -
11  1  0  1   1  2  1.000 1.000
12  1  0  2   ?  0    -     -
13  1  1  0   0  8  0.750 0.750
14  1  1  1   ?  0    -     -
15  1  1  2   ?  0    -     -
16  1  2  0   ?  0    -     -
17  1  2  1   ?  0    -     -
18  1  2  2   ?  0    -     -
19  2  0  0   0  1  0.000 0.000
20  2  0  1   ?  0    -     -
21  2  0  2   ?  0    -     -
22  2  1  0   ?  0    -     -
23  2  1  1   ?  0    -     -
24  2  1  2   ?  0    -     -
25  2  2  0   ?  0    -     -
26  2  2  1   ?  0    -     -
27  2  2  2   ?  0    -     -


# Complex solution
> CompSolFake <- eqmcc(DTmv, outcome = "Y", details = T)
> CompSolFake

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    A  B  C  OUT n  incl  PRI
 3  0  0  2   0  6  0.833 0.833
 5  0  1  1   0  6  0.667 0.667
 7  0  2  0   0  2  0.500 0.500
11  1  0  1   1  2  1.000 1.000
13  1  1  0   0  8  0.750 0.750
19  2  0  0   0  1  0.000 0.000

n OUT = 1/0/C: 2/23/0
  Total      : 25

S1: A{1}*B{0}*C{1}

                    incl   PRI    cov.r  cov.u
-------------------------------------------------
1  A{1}*B{0}*C{1}  1.000  1.000  0.111  0.111
-------------------------------------------------
   S1              1.000  1.000  0.111


NB: There is only one configuration to be explained. No minimization
```

```
    was performed.
83
84
85 > # Parsimonious solution
86 > ParsSolFake <- eqmcc(DTmv, outcome = "Y",
87 +   include = "?", details = T)
88 > ParsSolFake
89
90  OUT: outcome value
91    n: number of cases in configuration
92 incl: sufficiency inclusion score
93  PRI: proportional reduction in inconsistency
94
95     A  B  C  OUT n  incl  PRI
96  3  0  0  2   0  6  0.833 0.833
97  5  0  1  1   0  6  0.667 0.667
98  7  0  2  0   0  2  0.500 0.500
99 11  1  0  1   1  2  1.000 1.000
100 13  1  1  0   0  8  0.750 0.750
101 19  2  0  0   0  1  0.000 0.000
102
103 n OUT = 1/0/C: 2/23/0
104   Total      : 25
105
106 S1: A{1}*B{0}
107 S2: A{1}*C{1}
108 S3: B{0}*C{1}
109
110                                    ---------------------------
111               incl   PRI    cov.r  cov.u  (S1)   (S2)   (S3)
112 ------------------------------------------------------------------
113 1  A{1}*B{0}  1.000  1.000  0.111  0.000  0.111
114 2  A{1}*C{1}  1.000  1.000  0.111  0.000         0.111
115 3  B{0}*C{1}  1.000  1.000  0.111  0.000                0.111
116 ------------------------------------------------------------------
117    S1         1.000  1.000  0.111
118    S2         1.000  1.000  0.111
119    S3         1.000  1.000  0.111
```

We now use real data from Berg-Schlosser and Cronqvist (2005):

```
1 data(CronBerg)
2 # Vector of conditions
3 > cond <- names(CronBerg)[1:4]
4 > cond
5
6 [1] "GNP"   "URB"   "LIT"   "INDUS"
7
8 # Truth table
9 > TTCronBerg <- truthTable(CronBerg, outcome = "DEMOC", complete = T,
10 +     conditions = cond, incl.cut1 = 0.8)
11 > TTCronBerg
12
13  OUT: outcome value
14    n: number of cases in configuration
15 incl: sufficiency inclusion score
16  PRI: proportional reduction in inconsistency
```

```
17
18      GNP URB LIT INDUS OUT n   incl   PRI
19   1    0   0   0     0    0  5  0.000 0.000
20   2    0   0   0     1    ?  0    -      -
21   3    0   0   1     0    0  3  0.000 0.000
22   4    0   0   1     1    ?  0    -      -
23   5    0   1   0     0    ?  0    -      -
24   6    0   1   0     1    ?  0    -      -
25   7    0   1   1     0    ?  0    -      -
26   8    0   1   1     1    ?  0    -      -
27   9    1   0   0     0    ?  0    -      -
28  10    1   0   0     1    ?  0    -      -
29  11    1   0   1     0    1  2  1.000 1.000
30  12    1   0   1     1    0  1  0.000 0.000
31  13    1   1   0     0    ?  0    -      -
32  14    1   1   0     1    ?  0    -      -
33  15    1   1   1     0    ?  0    -      -
34  16    1   1   1     1    0  2  0.500 0.500
35  17    2   0   0     0    ?  0    -      -
36  18    2   0   0     1    ?  0    -      -
37  19    2   0   1     0    ?  0    -      -
38  20    2   0   1     1    1  2  1.000 1.000
39  21    2   1   0     0    ?  0    -      -
40  22    2   1   0     1    ?  0    -      -
41  23    2   1   1     0    ?  0    -      -
42  24    2   1   1     1    1  3  1.000 1.000
43
44
45  # Complex solution
46  > CompSolCronBerg <- eqmcc(TTCronBerg, details = TRUE)
47  > CompSolCronBerg
48
49  n OUT = 1/0/C: 7/11/0
50    Total      : 18
51
52  S1: GNP{2}*LIT{1}*INDUS{1} + GNP{1}*URB{0}*LIT{1}*INDUS{0}
53
54                                      incl   PRI    cov.r  cov.u
55  ------------------------------------------------------------
56  1  GNP{2}*LIT{1}*INDUS{1}           1.000  1.000  0.625  0.625
57  2  GNP{1}*URB{0}*LIT{1}*INDUS{0}    1.000  1.000  0.250  0.250
58  ------------------------------------------------------------
59     S1                               1.000  1.000  0.875
60
61
62  # Parsimonious solution
63  > ParsSolCronBerg <- eqmcc(TTCronBerg, include = "?",
64  +     details = T)
65  > ParsSolCronBerg
66
67  n OUT = 1/0/C: 7/11/0
68    Total      : 18
69
70  S1: GNP{2} + GNP{1}*INDUS{0}
71
72                        incl   PRI    cov.r  cov.u
73  ----------------------------------------------
```

```
1  GNP{2}           1.000  1.000  0.625  0.625
2  GNP{1}*INDUS{0}  1.000  1.000  0.250  0.250
-------------------------------------------------
   S1               1.000  1.000  0.875


# Intermediate solution
# GDP{1}, URB{0}, LIT{0}, INDUS{1} contribute to DEMOC
> IntSolCronBerg <- eqmcc(TTCronBerg, include = "?", direxp = c(1, 0,
    0, 1))
> IntSolCronBerg

p.sol: GNP{2} + GNP{1}*INDUS{0}

S1:    GNP{2}*INDUS{1} + GNP{1}*URB{0}*INDUS{0}
```

10.4.3 tQCA

To show tQCA we use the crisp set fake data:

```
> data(FakeCS)
> head(FakeCS)

  y j z w k
1 0 1 0 1 1
2 0 1 0 1 1
3 0 1 0 0 0
4 1 0 0 1 0
5 1 1 1 1 0
6 1 0 0 1 1
```

We assume `w` defines whether or not `j` was present before `z` so we can study time. We rename `w` to `w_before_z` and perform the analysis:

```
> names(FakeCS)[names(FakeCS)=="w"] <- "w_before_z"

# Truth table
> TT <- truthTable(FakeCS, outcome = "y", complete = TRUE)
> TT

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    J  Z  W_BEFORE_Z K  OUT n  incl  PRI
 1  0  0      0      0   ?  0    -     -
 2  0  0      0      1   1  3  1.000 1.000
 3  0  0      1      0   0  2  0.500 0.500
 4  0  0      1      1   1  1  1.000 1.000
 5  0  1      0      0   1  1  1.000 1.000
 6  0  1      0      1   0  2  0.500 0.500
 7  0  1      1      0   1  1  1.000 1.000
 8  0  1      1      1   0  1  0.000 0.000
```

```
 9  1  0       0       0    0   1   0.000  0.000
10  1  0       0       1    ?   0     -      -
11  1  0       1       0    ?   0     -      -
12  1  0       1       1    0   4   0.500  0.500
13  1  1       0       0    1   3   1.000  1.000
14  1  1       0       1    0   2   0.500  0.500
15  1  1       1       0    0   3   0.333  0.333
16  1  1       1       1    0   6   0.500  0.500


# Complex solution
> CompSol <- eqmcc(TT, details = TRUE)
> CompSol

n OUT = 1/0/C: 9/21/0
  Total      : 30

S1: j*z*K + j*Z*k + Z*w_before_z*k

                    incl   PRI    cov.r  cov.u
-----------------------------------------------
1  j*z*K            1.000  1.000  0.222  0.222
2  j*Z*k            1.000  1.000  0.111  0.056
3  Z*w_before_z*k   1.000  1.000  0.222  0.167
-----------------------------------------------
   S1               1.000  1.000  0.500
```

Now we look at some real data, as in the book (chapter 10). Data come from the article by Ragin and Stand (2008) (included in `QCA`):

```
> data(RagStr)
> head(RagStr)

# Complex solution
> CompSol <- eqmcc(RagStr, outcome = "REC", details = TRUE)
> CompSol

 OUT: outcome value
   n: number of cases in configuration
incl: sufficiency inclusion score
 PRI: proportional reduction in inconsistency

    P  E  A  S  EBA OUT n  incl  PRI
 3  0  0  0  0  dc   0  3  0.000 0.000
15  0  1  0  0  dc   0  1  0.000 0.000
22  0  1  1  1  0    1  1  1.000 1.000
27  1  0  0  0  dc   0  1  0.000 0.000
30  1  0  0  1  dc   0  3  0.000 0.000
36  1  0  1  1  dc   0  2  0.000 0.000
42  1  1  0  1  dc   1  1  1.000 1.000
44  1  1  1  0  1    1  2  1.000 1.000
46  1  1  1  1  0    1  1  1.000 1.000
47  1  1  1  1  1    1  2  1.000 1.000

n OUT = 1/0/C: 7/10/0
  Total      : 17
```

```
27
28 S1: P*E*S + E*A*S*eba + P*E*A*EBA
29
30                incl   PRI    cov.r  cov.u
31 ----------------------------------------
32 1  P*E*S       1.000  1.000  0.571  0.143
33 2  E*A*S*eba   1.000  1.000  0.571  0.143
34 3  P*E*A*EBA   1.000  1.000  0.714  0.286
35 ----------------------------------------
36    S1          1.000  1.000  1.000
```

The solution says that: $P*E*S+E*A*S*eba+P*E*A*EBA \rightarrow REC$ which means that union recognition happens when 1) the university is public, there are alliance and there is a strike threat; 2) there are alliances with elites, national union affiliation, strikes and the alliances were *not* present before national union affiliation; 3) the university is public, there are alliance, there is national union affiliation and the alliances *were* present before national union affiliation. Other solutions can be obtain with standard functions and options.