The Stata Journal (2008)
8, Number 1, pp. 79–104

fuzzy: A program for performing qualitative comparative analyses (QCA) in Stata

Kyle C. Longest
Department of Sociology
University of North Carolina at Chapel Hill
Chapel Hill, NC
klongest@email.unc.edu

Stephen Vaisey
Department of Sociology
University of North Carolina at Chapel Hill
Chapel Hill, NC

Abstract. Qualitative comparative analysis (QCA) is an increasingly popular analytic strategy, with applications to numerous empirical fields. This article briefly discusses the substantive motivation and technical details of QCA, as well as fuzzy-set QCA, followed by an in-depth discussion of how the new program `fuzzy` performs these techniques in Stata. An empirical example is presented that demonstrates the full suite of tools contained within `fuzzy`, including creating configurations, performing a series of statistical tests of the configurations, and reducing the identified configurations.

Keywords: st0140, fuzzy, cmvom, cnfgen, coincid, coverage, fzplot, mavmb, reduce, setgen, suffnec, truthtab, yavyb, yvn, yvo, yvv, yvy, qualitative comparative analysis, QCA, fuzzy sets, Boolean logic, Boolean data, postestimation command

1 Introduction

In recent years, researchers in a number of fields have begun using qualitative comparative analysis (QCA) or its fuzzy-set variant to analyze multivariate data (Ragin 1987, 2000). For examples, see Kalleberg and Vaisey (2005), Mahoney (2003), Roscigno and Hodson (2004), and Vaisey (2007). Rather than estimate the net effects of single variables, QCA employs Boolean logic to examine the relationship between an outcome and all binary combinations of multiple predictors. The advantage of QCA is that it allows the researcher to find distinct combinations of causal variables that, in turn, suggest different theoretical pathways to given outcomes. Although early versions of QCA (Ragin 1987) were criticized on the grounds that they were deterministic and that they bore little relation to commonly used variants of the general linear model, recent developments are now integrating QCA-based strategies with more formal statistical distributions and procedures (Ragin 2006; Smithson and Verkuilen 2006).

2 Why QCA?

Because QCA is still a relatively new strategy, we illustrate its utility by describing one substantive topic that might benefit from its application. The stress process model has inspired a wealth of research in the mental health field that has shown the deleterious consequences of an accumulation of stressors on increased negative health outcomes. Yet personal resources, active coping strategies, and social support can buffer these stressors, thereby reducing their harmful impact (for reviews, see Lin and Ensel [1989] and Thoits [1995]). The strength of this research notwithstanding, Thoits (1995) argued that there may be important pathways to negative health outcomes that have remained unseen because investigators typically apply linear estimation models to the stress process.

> Just as there may be different combinations of conditions across countries which lead to political revolution, there may be different configurations of factors across individuals which lead to heart attack or to the onset of major depression. . . . The assumption of one process for becoming depressed or ill and the concomitant use of the general linear model to test it requires us to reject or ignore other possible processes which are less frequently observed and do not manage to achieve statistical significance. (Thoits 1995, 68)

In this observation, Thoits was advocating the use of QCA in tests of the stress process model. That is, high levels of conjunction among stressors and buffering agents could define multiple routes to the same level of distress.

In addition to finding multiple paths to an outcome, QCA is especially appropriate for testing models, like that underlying stress theory, that involve a multitude of "interacting" factors. More precisely, QCA effectively addresses theoretical hypotheses that predict multiple variables will operate in tandem at specific levels (e.g., high stress events, low coping resources, and low social support) to produce particular outcomes (e.g., high distress). For example, stress theory predicts that numerous negative events should interact with chronic strain to produce high levels of distress, but the use of several active coping strategies should moderate this interaction, reducing the likelihood of elevated distress. Furthermore, high levels of mastery may enhance the buffering influence of active coping efforts, and high social support might further increase mastery's moderating influence on the interaction of active coping with stressors. This hypothesis modeled in a regression framework would involve a five-way interaction term, which would have to be interpreted along with all of its component interactions, an obviously difficult and inefficient task. QCA, on the other hand, explicitly and straightforwardly tests each possible combination of factors at specific levels with a given outcome. The results then can be interpreted more clearly, making QCA a potentially more effective analytic strategy for complex theoretical processes such as those posed by stress theory.

The stress process model is just one specific case that would benefit from the application of QCA.[1] QCA's utility as an analytic strategy stands to augment research in numerous fields. Greckhamer and his colleagues (2007) have recently argued for QCA's

1. Longest and Thoits (2007) have tested this model with QCA and found several intriguing results.

application in strategic management research because of its ability to analyze complex relationships between different industry- and corporate-level mechanisms in predicting business success. Similarly, researchers in epidemiology would benefit from QCA's capacity to capture holistically individuals' experience of risk and protective characteristics (Schuit et al. 2002). Finally, Shanahan et al. (2007) have demonstrated how QCA can be employed to examine the complex relationship between environmental and genetic factors leading to adult success.

We believe that part of the reason QCA has not been utilized more widely in empirical research across fields is because no software presently exists that easily combines QCA with conventional data management and statistical tests. The primary stand-alone program, fuzzy-set QCA, is able to compute logical truth tables for both fuzzy-set and dichotomous-set data, but it has no built-in capacity for probabilistically testing logical necessity and sufficiency, as advocated by Ragin (2006). There is also a program for the R statistics language that can perform logical reductions of Boolean data, but it lacks the ability to perform useful statistical tests. Further, there is no program in Stata that can perform the necessary analyses or reductions in fuzzy-set analyses.

In this paper, we present and outline `fuzzy`, a new Stata command we have developed that is capable of creating, testing, and performing logical reductions on both fuzzy and dichotomous (crisp) set-theoretic data. We will first outline some of the background of the technique and then provide a detailed explanation of the functionality of the command.

3 Statistical background

QCA evaluates the relationship between an outcome and all possible Boolean combinations of predictors. For example, given an outcome set `Y` and predictor sets `A` and `B`, QCA examines which combinations of `A` and `B` (i.e, $\texttt{A} \cdot \texttt{B}$, $\texttt{A} \cdot \texttt{b}$, $\texttt{a} \cdot \texttt{B}$, $\texttt{a} \cdot \texttt{b}$) are most likely to produce `Y`. In a QCA framework, the term "set" is used rather than "variable" to emphasize the idea that each variable has been transformed to represent the individual's level of membership in a given condition, for example, his or her level of membership in "heavy alcohol users". The combination of individual "sets"—for example, high depression and low self esteem—is then referred to as a "configuration". Sets are labeled, according to convention, with capital and lowercase letters. In the crisp-set case (i.e., all sets are dichotomous indicators) capital letters signify 1 (i.e., fully in `A`) and lowercase letters signify 0 (i.e., fully out of `A`). When using fuzzy sets, where set membership can take on any value between 0 and 1, uppercase simply means the level of set membership (e.g., value of `A`) and lowercase means 1 minus the set membership (e.g., $1-\texttt{A}$). The operator "$\cdot$" stands for the Boolean "and".

In the crisp-set case, the relationship between the predictors and the outcome can be evaluated using conditional probabilities—e.g., $\Pr(\texttt{Y} \mid \texttt{A} \cdot \texttt{B})$. In set-theoretic terms, higher conditional probabilities indicate greater empirical correspondence with the statement "$\texttt{A} \cdot \texttt{B}$ is a subset of `Y`", or, in logical terms, "if $\texttt{A} \cdot \texttt{B}$, then `Y`". Evaluating this logical or subset relationship becomes more problematic in the fuzzy-set case, however,

because unlike crisp sets, fuzzy sets can range between 0 (completely exclusive) and 1 (completely inclusive). Thus individuals can be more or less a member of a particular set (e.g., 0.33 would indicate something like "more out than in, but still somewhat in" the set, whereas 0.7 would signify something like "more in than out, but not entirely in" the set). Combining fuzzy sets into configurations is usually done using the minimum operator, so `A` · `B` = min(`A`,`B`), or `a` · `B` = min{(1−`A`) , `B`}.

The advantage of fuzzy sets over crisp sets is that we can transform our original measures without losing the variation associated with dichotomizing categorical or continuous measures. Using the minimum operation to calculate configuration membership more precisely defines the degree to which an individual experiences the combination of factors (i.e., individuals do not have to be completely in or completely out of every possible configuration). But this added nuance prohibits the use of a simple conditional probability to evaluate the degree of subsetness of each configuration in a given outcome. The most common approach to evaluating this relationship when using fuzzy sets is the inclusion ratio:

$$I_{XY} = \Sigma\min(x_i, y_i)/\Sigma x_i \tag{1}$$

where X signifies the predictor configuration (e.g., `A` · `B`), Y signifies the outcome set, x_i stands for each case's membership in the configuration X, and y_i stands for each case's membership in the set Y (see Ragin [2000, 2006] and Smithson and Verkuilen [2006] for discussions of other methods). As with conditional probabilities, the closer the value of I_{XY} to unity, the greater the consistency of the data with the assertion that X is a subset of Y or, in logical terms, with the statement "if X, then Y". (For this reason, this value is often referred to below as the "consistency score".)

Also there are a number of methods for deciding whether each configuration of predictors (X) should "count" as a (probabilistically) sufficient condition for Y. One way, advocated by Ragin (2000, 2006), is to determine a numeric benchmark (say, 0.8) and code all configurations, for which $I_{XY} > 0.8$, as sufficient. We take no position on any particular method here, and the `fuzzy` program allows multiple types of tests of probabilistic sufficiency. This flexibility is beneficial because the methods are still being refined and because several types of tests can support the robustness of claims of sufficiency.

The ultimate classification of some configurations as sufficient, however, is an important part of QCA. Once the sufficient configurations have been determined, one can use Boolean algebra to reduce the configurations into a more parsimonious solution. For example, if both `a` · `B` · `C` and `A` · `B` · `C` were coded as sufficient, this would reduce to `B` · `C`. This type of logical reduction can be extended to more complicated solution sets of configurations through the use of the Quine–McCluskey algorithm (see Ragin [1987]). In this way, one can obtain a logical description of the conditions sufficient to produce (probabilistically speaking) a particular outcome.

Finally, each final solution is evaluated with respect to its coverage of the outcome. Coverage is simply an indicator of how much of Y is covered by X; it is computed as follows:

$$C_{XY} = \Sigma\min(x_i, y_i)/\Sigma y_i$$

Although computationally similar, `coverage` addresses a different aspect than does the consistency score. Primarily, it helps to answer how much of the outcome is understood by taking into account the final solution set. For example, the set of skydiving parachute failures would be a near-perfect subset (i.e., high consistency) of the set of deaths, but this combination might not be very helpful (i.e., low coverage) in determining the most common or meaningful pathways to mortality in a given population.

The `fuzzy` program allows the user to create configurations from single sets coded as dichotomous or as fuzzy, to evaluate the sufficiency of these configurations statistically by using a variety of benchmarks, and to reduce the configurations determined sufficient to their common logical elements. The remainder of this paper describes the functionality of the program.

4 Creating, testing, and reducing sets

4.1 Syntax

`fuzzy` *varlist* [*if*] [*weight*] [`, label(`*capital_letter_list*`) keepsets drop`
`settest(`*testlist*`) group(`*varname*`) conval(`#`) sigonly slevel(`#`)`
`greater(col1|col2) common cluster(`*varname*`) necessity matx(`*matlist*`)`
`standardized altdisplay reduce remainders(`#`) dnc(`*configlist*`)`
`truthtab(`*filename*`) keepconfigs`]

where *testlist* is `yvn`, `yvo`, `yvv`, `yavyb`, `cmvom`, or `mavmb`; and *matlist* is `suffnec` or `coincid`.

`fweight`s, `iweight`s, and `pweight`s are allowed with `fuzzy`; see [U] **11.1.6 weight**.

The weights are applied with the `ratio` command, which is used to calculate the consistency score for each configuration, but are not used in any other parts of the routine (such as the creation of the `bestfit` variable).

4.2 Description

`fuzzy` is a suite of tools to perform QCA, as previously described. Without any options specified, it will create the `bestfit` variable that displays the number of cases that score greater than 0.500 on each configuration (which each case can only do for one configuration). The varlist should be treated similarly to other Stata commands, such that

the first variable listed is the outcome variable followed by the individual set variables. All variables entered must range from 0 to 1 (or be dichotomous coded 0/1).[2]

To create the configurations, `fuzzy` requires that all of the variables in the varlist be named with single, capital letters. If the user enters variables named as such, then `fuzzy`, without any options, will simply create the `bestfit` variable. But if any of the variables in the varlist are not named with single, capital letters, `fuzzy` will generate a copy variable of each variable in the varlist, naming them with single, capital letters, which are automatically deleted when the program is terminated. The user can control what letters are used to designate these copies by invoking the `label()` option, and when done the new variables will remain in the dataset, unless the `drop` option also is specified. Additionally, specifying `keepconfigs` will prevent the deletion of the generated configurations.

The primary advantages of `fuzzy`, compared to other QCA programs, lie in its options, most notably `settest()` and `reduce`. The `settest()` option defines the tests (each to be fully explained later) to be performed on the configurations' means or consistency scores. These tests help determine each configuration's degree of inclusion with the given outcome. `sigonly`, `slevel()`, `conval()`, and `greater()` all alter the configurations that are displayed by the given test and determine which configurations enter the reduction (if `reduce` is specified). At least one of these options (hereafter referred to as `settest()` options) must be specified for `reduce` to work, otherwise it would try to logically reduce every possible configuration (i.e., it would reduce to a logical contradiction). `common` can be used when multiple tests are run in a single call and will display (and send to `reduce`) only the configurations that pass all the tests designated.

`reduce` uses the Quine–McCluskey algorithm (Ragin 1987) to logically reduce the configurations specified by `settest()` and its options. Further, it displays the coverage statistics for each of the reduced configurations and for the total final solution set. If nonsingle, capital letter variables are entered in *varlist*, `reduce` will display its output using the original variable names.

Finally, the option `matx()` can be used to produce matrices of descriptive statistics. `matx(coincid)` will display the coincidence matrix for the varlist, and invoking the `standardized` option will produce the standardized scores. Similarly, `matx(suffnec)` produces a matrix showing the sufficiency and necessity scores for each variable entered into the varlist, and using the `altdisplay` option will produce a "flipped" matrix, placing the values where they are generally visualized graphically (i.e., sufficiency in the upper left and necessity in the lower right).

4.3 Options

`label(`*capital_letter_list*`)` allows the user to specify what sets should be named when used in creating and displaying the resulting configurations. If all variables in *varlist* are already named as single, capital letters, then there is no reason to specify this option (unless the user would like copies of the variables with new designations).

2. See `setgen` for a useful way to create such variables.

Further, this option is not required, but if any of the variables in *varlist* are not named with a single, capital letter and this option is not specified, then the generated copies of those variables will be named with a random single, capital letter, which will be used in displaying the configurations but dropped when `fuzzy` is terminated.

`keepsets` prevents the generic single, capital letter version of each variable from being deleted. This is only applicable when the `label()` option has not been specified and the original variables are not already named as single, capital letters.

`drop` automatically deletes any single, capital letter copies of variables entered in *varlist* when the program is terminated (only applicable if `label()` is also specified).

`settest(`*testlist*`)` defines which tests will be run and displayed:

`yvn` performs and displays the results (configuration; y consistency; n consistency; F distribution; p-value; number of best-fitting observations) of the test between each configuration's y consistency (inclusion in y) versus its n consistency (inclusion in not-y, or $1-y$). The test is performed using a Wald test (which uses an F distribution) comparing the consistency scores [i.e., equation (1)] derived using the `ratio` command; a similar test procedure is used for all the tests available in `settest()`. Thus a significant p-value means that the y consistency and the n consistency of a particular configuration are statistically different.

`yvo` performs and displays the results (configuration; y consistency; all other sets' y consistency; F distribution; p-value; number of best-fitting observations) of a test between each set's y consistency and the y consistency of all other configurations (excluding only the configuration in question). This "other" y consistency is calculated by taking the maximum value of every other configuration, excluding the configuration to be tested, and computing its inclusion in the outcome set. This test is generally applicable only if the configurations comprise binary crisp sets.

`yvv` performs a test of each configuration's y consistency versus a given numerical value (default is 0.800) and displays the configuration, y consistency, test value, F distribution, p-value, and number of best-fitting observations.

`yavyb` tests each configuration's y consistency for each of the subgroups defined in `group()`. The y consistency for each configuration is calculated separately for each subgroup, and then they are tested against each other. Hence this test will indicate, for each configuration, whether the y consistency of the first group defined in `group()` is significantly different from the y consistency of the second group. It displays the configuration; the first group's y consistency; the second group's y consistency; the F distribution; and the p-value. `group()` must be specified with `settest(yavyb)`.

`cmvom` operates similarly to `yvo`, but rather than using the configuration's consistency, it calculates and displays each configuration's weighted mean. The configuration's mean on the outcome is weighted by the membership in that configuration. This value is then tested against the mean as weighted by the maximum value of the other configurations. It displays each configuration's mean;

the "other" cumulative configuration's mean; the t value; the p-value; and the number of best-fitting observations.

`mavmb` operates similarly to `yavyb`, but it conducts the test using the configuration's weighted mean by each group specified by `group()`. The adjusted mean is calculated using the membership in that configuration for each subgroup. It then displays the weighted mean for the first group, weighted mean for the second group, t value, and p-value. `group()` must be specified with `settest(mavmb)`.

`group(`*varname*`)` defines the group variable used when invoking the `yavyb` or `mavmb` test. *varname* must be a 2-category variable.

`conval(#)` changes the value against which to test each configuration's y consistency if `settest(yvv)` is specified. The value can be any number between 0 and 1. The default is `conval(.800)`.

`sigonly` restricts the display of any tests specified in `settest()` to only those with significant p-values.

`slevel(#)` changes the significance level to be used in determining `sigonly`. The default is `slevel(.05)`.

`greater(col1|col2)` restricts the display of any test specified by `settest()` to only those in which the value in the designated column (of the output) is greater than the other column. The first column is always the consistency (or mean) for each configuration, while the second column is what that consistency is being tested against.

Note: `sigonly` and `greater(col1|col2)` can be used in conjunction. Thus, for example, if `sigonly` and `greater(col1)` were specified, any test designated in `settest()` would display only the results for those configurations that had a significant p-value on the test and that had a first column value greater than the second column.

`common` displays only the configurations that pass all of the tests and conditions specified by `settest()`. For example, if `settest(yvn yvv) sigonly common` was entered, `common` would display the configurations that had a significantly greater y consistency than n consistency and a y consistency significantly greater than 0.800.

Note: At least one of the `settest()` options must be used if `reduce` is invoked, in order to specify which configurations should be entered into the reduction. Further, if one (or more) of these restrictions is invoked along with `reduce`, only those configurations that are displayed will be entered into the reduction. Finally, if multiple tests are specified in `settest()`, without `common`, the last displayed configurations (regardless of the order in which the tests are specified) will be entered into the reduction (the order of the display will follow the order listed in the syntax above). Specifying `common` will send those common configurations to `reduce`.

`cluster(`*varname*`)` allows the standard errors produced by the `ratio` command when calculating consistencies to adjust for intragroup correlation.

`necessity` produces a table of each configuration's necessity value (similar to consistency except the denominator is the sum of the outcome instead of the sum of the configuration).

`matx(`*matlist*`)` defines which matrix to produce:

`suffnec` produces a sufficiency and necessity matrix for all the variables entered in *varlist*. Thus this option can be used to help determine the relationship between individual sets with each other and with the outcome. Sufficiency, in this case, is equivalent to computing individual set consistency scores.

`coincid` produces a coincidence matrix for all the variables entered in *varlist*. Again this is useful to help understand the relationship between the independent variables by using methods in line with fuzzy-set theory. Coincidence measures the amount of overlap or coincidence between the two sets or configurations (see Ragin [2006] for full details).

`standardized` alters the coincidence scores by taking into account the size of the sets. Coincidence is, in part, determined by the size of the sets because the larger they are the more room there is to overlap. This standardizing procedure divides the coincidence score by the total membership of the smaller set. The values in this matrix thus indicate the proportion of the total possible overlap between the sets.

`altdisplay` flips the `suffnec` matrix so that the sufficiency scores are in the upper left and the necessity scores are in the lower right (which is how these values are generally portrayed graphically).

`reduce` uses the elements passed by `settest()` to implement the Quine–McCluskey algorithm to produce a reduced final solution set and its accompanying coverage statistics. For example, if the input

```
. fuzzy Y A B C, settest(yvn) sigonly
```

displayed the configurations `ABc` and `ABC`, then `reduce` would produce `AB` and display this reduced configuration's coverage statistics. Again, when invoking `reduce`, it is also necessary to give some criteria to prevent the total possible configuration set from being entered into the reduction, which would reduce to a logical contradiction. Thus, when `reduce` is specified, `settest()` and `sigonly` or `greater()` must be specified.

`remainders(#)` runs the reduction twice: once with those configurations specified as remainders included in the reduction as "do-not-care" configurations, and once with them excluded. The # determines that any configuration with fewer or equal to # best-fitting observations are treated as remainders. This is highly advisable when the configuration contains many sets or when the sample size is small.

`dnc(`*configlist*`)` specifies configurations as "do-not-care" configurations. The entered configurations are treated as do-not-care configurations regardless of whether they pass any specified tests. These configurations are used in the first step of the reduction but not in the second step (see Ragin [2000] for a full explanation of do-not-care configurations).

`truthtab(`*filename*`)` outsheets (and replaces) a file containing the resulting truth table. If no options are specified, it will outsheet the entire truth table; otherwise, it operates similarly to `reduce` in how it defines which configurations are included. The *filename* should end in the desired output type (`.dta`, `.csv`, etc.).

`keepconfigs` prevents the generated configuration variables from being deleted when `fuzzy` is terminated.

4.4 Saved results

Macros

`r(y)`	outcome variable
`r(sets)`	total possible configuration set
`r(start)`	independent variable sets
`r(colsig)`	configurations passing the last displayed results (if `settest()` is specified)
`r(comm)`	common configurations (if `common` is specified)
`r(reducsols)`	final reduced configurations (if `reduce` is specified)

Matrices

`r(coincid)`	coincidence matrix (if `matx(coincid)` is specified)
`r(suffnec)`	sufficiency and necessity matrix (if `matx(suffnec)` is specified)

4.5 Postestimation commands and stand-alones

In addition to the base command and its options, the `fuzzy` program also includes many of the same options as stand-alone programs, to be used in testing specific configurations or as "postestimation" commands. Specifically, all the possible tests specified in `settest()`, the matrices in `matx()`, `truthtab()`, `coverage`, and `reduce` can all be used as stand-alone or post`fuzzy` commands. In both cases, they accept all the options that would pertain to them if specified in `fuzzy`.

When run as stand-alone programs, the user can enter specific configurations that do not have to exist in the dataset, but if they do not, their set components (named with single, capital letters) must. For example, if the user was specifically interested in the consistencies of two particular configurations (`A` · `B` · `C` and `a` · `B` · `C`), the command

```
. yvn Y ABC abc
```

could be run, which would produce the typical `yvn` output, but only for `ABC` and `abc`. Variables representing the configurations `ABC` and `abc` do not need to be present in the dataset for this command to work, but the individual set variables `Y`, `A`, `B`, and `C` do. The user could also specify the `sigonly`, the `greater()`, or both options, just as would be done if `yvn` was used in the `fuzzy` command.

These commands can also be used as postestimation commands. So, for example, if the user ran a simple `fuzzy` command with no options, `yvn` could be entered subsequently, which would produce the same output as if `yvn` had been included in `settest()` with the original `fuzzy` command. When these postestimation commands are used in this manner, they use the full varlist from the last run `fuzzy` command unless `last` is specified as an option, which will then use the last "displayed" configurations resulting from any invoked `settest()` options. If the user wishes to run the test on the final

reduced solution set (assuming `reduce` had been specified in the previous command), `usereduction` should be entered as an option.

Additionally, `reduce`, `coverage`, and `truthtab` can be used as postestimation commands and will automatically perform their operations on the last displayed configurations (`last` does not need to be specified because these programs can be used only if `settest()` was used in the last run `fuzzy` or stand-alone command). They also can be used as stand-alones to manually run their operations. For example, a user could perform a reduction on a specified set of configurations. When used in this manner, full configurations must be entered (i.e., it will not reduce `ABc aB` because the latter does not contain all of the constituent sets; but it would reduce `ABC aBc aBC`, which is logically equivalent). Finally, `reduce` as a stand-alone accepts the `dncare()` option for specifying configurations as do-not-care configurations, and `truthtab` accepts its `outsheet()` option.

Next there are a few extensions of `fuzzy` that only work as stand-alone programs.

`cnfgen` *newconfigurationlist* generates specific configuration variables. The individual sets must be included in the dataset, named with single, capital letters, and range from 0 to 1. For example, the user could enter

```
. cnfgen aBc ABC abc
```

to produce variables representing these three configurations.

`yvy` *outcomevar configuration1 configuration2* tests the *y* consistency of the first configuration entered against the *y* consistency of the second configuration entered. The configurations do not need to exist in the dataset, but their individual single, capital letter versions must. It is also possible to test individual sets against each other, but they must exist in the dataset (and be individual letters) and range from 0 to 1.

`fzplot` *outcomevar configuration* [`, mlabel(`*varname*`)`] will produce a sufficiency plot of the specified configuration and the outcome variable. The configuration does not need to exist in the dataset, but its individual single, capital letter versions must exist.

`mlabel(`*varname*`)` is an available option and is used just as it would be with `scatter`; see [G] **graph twoway scatter**.

`coverage` [*outcomevar configurations*] will produce the coverage statistics for the given configurations. The configurations do not need to exist in the dataset, but their individual single, capital letter versions must exist.

`setgen` *newvar* `=` *fcn*`(`*arguments*`)` [`,` *option*] works like `egen` but specifically for creating fuzzy sets that range from 0 to 1. The function *fcn* is one of the following:

`stdrank(`*varname*`)` rank orders the variable and then standardizes this ranking to range from 0 to 1. The equation for this standardization is

$$\frac{rankedvar - \min(rankedvar)}{\max(rankedvar) - \min(rankedvar)}$$

`drect(`*varname*`), anchors(`*numlist*`)` performs the "direct" transformation, outlined in Ragin (2008), which uses set values (the `anchors()` option) to calibrate the membership scores as levels of deviation from the anchors, in terms of log odds. The anchors should be entered in ascending order (i.e., mimimum threshold, crossover point, maximum threshold).

`ndrect(`*varname*`), grpdvar(`*varname*`)` performs the "indirect" transformation, outlined in Ragin (2008), which uses a regression based on the qualitative cutpoints in the `grpdvar()` option to determine membership scores. The `grpdvar()` option should be a categorization of the original variables into groups according to their levels of membership (e.g., more in than out = 0.33). This new, qualitatively grouped variable is regressed on the original in order to reshape the original variable into a set, ranging from 0 to 1, that is based on the knowledge-based grouping.

`crisp(`*varname*`)` [`, cutpt(`#`)`] will produce a "crispy" copy of the variable (i.e., a binary 0/1 variable). The default is to split the new variable at the original variable's median, such that the median value and below will be coded as 0, otherwise 1 (unless the median and the maximum value are the same in the original variable, in which case the median will be coded 1). If the median is not an appropriate delineation, the `cutpt()` option can be used to specify the value at which to split the variable, such that all those values equal to or less than the cutpoint will be coded 0 and the rest 1.

5 Remarks and examples

To illustrate the capabilities and functionality of the `fuzzy` command, we present analyses using the National Study of Youth and Religion (Smith and Denton 2003). The study is a nationally representative survey of 3,390 teenagers and their parents. For simplicity, 5th and 6th grade adolescents are dropped and listwise deletion is employed for all variables in analyses, resulting in a sample size of $n = 3{,}112$.

For the purposes of these examples, we will be using the teen's reported grades (`grds`) during the past academic semester as the outcome variable (higher scores indicate better grades). The independent variables will include their number of work hours (`workhrs`), their number of friends reported to use drugs or drink alcohol (`peerus`), an index of parental monitoring (`pmonit`), and their reported alcohol usage (`alcuse`). All of the variables have been transformed into sets using the `stdrank()` function in `setgen`.

This transformation is one of many that could be used to convert variables into sets. To illustrate its general properties, we have presented the frequency distributions from the original grade variable and its new set version.

```
. use nsyr_example_data
. tabulate grds, nol
```

grds	Freq.	Percent	Cum.
1	13	0.42	0.42
2	11	0.35	0.77
3	19	0.61	1.38
4	88	2.83	4.21
5	387	12.44	16.65
6	503	16.16	32.81
7	402	12.92	45.73
8	1,088	34.96	80.69
9	325	10.44	91.13
10	276	8.87	100.00
Total	3,112	100.00	

```
. setgen G = stdrank(grds)
. tabulate G
```

rank of (grds)	Freq.	Percent	Cum.
0	13	0.42	0.42
.0040438	11	0.35	0.77
.0090986	19	0.61	1.38
.0271272	88	2.83	4.21
.1071609	387	12.44	16.65
.2571188	503	16.16	32.81
.409604	402	12.92	45.73
.6606571	1,088	34.96	80.69
.8987363	325	10.44	91.13
1	276	8.87	100.00
Total	3,112	100.00	

The distribution of cases has not changed, but the scale has been "fuzzified" to range between 0 and 1, with the values now representing the level of membership in the set "good grades". The similarity of distributions is not required, and in fact there are situations that may call for more user-knowledge-based coding (for a discussion of this type of transformation, see Ragin [2000]). In the current example, we have chosen to use the standardized rank transformation because it is a relatively straightforward conversion.

The following is the distribution of each variable and its corresponding set:

Variable	Original range	Original mean	Set mean
Grades	1–10	7.26	0.52
Work hours	0–40	3.31	0.19
Peer substance use	0–5	0.70	0.25
Parent monitoring	0–4	2.62	0.51
Alcohol use	0–5.5	0.60	0.27

5.1 Configuration testing

The first step in the fuzzy analysis might be to see which configurations contain the greatest number of individuals.

```
. use nsyr_example_data, clear
. foreach var of varlist grds workhrs peerus pmonit alcuse {
  2.         setgen st`var´ = stdrank(`var´)
  3. }
. fuzzy stgrds stworkhrs stpeerus stpmonit stalcuse, label(G W P M A)
. tabulate bestfit
```

bestfit	Freq.	Percent	Cum.
WPMA	45	1.45	1.45
WPMa	32	1.03	2.47
WPmA	198	6.36	8.84
WPma	47	1.51	10.35
WpMA	37	1.19	11.54
WpMa	148	4.76	16.29
WpmA	83	2.67	18.96
Wpma	113	3.63	22.59
wPMA	110	3.53	26.12
wPMa	115	3.70	29.82
wPmA	318	10.22	40.04
wPma	131	4.21	44.25
wpMA	158	5.08	49.33
wpMa	882	28.34	77.67
wpmA	220	7.07	84.74
wpma	475	15.26	100.00
Total	3,112	100.00	

Thus 1.45% of adolescents are likely to experience all of the independent measures at above-median levels (`WPMA`), while the most common configuration (`wpMa`), with 28.34% of the sample best fitting it, indicates low work hours, not many friends who use substances, high parent monitoring, and low levels of individual alcohol use. This command would also have produced five new individual variables (`G`, `W`, `P`, `M`, `A`), all copies of the original variables for which they were named. If the user did not wish to keep these new variables, `drop` could be specified.

The `label()` option, however, is not necessary. Had it not been specified, we would have seen the following:

```
. fuzzy stgrds stworkhrs stpeerus stpmonit stalcuse
. tabulate bestfit
```

bestfit	Freq.	Percent	Cum.
BDEF	45	1.45	1.45
BDEf	32	1.03	2.47
BDeF	198	6.36	8.84
BDef	47	1.51	10.35
BdEF	37	1.19	11.54
BdEf	148	4.76	16.29
BdeF	83	2.67	18.96
Bdef	113	3.63	22.59
bDEF	110	3.53	26.12
bDEf	115	3.70	29.82
bDeF	318	10.22	40.04
bDef	131	4.21	44.25
bdEF	158	5.08	49.33
bdEf	882	28.34	77.67
bdeF	220	7.07	84.74
bdef	475	15.26	100.00
Total	3,112	100.00	

The only difference in the two commands, therefore, is the naming of the configurations. Of course, using `label()` may be helpful in keeping straight the sets involved in creating the configurations. Also, because `label()` was not used, the variables `B`, `D`, `E`, and `F` are, by default, deleted when the program is terminated, but `keepsets` could be specified to prevent this if the user wanted to use these generic names in future calls. In the remaining examples, we will use the single letter designations with the `fuzzy` command (treating it as though the user had already run the first invocation of `fuzzy` above). But all of the commands could be run with original variables (with or without the `label()` option).

❑ Technical note

While the total in the `bestfit` variable does add up to the total number of non-missing cases, this may not always be true. Cases scoring 0.5 on all individual predictor sets will not appear because they belong equally to all configurations.

❑

Next one might want to get a sense of the relationship between the independent variable sets by using fuzzy-set methods.

(*Continued on next page*)

```
. fuzzy G W P M A, matx(coincid suffnec) standardized
```

Coincidence Matrix

	G	W	P	M	A
G	1.000				
W	0.603	1.000			
P	0.574	0.422	1.000		
M	0.699	0.485	0.453	1.000	
A	0.595	0.459	0.630	0.482	1.000

Sufficiency and Necessity Matrix

	G	W	P	M	A
G	1.000	0.214	0.276	0.677	0.311
W	0.603	1.000	0.422	0.485	0.459
P	0.574	0.311	1.000	0.453	0.630
M	0.699	0.178	0.225	1.000	0.261
A	0.595	0.311	0.579	0.482	1.000

The high work hours and high grades sets overlap by 60% of their possible shared area, as shown by their 0.603 coincidence score. The standardization option is especially helpful because several of the variables (e.g., work hours and alcohol use) do not contain many members at high degrees. In fact, the coincidence score between grades and work hours was one of the lowest when the size of the variables was not accounted for by the standardization option, which demonstrates the utility of also invoking `standardized` with the coincidence matrix. We also see that high parent monitoring is the single set that—alone—is most sufficient for predicting the outcome (consistency = 0.699). Both of these matrices are returned and could be used to run further tests.

Seeing that the variable sets are indeed related, it would now be helpful to examine their resulting configurations' sufficiency with the outcome. To do so, we will run a series of tests, the first of which is the most basic, to get a sense of each configuration's consistency with the outcome.

```
. fuzzy G W P M A, settest(yvv)
```

Y-Consistency vs. Set Value

Set	YConsist	Set Value	F	P	NumBestFit
wpma	0.790	0.800	1.78	0.182	475
wpmA	0.771	0.800	4.76	0.029	220
wpMa	0.739	0.800	69.15	0.000	882
wpMA	0.805	0.800	0.14	0.705	158
wPma	0.770	0.800	4.11	0.043	131
wPmA	0.650	0.800	83.20	0.000	318
wPMa	0.801	0.800	0.01	0.926	115
wPMA	0.759	0.800	5.51	0.019	110
Wpma	0.804	0.800	0.06	0.804	113
WpmA	0.759	0.800	3.12	0.077	83
WpMa	0.790	0.800	0.41	0.523	148
WpMA	0.843	0.800	3.95	0.047	37
WPma	0.819	0.800	0.76	0.385	47
WPmA	0.648	0.800	47.15	0.000	198
WPMa	0.835	0.800	1.95	0.162	32
WPMA	0.796	0.800	0.03	0.869	45

Each configuration's consistency is displayed, as well as the resulting test against 0.800. The results indicate that only the configuration `WpMA` is significantly more consistent than 0.800 at the 0.05 level. Of course, one of the primary advantages of the `fuzzy` command is that it can perform more stringent tests of each configuration's consistency value. We look for the configurations that have y consistencies significantly greater than 0.700, as well as significantly greater than their n consistencies.

```
. fuzzy G W P M A, settest(yvv yvn) sigonly greater(col1) conval(.700) common
```

Y-CONSISTENCY vs N-CONSISTENCY

Set	YCons	NCons	F	P	NumBestFit
wpma	0.790	0.734	16.49	0.000	475
wpMa	0.739	0.646	45.83	0.000	882
Wpma	0.804	0.727	6.79	0.009	113
WpMa	0.790	0.698	9.46	0.002	148
WPma	0.819	0.715	5.70	0.017	47

Y-Consistency vs. Set Value

Set	YConsist	Set Value	F	P	NumBestFit
wpma	0.790	0.700	136.51	0.000	475
wpmA	0.771	0.700	27.69	0.000	220
wpMa	0.739	0.700	27.20	0.000	882
wpMA	0.805	0.700	68.45	0.000	158
wPma	0.770	0.700	21.90	0.000	131
wPMa	0.801	0.700	48.95	0.000	115
wPMA	0.759	0.700	11.33	0.001	110
Wpma	0.804	0.700	43.85	0.000	113
WpmA	0.759	0.700	6.45	0.011	83
WpMa	0.790	0.700	31.27	0.000	148
WpMA	0.843	0.700	43.19	0.000	37
WPma	0.819	0.700	29.96	0.000	47
WPMa	0.835	0.700	28.48	0.000	32
WPMA	0.796	0.700	12.50	0.000	45

```
Common Sets
wpma wpMa Wpma WpMa WPma
```

Notice that the `sigonly` and `greater()` options apply to both the `yvv` and `yvn` tests, while `conval()` pertains only to `yvv`. Using the `common` option is a quick way to see the configurations that pass both tests. From the given output, it appears that `wpma`, `wpMa`, `Wpma`, `WpMa`, and `WPma` are the most highly consistent configurations with good grades. It is possible though that these configurations may logically overlap. To perform the reduction:

(*Continued on next page*)

```
. fuzzy G W P M A, settest(yvv yvn) sigonly greater(col1) conval(.700)
> common reduce
```

(*output omitted*)

```
Common Sets
wpma wpMa Wpma WpMa WPma

5 Solutions Entered as True

Minimum Configuration Reduction Set
Wma pa

Final Reduction Set

Coverage
Set         Raw Coverage     Unique Coverage     Solution Consistency
W*m*a          0.113             0.017                0.778
p*a            0.732             0.636                0.603

Total Coverage = 0.749
Solution Consistency = 0.604
```

The five initial configurations have been collapsed into two. (The "Minimum Configuration Reduction Set" displays the reduced configurations from the initial step. In certain cases, this will be different than the "Final Reduction Set", which results from the second step—employing prime implicants—of the Quine–McCluskey algorithm.) We also can tell that low personal alcohol use (`a`) is key to higher grades. When this base set is conjoined with either low peer substance use (`p`) or high work hours combined with low parent monitoring (`W * m`), the adolescent is also likely to be achieving higher grades.[3] Additionally, this example displays the benefit of fuzzy methods more generally, as we find a somewhat surprising relationship between work hours and a positive outcome. Normally, higher work hours have been found to increase the likelihood of a number of delinquent activities (Bachman and Schulenberg 1993; Safron, Schulenberg, and Bachman 2001; and Paschall, Ringwalt, and Flewelling 2002). When this high work intensity, however, is concurrently combined with low personal alcohol use and low parent monitoring, it is associated with higher academic achievement. Understanding why low parent monitoring is included in this configuration is difficult without further examination although it may indicate independent youth (i.e., working many hours breaks them from parent monitoring, but they still do not participate in delinquent activities, which all conjoins to be associated with positive academic outcomes).

Finally, it would have been possible to run the entire set of analyses in the following single command:

3. We recognize that the reduced configuration `p*a` has a consistency below the 0.7 set value. This drop in value (i.e., increased coverage but reduced effectiveness) is due to the increased amount of people who belong to the minimized configuration. We recognize this difference as an important methodological (and substantive) issue that should be addressed in future research.

```
. drop G W P M A
. fuzzy stgrds stworkhrs stpeerus stpmonit stalcuse, label(G W P M A)
> matx(coincid suffnec) standard settest(yvv yvn) sigonly greater(col1)
> conval(.700) common reduce
  (output omitted)
Common Sets
wpma wpMa Wpma WpMa WPma
5 Solutions Entered as True
Minimum Configuration Reduction Set
Wma pa
Final Reduction Set
Coverage
Set                          Raw Coverage   Unique Coverage   Solution Consistency
STWORKHRS*stpmonit*stalcuse     0.113           0.017              0.778
stpeerus*stalcuse               0.732           0.636              0.603
Total Coverage = 0.749
Solution Consistency = 0.604
```

When the original variables are used along with `reduce`, the reduction output uses the original variable names, making it possible to use this output in potential tables.

❑ Technical note

It is possible to pass configurations to that have consistency scores that are greater with the negation (i.e., $1 - outcome$) than the outcome to reduce (e.g., `fuzzy` *varlist*`, settest(yvn) greater(col2) reduce`), but if this is done, `reduce` still uses the outcome to compute the coverage and consistency scores of the reduced configurations. If these values are desired for the negation, we suggest the user create a specific variable, to be used as the outcome, that represents $1 - outcome$ and revert to specifying `greater(col1)` with the `yvn` test.

❑

5.2 Postestimation testing and stand-alones

Many of the options within `fuzzy` have been constructed to be used as stand-alone programs, which may be highly useful to test specific configurations. For example, perhaps one is interested in the coincidence of the configurations, in addition to the individual variables:

(*Continued on next page*)

```
. fuzzy G W P M A
. coincid `r(sets)´
```

Coincidence Matrix

	c1	c2	c3	c4	c5	c6
r1	1.000					
r2	0.220	1.000				
r3	0.432	0.133	1.000			
r4	0.180	0.550	0.162	1.000		
r5	0.156	0.168	0.096	0.145	1.000	
r6	0.074	0.202	0.046	0.159	0.284	1.000
r7	0.128	0.140	0.113	0.171	0.535	0.215
r8	0.066	0.189	0.056	0.237	0.275	0.407
r9	0.097	0.079	0.065	0.074	0.074	0.042
r10	0.039	0.115	0.025	0.106	0.059	0.064
r11	0.085	0.066	0.076	0.077	0.060	0.034
r12	0.035	0.104	0.028	0.126	0.053	0.056
r13	0.033	0.053	0.021	0.049	0.128	0.074
r14	0.020	0.055	0.013	0.048	0.069	0.089
r15	0.028	0.047	0.023	0.056	0.109	0.062
r16	0.018	0.054	0.014	0.064	0.069	0.082

	c7	c8	c9	c10	c11	c12
r7	1.000					
r8	0.343	1.000				
r9	0.065	0.046	1.000			
r10	0.052	0.073	0.269	1.000		
r11	0.069	0.048	0.464	0.180	1.000	
r12	0.061	0.087	0.213	0.482	0.237	1.000
r13	0.109	0.086	0.207	0.189	0.139	0.170
r14	0.057	0.090	0.098	0.195	0.071	0.149
r15	0.131	0.100	0.158	0.158	0.175	0.220
r16	0.080	0.133	0.098	0.201	0.102	0.287

	c13	c14	c15	c16
r13	1.000			
r14	0.242	1.000		
r15	0.464	0.176	1.000	
r16	0.264	0.315	0.369	1.000

The `fuzzy` command is used to generate all the possible configurations and then create a full coincidence matrix of these configurations. We could have included the outcome variable in this matrix, but we do not have to in this case. If we had not entered `r(sets)` after the `coincid` command in this case, a coincidence matrix for just the individual set variables would have been reproduced. There are also two alternative methods for producing similar results: (1) in the `fuzzy` command line, the user could have invoked the `keepconfigs` option and then entered

```
. coincid wpma - WPMA
```

or (2) the user could have, without running the first `fuzzy` command, manually entered every possible (or desired) configuration:

```
. coincid wpma wpmA wpMA
```

Again, when these postestimation commands are used, only the individual sets (named as capital letters) must exist in the dataset. For example, we could run the following command as long as the set variables `G`, `W`, `P`, `M`, and `A` existed in the dataset (even if the configuration variables did not):

```
. yvn G wpma Wpma WPma
Y-CONSISTENCY vs N-CONSISTENCY
Set         YCons     NCons       F         P     NumBestFit
wpma        0.790     0.734     16.49     0.000       475
Wpma        0.804     0.727      6.79     0.009       113
WPma        0.819     0.715      5.70     0.017        47
```

When using the options as primary programs, they will accept all of the options associated with them in the main `fuzzy` (e.g., we could have specified `sigonly`, `slevel()`, `greater()`, or a combination of these in the above example). When using the tests in this manner, it is still necessary to specify the dependent variable in addition to the configurations to be tested.

❑ Technical note

If `yvo` or `cmvom` is used in this manner, the "other" configuration that each configuration is tested against consists only of those configurations that are entered into the command line. For example, had `yvo` been used instead of `yvn` in the last example, each configuration would have been tested against the other two configurations instead of the full possible configuration set. Additionally, if `yvo` or `cmvom` is used in this manner, at least two configurations must be specified.

❑

(*Continued on next page*)

Many of the described extensions work in this manner as well. For example, in figure 1, we can visually see the relationship between a specific configuration and the outcome.

```
. fzplot G Wpma
```

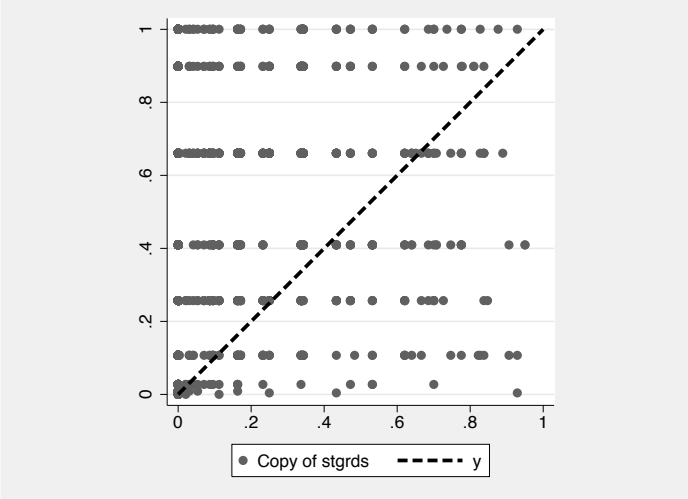

Figure 1. Graph of the relationship between outcome (`stgrds`) and configuration (`Wmpa`).

Or it may be beneficial to test two configurations' consistencies against one another. For example, given the reduction above, we might have concluded that the configuration `Wma` was perhaps the most important because of its higher consistency. But we can test if this difference is significant with

```
. yvy G Wma pa
Y-CONSISTENCY vs Y-CONSISTENCY
WmaYcons    paYCons          F         P
0.778        0.603         142.28     0.000
```

This test shows that, in fact, the two configurations' consistencies are significantly different. Further, the configurations do not have to be "full" in the sense that they use all of the original sets. Rather, the configurations are generated by taking the minimum value of the specified combination. `reduce` used as a stand-alone program does require "full" configurations to perform the reduction properly.

Finally, these commands can be used following `fuzzy` (and each other) as postestimation commands. When this is done, the default is to use the original variable list from the last run `fuzzy` (or other postestimation command). For example,

```
. fuzzy G W P M A
. yvv
```

Y-Consistency vs. Set Value

Set	YConsist	Set Value	F	P	NumBestFit
wpma	0.790	0.800	1.78	0.182	475
wpmA	0.771	0.800	4.76	0.029	220
wpMa	0.739	0.800	69.15	0.000	882
wpMA	0.805	0.800	0.14	0.705	158
wPma	0.770	0.800	4.11	0.043	131
wPmA	0.650	0.800	83.20	0.000	318
wPMa	0.801	0.800	0.01	0.926	115
wPMA	0.759	0.800	5.51	0.019	110
Wpma	0.804	0.800	0.06	0.804	113
WpmA	0.759	0.800	3.12	0.077	83
WpMa	0.790	0.800	0.41	0.523	148
WpMA	0.843	0.800	3.95	0.047	37
WPma	0.819	0.800	0.76	0.385	47
WPmA	0.648	0.800	47.15	0.000	198
WPMa	0.835	0.800	1.95	0.162	32
WPMA	0.796	0.800	0.03	0.869	45

This is similar to running the command on one line. But it is also possible to run post hoc examination of limited sets of configurations. For example, if in the last run `fuzzy` command the display was altered using one of the `settest()` options or `reduce`, then it is possible to specify `last` or `usereduction` as an option with the postestimation commands to restrict their analyses to the last displayed set of configurations.

```
. fuzzy G W P M A, settest(yvv yvn) sigonly greater(col1) conval(.700) common
> reduce
```

(*output omitted*)

```
. yvv, usered conv(.700)
```

Y-Consistency vs. Set Value

Set	YConsist	Set Value	F	P	NumBestFit
Wma	0.778	0.700	27.39	0.000	160
pa	0.603	0.700	233.46	0.000	1618

The use of the options as postestimation commands also allows for more complex and flexible sets of tests. For example, if the user wanted to identify and reduce all the configurations having consistencies with the outcome that were greater than with the negation (but not necessarily significantly so) and that were also significantly greater than 0.700, it would be impossible to do in one call to `fuzzy` (because the `settest()` options apply to all of the tests in `settest()`). But the user could accomplish this goal using a simple series of commands:

(*Continued on next page*)

```
. fuzzy G W P M A, settest(yvn) greater(col1)
```

Y-CONSISTENCY vs N-CONSISTENCY

Set	YCons	NCons	F	P	NumBestFit
wpma	0.790	0.734	16.49	0.000	475
wpMa	0.739	0.646	45.83	0.000	882
wpMA	0.805	0.801	0.03	0.855	158
wPMa	0.801	0.768	1.50	0.220	115
Wpma	0.804	0.727	6.79	0.009	113
WpmA	0.759	0.756	0.00	0.949	83
WpMa	0.790	0.698	9.46	0.002	148
WpMA	0.843	0.796	1.48	0.223	37
WPma	0.819	0.715	5.70	0.017	47
WPMa	0.835	0.753	3.07	0.080	32
WPMA	0.796	0.791	0.01	0.925	45

```
. yvv, last conval(.700) greater(col1) sigonly
```

Y-Consistency vs. Set Value

Set	YConsist	Set Value	F	P	NumBestFit
wpma	0.790	0.700	136.51	0.000	475
wpMa	0.739	0.700	27.20	0.000	882
wpMA	0.805	0.700	68.45	0.000	158
wPMa	0.801	0.700	48.95	0.000	115
Wpma	0.804	0.700	43.85	0.000	113
WpmA	0.759	0.700	6.45	0.011	83
WpMa	0.790	0.700	31.27	0.000	148
WpMA	0.843	0.700	43.19	0.000	37
WPma	0.819	0.700	29.96	0.000	47
WPMa	0.835	0.700	28.48	0.000	32
WPMA	0.796	0.700	12.50	0.000	45

```
. reduce
11 Solutions Entered as True
```

Minimum Configuration Reduction Set

```
pa pM Ma Wp Wa WM
```

Final Reduction Set

Coverage

Set	Raw Coverage	Unique Coverage	Solution Consistency
p*a	0.732	0.134	0.603
p*M	0.602	0.026	0.717
M*a	0.611	0.034	0.721
W*p	0.152	0.013	0.666
W*a	0.153	0.012	0.696
W*M	0.129	0.011	0.750

```
Total Coverage = 0.847
Solution Consistency = 0.598
```

Finally, if the user knew the specific configurations that should be treated as true, then `reduce` could be used as its own stand-alone command. As a simple illustrative example, if the user had some reason to believe only those configurations with high work hours and high parent monitoring should be considered true, the following command could be used:

```
. reduce G WpMa WPMa WpMA WPMA

4 Solutions Entered as True

Minimum Configuration Reduction Set
WM

Final Reduction Set

Coverage
Set       Raw Coverage     Unique Coverage     Solution Consistency
W*M               0.129             0.129                    0.750

Total Coverage = 0.129
Solution Consistency = 0.750
```

As expected, the configuration set reduces to just high work hours (`W`) and high parent monitoring (`M`).

6 Acknowledgments

We would like to thank Nick Cox (University of Durham, UK) for early help with the programming that got the larger program off the ground. We also appreciate Charles Ragin and Sarah Strand (University of Arizona) for useful testing and feedback of earlier versions. The development of this program was supported in part by the Center for Developmental Sciences, University of North Carolina at Chapel Hill.

7 References

Bachman, J. G., and J. E. Schulenberg. 1993. How part-time work intensity relates to drug use, problem behavior, time use, and satisfaction among high school seniors: Are these consequences or merely correlates? *Developmental Psychology* 29: 220–235.

Greckhamer, T., V. F. Misangyi, H. Elms, and R. Lacey. 2007. Using Qualitative Comparative Analysis in Strategic Management Research: An Examination of Combinations of Industry, Corporate, and Business-unit Effects. *Organizational Research Methods OnlineFirst* 1–32.

Kalleberg, A. L., and S. Vaisey. 2005. Pathways to a good job: Perceived work quality among the machinists in North America. *British Journal of Industrial Relations* 43: 431–454.

Lin, N., and W. M. Ensel. 1989. Life stress and health: Stressors and resources. *American Sociological Review* 54: 382–399.

Longest, K. C., and P. Thoits. 2007. The stress process and physical health: A configurational approach. Paper presented at the American Sociological Annual Meeting.

Mahoney, J. 2003. Long-run development and the legacy of colonialism in Spanish America. *American Journal of Sociology* 109: 50–106.

Paschall, M. J., C. Ringwalt, and R. L. Flewelling. 2002. Explaining higher levels of alcohol use among working adolescents: An analysis of potential explanatory variables. *Journal of Studies on Alcohol* 63: 169–178.

Ragin, C. 2000. *Fuzzy-Set Social Science*. Chicago: University of Chicago Press.

———. 2006. Set relations in social research: Evaluating the consistency and coverage. *Political Analysis* 14: 291–310.

———. 2008. Fuzzy set analysis: Calibration versus measurement. In *Oxford Handbook of Political Methodology*, ed. J. Box-Steffensmeier, H. Brady, and D. Collier. Oxford: Oxford University Press.

Ragin, C. C. 1987. *The Comparative Method: Moving Beyond Qualitative and Quantitative Strategies*. Berkeley: University of California Press.

Roscigno, V. J., and R. Hodson. 2004. The organizational and social foundations of worker resistance. *American Sociological Review* 69: 14–39.

Safron, D. J., J. E. Schulenberg, and J. G. Bachman. 2001. Part-time work and hurried adolescence: The links among work intensity, social activities, health behaviors, and substance use. *Journal of Health and Social Behavior* 42: 425–449.

Schuit, A. J., A. J. M. Van Loon, M. Tijhuis, and M. C. Ocké. 2002. Clustering of lifestyle risk factors in a general adult population. *Preventive Medicine* 35: 219–224.

Shanahan, M. J., L. D. Erikson, S. Vaisey, and A. Smolen. 2007. Helping relationships and genetic propensities: A combinatoric study of DRD2, mentoring, and educational continuation. *Twin Research and Human Genetics* 10: 285–298.

Smith, C., and M. L. Denton. 2003. Methodological design and procedures for the National Study of Youth and Religion. Technical report, University of North Carolina, Chapel Hill, NC. http://www.youthandreligion.org/.

Smithson, M., and J. Verkuilen. 2006. *Fuzzy Set Theory: Applications in the Social Sciences*. Thousand Oaks, CA: Sage.

Thoits, P. A. 1995. Stress, coping, and social support processes: Where are we? What next? *Journal of Health and Social Behavior* 35: 53–79.

Vaisey, S. 2007. Culture, structure, and community: The search for belonging in 50 urban communes. *American Sociological Review* 72: 851–873.

About the authors

Kyle C. Longest is a PhD candidate in the department of sociology at the University of North Carolina at Chapel Hill. His research interests focus on adolescent development with special attention to identity, education, substance use, and the transition out of high school.

Stephen Vaisey is a doctoral student in the department of sociology at the University of North Carolina at Chapel Hill. His research focuses on the relationship between culture and cognition.