Manipulation of categorical data edits and error localization with the editrules package

package version 1.9.0

Mark van der Loo and Edwin de Jonge

November 16, 2011

Abstract

Analyses of categorical data are often hindered by the occurrence of inconsistent or incomplete raw data. Although R has many features for analyzing categorical data, the functionality for error localization and correction are currently limited. The editrules package is designed to offer a user-friendly interface for edit definition, manipulation, and error localization based on the generalized paradigm of Fellegi and Holt. The package is a toolbox, providing basic functionality such as rule definition, error checking, rule consistency tests and more. On top of that, the package includes algorithms which localize errors based on the generalized paradigm of Fellegi and Holt.

Under the hood, the package has several innovations of which the most important one is a variable elimination method which -as far as the authors know- is new to the field of data editing. The paper also introduces an elegant description of edit rule manipulation in terms of the so-called resolution operator.

This is the second paper describing functionalities of the R editrules package and marks the completion of editrules version 2.0. The first paper (De Jonge and Van der Loo, 2011) describes methods and implementation for handling numerical data while this paper is concerned with handling categorical data.

Contents

List of Algorithms

Reading guide. This paper describes the algorithms and some of the math behind the package as well as the functionality of the editrules package. Readers less interested in the technical background who want to get started quickly can skip some technical sections and read sections 1 $\rightarrow$ 2.2 $\rightarrow$ 2.3 $\rightarrow$ 3.1 $\rightarrow$ 3.4 $\rightarrow$ 4.2 $\rightarrow$ 4.3.

1 Introduction

Analyses of categorical data are often hindered by occurrences of incomplete or inconsistent raw data records. The process of locating and correcting such errors is referred to as *data editing*, and it has been estimated that National Statistics Institutes spend up to 40% of their resources on this process (De Waal et al., 2011). For this reason, considerable attention is paid to the development of data editing methods that can be automated. Since data are often required to obey many interrelated consistency rules, data editing can be too complex to perform manually. Winkler (1999) mentions practical cases where records have to obey 250, 300 or even 750 internal consistency rules. Although the R statistical environment has numerous facilities for analyzing categorical data [See *e.g.* Husson et al. (2010)], the options for error localization and record correction are currently limited.

This paper presents the editrules package which was developed to help closing the gap between raw data retrieval and data analysis with R. The main purpose of the editrules package is to provide a user-friendly environment for handling data restriction rules, to apply those rules to data, and to localize erroneous fields in data based on the generalized principle of Fellegi and Holt (1976). The package does not offer functionality for data correction. However, it does facilitate the identification of the set of solutions for an error correction problem.

Under the hood, the package contains several innovations with respect to the branch-and-bound algorithm for error localization in categorical data described in De Waal et al. (2011). The most important innovation is a variable elimination algorithm which allows for on-the-fly redundant rule removal. As far as the authors know, this algorithm is new to the field of data editing. To facilitate the description of algorithms and their properties, we also introduce a formulation of categorical edit manipulations in terms of the so-called resolution operator. This formulation allows for elegant algebraic proofs of properties of the algorithms and rule manipulations applied in the editrules package.

The current paper complements our previous paper on the treatment of numerical data (De Jonge and Van der Loo, 2011). We describe the algorithms underlying editrules' functionality and the internal representation of categorical data. Examples in R code are given throughout the text to assist new users in getting started with the package.

2 Categorical data and edit rules

The value domain of categorical data records is usually limited by rules interrelating these variables. The simplest examples are cases where the value of one variable excludes values of another variable. For example: if the age class of a person is "child", then (by law) the marital status cannot be "married". In survey or administrative data, violations of such rules are frequently encountered. Resolving such violations is an important step prior to data analysis and estimation.

In this section we describe the representation of edits and records as implemented in the editrules package. Subsection 2.1 describes the background while subsection 2.2 describes implementation and gives coded examples.

2.1 Boolean representation of records and edits

A categorical data record v with n variables may be defined as an element of the Cartesian product space D (for domain):

$$D = D_1 \times D_2 \times \cdots \times D_n, \tag{1}$$

where each D_k is a finite set of d_k possible categories for variable k. The value domain defined above is also referred to as the *data model* of a dataset. We label the categories as follows:

$$D_k = \{v_k \in D_k \,|\, v_k = 1, 2, \ldots, d_k\}. \tag{2}$$

A restriction e is a subset of D and we say that *if* $v \in e$ *then* v *violates* e. Conversely, when $v \notin e$ we say that v *satisfies* e. In data editing literature, such rules are referred to as *edit rules* or *edits*, in short. In the context of contingency tables they are referred to as *structural zeros* since each rule implies that one or more cells in the $d_1 \times d_2 \times \cdots \times d_n$ contingency table must be zero. A record is *valid* if it satisfies every edit imposed on D.

Categorical records may be represented as a vector of boolean values. A boolean vector of dimension d is an element of the boolean algebra

$$\mathbb{B}^d = \left(\{0,1\}^d, \wedge, \vee, \neg\right), \tag{3}$$

where 0 and 1 have the usual interpretations FALSE and TRUE and the logical operators work element-wise on their operands. To facilitate the discussion we will also allow the standard arithmetic operations addition and subtraction on boolean vectors (this is also consistent with the way R handles vectors of class logical).

To represent a record $v = (v_1, v_2, \ldots, v_n)$, we assign to every category v_k in D_k a unique standard basis vector $\vec{\delta}_k(v_k)$ of $\mathbb{B}^{d_k}$. The boolean representation $\rho(v)$ of the full record is the direct sum

$$v \xrightarrow{\rho} \vec{\delta}_1(v_1) \oplus \vec{\delta}_2(v_2) \oplus \cdots \oplus \vec{\delta}_n(v_n), \tag{4}$$

which we will write as the direct vector sum

$$\rho(v) = \mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \cdots \oplus \mathbf{v}_n \equiv \mathbf{v}. \tag{5}$$

The dimension d of $\rho(v)$ is given by the total number of categories of all variables

$$d = \sum_{k=1}^{n} d_k. \tag{6}$$

Example 2.1.1. *Consider the variables marital status, age, and position in household from the domain $D = D_1 \times D_2 \times D_3$. We define*

$$\begin{aligned} D_1 &= \{\mathsf{married}, \mathsf{unmarried}, \mathsf{widowed}, \mathsf{divorced}\} && (7)\\ D_2 &= \{\mathsf{under\text{-}aged}, \mathsf{adult}\} && (8)\\ D_3 &= \{\mathsf{partner}, \mathsf{child}, \mathsf{other}\}. && (9)\end{aligned}$$

The record $r = (\mathsf{married}, \mathsf{adult}, \mathsf{partner})$ has boolean representation

$$\rho(r) = (1,0,0,0)_1 \oplus (0,1)_2 \oplus (1,0,0)_3 = (1,0,0,0,0,1,1,0,0). \tag{10}$$

□

An edit e is a subset of records in D which can be written as the Cartesian product

$$e = A_1 \times A_2 \cdots \times A_n, \text{ where } A_k \subseteq D_k,\ k = 1, 2, \cdots n. \tag{11}$$

The interpretation of an edit is that if a record $v \in e$, then v is considered invalid. The following properties follow immediately.

Remark 2.1.2. *If $e \subset D$ and $e' \subset D$ are edits, then $e \cup e' = \{e, e'\}$ and $e \cap e' = A_1 \cap A'_1 \times A_2 \cap A'_2 \times \cdots \times A_n \cap A'_n$ are also edits.* □

An edit, expressed as in Eq. (11) is said to be in normal form. A variable k is *involved* in an edit if $A_k \subset D_k$. Conversely, we say that e *involves* k if k is involved in e. A variable k for which $A_k = D_k$ is not involved in e. Since every category v_k is mapped to a unique basis vector $\vec{\delta}_k(v_k)$, edits have a boolean representation $\rho(e)$, given by

$$e \overset{\rho}{\to} \bigvee_{v_1 \in A_1} \vec{\delta}_1(v_1) \oplus \bigvee_{v_2 \in A_2} \vec{\delta}_2(v_2) \oplus \cdots \oplus \bigvee_{v_n \in A_n} \vec{\delta}_n(v_n), \tag{12}$$

which may simply be written as

$$\rho(e) = \mathbf{a}_1 \oplus \mathbf{a}_2 \oplus \cdots \oplus \mathbf{a}_n \equiv \mathbf{a}. \tag{13}$$

Example 2.1.3. *Using the domain from Example 2.1.1, the edit that says that under-aged people cannot be married has set representation*

$$e = \{\text{married}\} \times \{\text{under-aged}\} \times \{\text{partner}, \text{child}, \text{other}\} \tag{14}$$

which translates to the boolean representation

$$\rho(e) = (1,0,0,0) \oplus (1,0) \oplus (1,1,1) = (1,0,0,0,0,1,1,1,1). \tag{15}$$

□

In the boolean representation some properties can be checked by simple calculations. For example, an edit involves variable k if and only if the inner product $\mathbf{1}_{d_k} \cdot \mathbf{a}_k < d_k$, where $\mathbf{1}_{d_k}$ is a d_k vector of ones.

A record v violates an edit if every $v_k \in A_k$. In the boolean representation this can be written as a condition on the standard inner product between the boolean representation of a record and an edit:

$$\sum_{k=1}^{n} \vec{\delta}_k(v_k) \cdot \mathbf{a}_k = \mathbf{v} \cdot \mathbf{a} = n. \tag{16}$$

Each edit may be written as the union of every record it contains. We therefore introduce the following definition.

Definition 2.1.4 (domination). *Given two edits e and e'. If each record $v \in e$ is also in e', we say that e' dominates e. We use the notation $e \prec e'$ if e' contains at least one record more than e and $e \preceq e'$ otherwise. Similarly, if E and E' are sets of edits, and every $v \in E$ is also in E', we write $E \prec E'$ or $E \preceq E'$.*

We introduce the symbols $\prec$ and $\preceq$ to indicate a difference with $\subset$ and $\subseteq$. The first symbols are used to in connection with the space of records covered by a set of edits. The latter are used to indicate subsets of explicitly defined edits. The symbols $\prec$ and $\preceq$ are independent of the explicit definition of edits while $\subset$ and $\subseteq$ are dependent.

Example 2.1.5. *Consider a 2-variable domain $D = D_1 \times D_2$ with $D_1 = \{\mathrm{a, b, c}\}$ and $D_2 = \{\mathrm{d, e}\}$. Furthermore, we have the edits $e_1 = \{\mathrm{a, b}\} \times \{\mathrm{d}\}$, $e_2 = \{\mathrm{a}\} \times \{\mathrm{d}\}$ and $e_3 = \{\mathrm{b}\} \times \{\mathrm{d}\}$. Then $e_2 \prec e_1$ and $e_3 \prec e_1$. Also, if $E = \{e_2, e_3\}$ then $\{e_2\} \subset E$ and $e_1 \preceq E$. In contrast, $\{e_1\} \not\subseteq E$.*

Remark 2.1.6 (edit redundancy). *Suppose that E is a set of edits of the form described in Eq. (11). It is not difficult to verify that an edit $e \in E$ is redundant if*

$$\begin{aligned} A_k &= \varnothing, \text{ for any } k \in 1, 2, \dots, n \qquad (17)\\ &\text{or}\\ e &\preceq e' \text{ with } e' \in E. \qquad (18) \end{aligned}$$

In (17), e is redundant since it cannot contain any records. It can be tested by checking if any $\mathbf{1}_{d_k} \cdot \mathbf{a}_k = 0$. In the case of (18), e is redundant because any edit violating e also violates e'. Using $\rho(e) = \mathbf{a}$ and $\rho(e') = \mathbf{a}'$, this can be tested by checking if $\mathbf{a} \wedge \mathbf{a}' = \mathbf{a}$ or equivalently if $\mathbf{a} \vee \mathbf{a}' = \mathbf{a}'$. □

Remark 2.1.7. *The boolean representation of records and edits a bijection. For convenience we use the symbols $\in$, $\prec$ and $\preceq$ for edits and record in set as well as in boolean representation. For example, in stead of writing $\mathbf{v} \cdot \mathbf{a} = n$ we write $\mathbf{v} \in \mathbf{a}$ (which is equivalent to $v \in e$) when convenient.* □

In the editrules package, the boolean representation is mainly used to store edits and to manipulate them through variable substitution and elimination. Data records can be stored in data.frame objects, as usual.

2.2 The editarray object

In the editrules package, a set of categorical edits is represented as an editarray object. Formally, we denote an editarray E for n categorical variables and m edits as (brackets indicate a combination of objects)

$$E = \langle \mathbf{A}, \mathbf{ind} \rangle, \text{ with } \mathbf{A} \in \{0,1\}^{m\times d} \text{ and } d = \sum_{k=1}^{n} d_k, \tag{19}$$

Each row $\mathbf{a}$ of $\mathbf{A}$ contains the boolean representation of one edit, and the d_k denote the number of categories of each variable. The object $\mathbf{ind}$ is a nested list which relates columns of $\mathbf{A}$ to variable names and categories. Labeling variables with $k \in 1, 2, \ldots, n$ and category values with $v_k \in 1, 2, \ldots, d_k$, we use the following notations:

$$\mathbf{ind}(k, v_k) = \sum_{l<k} d_l + v_k \tag{20}$$

$$\mathbf{ind}(k) = \{\mathbf{ind}(k, v_k) \mid v_k \in D_k\}. \tag{21}$$

So $\mathbf{ind}(k, v_k)$ is the column index in $\mathbf{A}$ for variable k and category v_k and $\mathbf{ind}(k)$ is the set of column indices corresponding to all categories of variable k. The editarray is the central object for computing with categorical edits, just like the editmatrix is the central object for computations with linear edits.

It is both tedious and error prone to define and maintain an editarray by hand. In practice, categorical edits are usually stated verbosely, such as: “a male subject cannot be pregnant”, or “an under-aged subject cannot be married”. To facilitate the definition of edit arrays, editrules is equipped with a parser, which takes R-statements in character format, and translates them to an editarray.

Figure 1 shows a simple example of defining an editarray with the editrules package. The first two edits in Figure 1 define the data model. The editarray function derives the datamodel based on the variable names and categories it finds in the edits, whether they are univariate (defining domains) or multivariate. This means that if all possible variables and categories are mentioned in the multivariate edits, the correct datamodel will be derived as well.

When printed to the screen, the boolean array is shown with column heads of the form

<abbreviated var. name><separator><abbreviated cat. label>

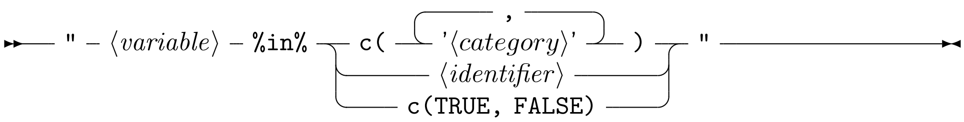
```
> E <- editarray(c(
+     "gender %in% c('male','female')",
+     "pregnant %in% c('yes','no')",
+     "if (gender == 'male') pregnant == 'no'"
+     )
+ )
> E

Edit array:
     levels
edits gndr:feml gndr:male prgn:no prgn:yes
   e1     FALSE      TRUE   FALSE     TRUE

Edit rules:
d1  :  gender %in% c('female', 'male')
d2  :  pregnant %in% c('no', 'yes')
e1  :  if( gender == 'male' ) pregnant != 'yes'

> datamodel(E)

  variable  value
1   gender female
2   gender   male
3 pregnant     no
4 pregnant    yes
```

Figure 1: Defining a simple editarray with the editarray function. The array is printed with abbreviated column heads, which themselves consist of variable names and categories separated by a colon (by default). When printed to screen, a character version of the edits is shown as well, for readability.

where both variable names and categories are abbreviated for readability, and the standard separator is a colon (:). The separator may not occur as a symbol in either variable or category name, and its value can be determined by passing a custom sep argument to editarray. For convenience, the function datamodel accepts an editarray as input and returns an overview of variables and their categories for easy inspection in the form of a data.frame.

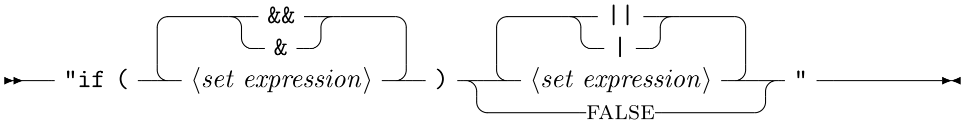
Internally, editarray uses the R internal parse function to transform the character expressions to a parse tree, which is subsequently traversed recursively to derive the entries of the editmatrix. The opposite is also possible. The R internal function as.character has been overloaded to derive a character representation from a boolean representation. When printed to the screen, both the boolean and textual representation are shown.

Univariate edits define the domain of a variable. The domains form together a data model. A domain can be defined with common R syntax using the %in% operator. If a domain is defined explicitly, it must follow the following syntax diagram.

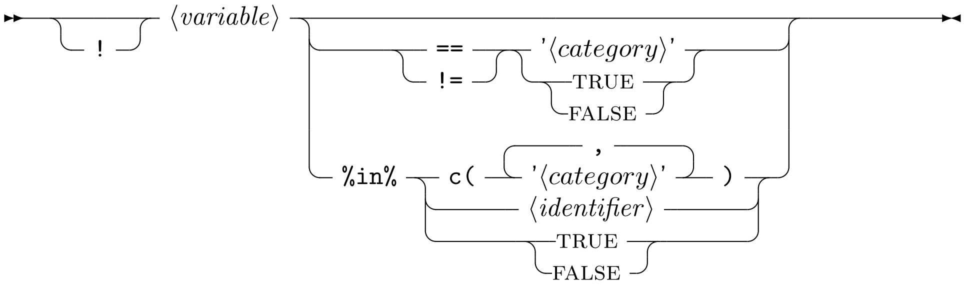


Here, ⟨*variable*⟩ is the name of a categorical variable, and ⟨*category*⟩ is a literal category name. Note that the category name is enclosed by single quotes while the entire statement is between double quotes. That is, the entire statement has to be offered in string format to editarray. The ⟨*identifier*⟩ is the name of a predefined character variable storing the unique categories for a variable. In principle, ⟨*identifier*⟩ may be replaced by any valid R symbol evaluating to a character or factor vector. However, such constructions are not recommended, since multivariate edit rules depend on a fixed datamodel.

The multivariate rules can be defined in two ways. The most useful and common way to define edits follows the following syntax diagram.



Where each ⟨*set expression*⟩ is a logical statement, following



The reader can check that the examples given in Figure 1 follow this syntax. The example below illustrates the use of separately defined data models and boolean categories.

```
> xval <- letters[1:4]
> yval <- c(TRUE,FALSE)
> ( M <- editarray(c(
+     "x %in% xval",
+     "y %in% yval",
+     "if ( x %in% c('a','b') ) !y "
+     )) )

Edit array:
     levels
edits  x:a  x:b   x:c   x:d y:FALS y:TRUE
   e1 TRUE TRUE FALSE FALSE  FALSE   TRUE

Edit rules:
d1  :  x %in% c('a', 'b', 'c', 'd')
d2  :  y %in% c(FALSE, TRUE)
e1  :  if( x %in% c('a', 'b') ) y == FALSE
```

The second way to define multivariate edits is based on rewriting on the basic classical logic law $P \Rightarrow Q = \neg P \vee Q$. It involves the following syntax diagram.

Where each ⟨*setexpression*⟩ is as in the syntax diagram above. In practice, a user will commonly not use this form since it is less readable. However, the as.character method for editarray can generate such statements by passing the argument useIf=FALSE, as shown below.

```
> as.character(M,useIf=FALSE)

                                      d1                                 d2
        "x %in% c('a', 'b', 'c', 'd')"               "y %in% c(FALSE, TRUE)"
                                      e1
"!( x %in% c('a', 'b') ) | y == FALSE"
```

The main advantage of this form is that contrary to the if() form, it allows for vectorized checking of edits, which is why it is used internally.

2.3 Coercion, checking, redundancy and feasibility

The editrules package is equipped with functions operating on sets of edits represented as an editarray. An overview is given in Table 1. The datamodel function extracts the variables and categories from an editarray, and returns them as a two-column data.frame. With as.data.frame or as.character one can coerce an editarray so that it can be written to a file or database. Character coercion is used when edits are printed to the screen. Optionally, coercing the datamodel to character form can be switched off. The result of as.data.frame has columns with edit names, a character representation of the edits and a column for remarks.

The function violatedEdits takes an editarray and a data.frame as input and returns a logical matrix indicating which record (rows) violate which edits (columns). It works by parsing the editarray to R-expressions and evaluating them within the data.frame environment. By default, the records are checked against the data model. This can be turned off by providing the optional argument datamodel=FALSE.

When manipulating edit sets, redundant edits of the form of Eq. (17) may arise. Such redundancies can be detected in the boolean representation with isObviouslyRedundant. By default, this function also checks for duplicate edits, but this may be turned off. The function duplicated is overloaded from the standard R function and the function isSubset (pseudocode in Algorithm 1) detects which edits are a subset or duplicate of another one. In the actual R implementation, the only explicit loop is a call to R's vapply function. The other loops are avoided using R's indexing and vectorization properties.

Manipulations may also lead to edits of the form $e = D$, in which case every possible record is invalid, and the editarray has become impossible to satisfy. The function isObviouslyInfeasible detects whether any such edits are

Table 1: Functions for objects of class editarray. Only mandatory arguments are shown, refer to the built-in documentation for optional arguments.

Function	description
datamodel(E)	get datamodel
getVars(E)	get a list of variables
as.data.frame(E)	coerce edits to data.frame
contains(E)	which edits contains which variable
as.character(E)	coerce edits to character vector
blocks(E)	Get list of independent blocks of edits
reduce(E)	Remove empty unnecessary variables and rows
isObviouslyRedundant(E)	find redundancies [Eq. (17)], duplicates
duplicated(E)	find duplicate edits
isSubset(E)	find edits, subset of another edit [Eq. (18)]
isObviouslyInfeasible(E)	detect simple contradictions
isFeasible(E)	detect if at least 1 valid record exists
substValue(E,var,value)	substitute a value
eliminate(E,var)	eliminate a variable (sect. 3.4)
violatedEdits(E,dat)	check which edits are violated by x
localizeErrors(E,dat)	localize errors (sect. 4.2)
errorLocalizer(E,x)	backtracker for error localization (sect. 4.3)
summary(E)	summarize the content of E
plot(E)	plot a graph of E (requires igraph package)

Algorithm 1 isSubset(E)

Input: An editarray $E = \langle \mathbf{A}, \mathbf{ind} \rangle$.

1: $\mathbf{s} \leftarrow (\text{FALSE})^m$
2: **for** $(\mathbf{a}^{(i)}, \mathbf{a}^{(i')}) \in \text{rows}(\mathbf{A}) \times \text{rows}(\mathbf{A})$ **do**
3:     **if** $\mathbf{a}^{(i)} \vee \mathbf{a}^{(i')} = \mathbf{a}^{(i')}$ **then**
4:         $s_i \leftarrow \text{TRUE}$

Output: Boolean vector $\mathbf{s}$ indicating which edits represented by $\mathbf{A}$ are a subset of another edit.

present. The function isFeasible checks if the set of edits in it's argument allows at least one valid record. This may yield results which are counter-intuitive at first glance. For example, consider set of edits on the domain $D = \{(x, y) \in \{\mathsf{a}, \mathsf{b}\} \times \{\mathsf{c}, \mathsf{d}\}\}$.

```
> M <- editarray(c(
+     "x %in% c('a','b')",
+     "y %in% c('c','d')",
+     "if ( x == 'a' ) y == 'c'",
+     "if ( x == 'a' ) y != 'c'"))
>
```

This set of edits is feasible, even though edits e_1 and e_2 seem to contradict each other:

```
> isFeasible(M)

[1] TRUE
```

The explanation is that e_1 and e_2 contradict each other only when $x = \mathsf{a}$, so

```
> isFeasible(substValue(M,'x','a'))

[1] FALSE
```

where the function substValue is discussed in the next section. One can check that the record $(x = \mathsf{b}, y = \mathsf{d})$ indeed satisfies all edits in M.

The feasibility check works by eliminating all variables in an editarray one by one until either no edits are left or an obvious contradiction is found. Eliminating all variables amounts constructing the solution of an error localization problem in the branch-and-bound algorithm of De Waal (2003) where all variables have to be adapted. Variable elimination is discussed further in the next section while error localization is discussed in Section 4.

3 Manipulation of categorical restrictions

The basic operations on sets of categorical edits are value substitution and variable elimination. The former amounts to adapting the datamodel underlying the edit set while the latter amounts to deriving relations between variables not involving the eliminated variable.

In the next subsection we give an example of value substitution with the editrules package, as well as some background. In subsection 3.2 we describe a variable elimination method, which to the authors seems new to the field of data editing. In subsection 3.3 it is shown that the method yields results equivalent to Fellegi and Holt's elimination method. Finally, in subsection 3.4 we give an example of eliminating variables with the editrules package.

3.1 Value substitution

If it is assumed that in a record, one of the variables takes a certain value, that value may be substituted in the edit rules. In the boolean representation this amounts to removing all edits which exclude that value, since the record cannot violate those edits. Secondly, the columns related to the substituted variable but not to the substituted category are removed, thereby adapting the datamodel to the new assumption. Algorithm 2 gives the pseudocode for reference purposes.

In the editrules package, value substitution is performed by the substValue function, which accepts an editarray, a variable name and a category name. In the following example the editmatrix defined in Figure 1 is used.

```
> substValue(E,"gender","female")
```

Algorithm 2 $\textsc{substValue}(E,k,v)$

Input: an editarray $E = \langle \mathbf{A}, \mathbf{ind}\rangle$, a variable index k and a value v

1: $i \leftarrow \mathbf{ind}(k, v)$
2: $\mathbf{A} \leftarrow \mathbf{A}\backslash\{\mathbf{a} \in \text{rows}(\mathbf{A}) \,|\, a_i = \text{FALSE}\}$ ▷ Remove rows not involving v
3: $\mathbf{A} \leftarrow \mathbf{A}\backslash\{\mathbf{a}^t_j \in \text{columns}(\mathbf{A}) \,|\, j \in \mathbf{ind}(k)\backslash i\}$ ▷ Remove categories $\neq v$
4: Update **ind**

Output: $\langle \mathbf{A}, \mathbf{ind}\rangle$ with v substituted for variable k.

```
Edit array:
      levels
edits  gndr:feml gndr:male prgn:no prgn:yes

Edit rules:
d1  :  gender %in% c('female', 'male')
d2  :  pregnant %in% c('no', 'yes')
```

In this case, the variable *gender* is substituted by the value female. The rules concerning *gender* = male may be deleted, so here only the datamodel is left without any multivariate rules. In fact, the datamodel itself may be reduced, which can be achieved by setting the option reduce=TRUE.

```
> substValue(E,"gender","female",reduce=TRUE)

Edit array:
      levels
edits  gndr:feml prgn:no prgn:yes

Edit rules:
d1  :  gender %in% 'female'
d2  :  pregnant %in% c('no', 'yes')
```

3.2 Variable elimination by category resolution

Given two edits e and e', with boolean representations $\mathbf{a}$ and $\mathbf{a}'$ respectively. We define the *resolution operator* $\mathfrak{R}_k$ as:

$$\begin{aligned}\mathbf{a}\,\mathfrak{R}_k\,\mathbf{a}' = \mathbf{a}_1 \wedge \mathbf{a}'_1 \oplus \cdots \oplus \mathbf{a}_{k-1} \wedge \mathbf{a}'_{k-1}\\ \oplus \quad \mathbf{a}_k \vee \mathbf{a}'_k \oplus \mathbf{a}_{k+1} \wedge \mathbf{a}'_{k+1} \oplus \cdots \oplus \mathbf{a}_n \wedge \mathbf{a}'_n.\end{aligned} \tag{22}$$

For two edit sets $\mathbf{A}$ and $\mathbf{A}'$, we also introduce the notation

$$\mathbf{A}\,\mathfrak{R}_k\,\mathbf{A}' = \{\mathbf{a}\,\mathfrak{R}_k\,\mathbf{a}' \,|\, (\mathbf{a}, \mathbf{a}') \in \text{rows}(\mathbf{A}) \times \text{rows}(\mathbf{A}')\}. \tag{23}$$

Remark 3.2.1. *The resolution operator has the following properties relevant for record checking.*

$$\textit{if } \mathbf{v} \in \mathbf{a}\,\mathfrak{R}_k\,\mathbf{a}' \quad \textit{then} \quad \mathbf{v} \in \mathbf{a} \vee \mathbf{v} \in \mathbf{a}' \tag{24}$$

$$\textit{if } \mathbf{v} \in \mathbf{a} \quad \textit{then} \quad \big(\mathbf{v} \in \mathbf{a}\,\mathfrak{R}_k\,\mathbf{a}'\big) \vee \big(\mathbf{a}\,\mathfrak{R}_k\,\mathbf{a}' = \varnothing\big)\,, \tag{25}$$

where we used notation as defined in remark 2.1.7. That is, if a record violates $\mathbf{a}\,\mathfrak{R}_k\,\mathbf{a}'$*, it does so because it violates* $\mathbf{a}$ *and/or* $\mathbf{a}'$*. Therefore,* $\mathbf{a}\,\mathfrak{R}_k\,\mathbf{a}'$ *is also an edit in the sense that a record is invalid if it falls in the derived edit. When* $\mathbf{a}_k = \mathbf{a}'_k$*, the resulting edit is the intersection of the original edits, in which case the resulting edit is redundant.* □

The operator is called resolution operator since in certain cases its action is equivalent to a resolution operation from formal (automated) logic derivation (Robinson, 1965). If $\mathbf{a}_k \vee \mathbf{a}'_k = (\text{TRUE})^{d_k}$, the operator "resolves" or eliminates the k^{th} variable and we are left with a relation between the other variables, regardless of the value of variable k. The edit resulting from a resolution operation on two explicitly defined edits is called an *implied edit*. If the resolution operation happens to eliminate one of the variables, it is called an *essentially new implied edit*. These terms were introduced by Fellegi and Holt (1976) who first solved the problem of error localization for categorical data.

Example 3.2.2. *Consider again the variable domain of Example 2.1.1. We define the edits*

$e^{(1)}$: *Under aged people cannot be married.*
$e^{(2)}$: *A marriage partner must be married.*

In the boolean representation we get:

$$\begin{aligned}\rho(e^{(1)}) &= (1,0,0,0)\oplus(1,0)\oplus(1,1,1)\\ \rho(e^{(2)}) &= (0,1,1,1)\oplus(1,1)\oplus(0,1,1).\end{aligned}$$

A new edit can be derived, using $\mathfrak{R}_1$

$$\rho(e^{(1)})\,\mathfrak{R}_1\,\rho(e^{(2)}) = (1,1,1,1)\oplus(1,0)\oplus(0,1,1).$$

Which can be translated to the rule that under-aged people cannot be a marriage partner.

The resolution operator can be used to eliminate a variable k from a set of edits (represented by $\mathbf{A}$) category by category as follows (Algorithm 3). Suppose that j is the column index of the first category of k. Collect all pairs of $(\mathbf{a}^+, \mathbf{a}^-)$ obeying $a_j^+ = \text{TRUE}$ and $a_j^- = \text{FALSE}$. If there are no edits of type $\mathbf{a}^+$, the variable cannot be eliminated and the empty set is returned. Otherwise, copy all $\mathbf{a}^+$ to a new set of edits and add every $\mathbf{a}^+\,\mathfrak{R}_k\,\mathbf{a}^-$. By construction, these new edits all have $a_j = \text{TRUE}$. Possibly, redundant edits have been produced, and these may be removed. The procedure is iterated for every category of k, adding a category for which each $a_j = \text{TRUE}$ at each iteration.

Algorithm 3 ELIMINATE(E,k)

Input: an editarray $E = \langle \mathbf{A}, \mathbf{ind} \rangle$, a variable index k

for $j \in \mathbf{ind}(k)$ **do**
  $\mathbf{A}^{+} = \{\mathbf{a} \in \text{rows}(\mathbf{A}) \,:\, a_j = \text{TRUE}\}$
  $\mathbf{A}^{-} = \{\mathbf{a} \in \text{rows}(\mathbf{A}) \,:\, a_j = \text{FALSE}\}$
  if $\mathbf{A}^{+} = \varnothing$ **then**
    $\mathbf{A} \leftarrow \varnothing$
    break
  $\mathbf{A} \leftarrow \mathbf{A}^{+} \cup \mathbf{A}^{+} \,\mathfrak{R}_k\, \mathbf{A}^{-}$
  Delete rows which have ISSUBSET($\langle \mathbf{A}, \mathbf{ind} \rangle$) = TRUE.

Output: editarray $\langle \mathbf{A}, \mathbf{ind} \rangle$ with variable k eliminated

3.3 Some properties of the elimination method

In this subsection we prove that the function ELIMINATE of Algorithm 3 generates all edits necessary to solve the error localization problem. A short comparison with the elimination method of Fellegi and Holt (1976) is given as well.

Given a set of m edits:

$$E = \{e^{(i)} \in \mathcal{P}(D) \,|\, e^{(i)} \text{ as in Eq. (11)}, i \in 1, 2, \ldots, m\}, \tag{26}$$

where $\mathcal{P}(D)$ is the power set of D. Fellegi and Holt (1976), but also De Waal et al. (2011) define a way to derive new edits, which may be written as a function F_k,

$$\begin{aligned} F_k(E) = {}& \cap_{i=1}^{m} A_1^{(i)} \times \cap_{i=1}^{m} A_2^{(i)} \times \cdots \times \cap_{i=1}^{m} A_{k-1}^{(i)} \times \\ & \times \quad \cup_{i=1}^{m} A_k^{(i)} \times \cap_{i=1}^{m} A_{k+1}^{(i)} \times \cdots \times \cap_{i=1}^{m} A_m^{(i)}, \end{aligned} \tag{27}$$

where k is called the *generating variable*. In the boolean representation, we have $\mathbf{A} = \rho(E)$. Using the relations $\rho(e \cap e') = \rho(\mathbf{e}) \wedge \rho(\mathbf{e})$ and $\rho(e \cup e') = \rho(e) \vee \rho(e')$ we may write

$$F_k(\mathbf{A}) = \mathbf{a}^{(1)} \,\mathfrak{R}_k\, \mathbf{a}^{(2)} \,\mathfrak{R}_k \cdots \mathfrak{R}_k\, \mathbf{a}^{(m)}, \quad \mathbf{a}^{(i)} = \rho\left(e^{(i)}\right), \tag{28}$$

where we used some obvious properties (symmetry, associativity) of the $\wedge$ and $\vee$ operators as well. The following lemma and corollary show that we can do all the work, necessary for implied edit derivation with the resolution operator.

Lemma 3.3.1 (Fellegi and Holt (1976))**.** *If E is a set of edits, every edit, logically implied by E can be derived by repeated application of Eq.* (27) *on subsets of E.*

Proof. The proof is given in the reference and will not be repeated here. □

Corollary 3.3.2. *If E is a set of edits, all implied edits can be derived by repeated application of the resolution operator on elements of the boolean representation of E.*

Proof. This follows from the equivalence between Eqs. (27) and (28). □

Having established the use of the resolution operator, it becomes interesting to study its algebraic properties. By substitution in the definition, it is easily shown that the resolution operator is symmetric, associative and idempotent. As a reminder, these properties are defined as follows.

$$\begin{array}{ll} \text{symmetry:} & \mathbf{a}\,\mathfrak{R}_k\,\mathbf{b} = \mathbf{b}\,\mathfrak{R}_k\,\mathbf{a} \\ \text{associativity:} & (\mathbf{a}\,\mathfrak{R}_k\,\mathbf{b})\,\mathfrak{R}_k\,\mathbf{c} = \mathbf{a}\,\mathfrak{R}_k(\mathbf{b}\,\mathfrak{R}_k\,\mathbf{c}) \\ \text{idempotency:} & \mathbf{a}\,\mathfrak{R}_k\,\mathbf{a} = \mathbf{a}. \end{array} \tag{29}$$

The resolution operator (although not called as such) was found earlier and independently of the current authors by Willenborg (1988), who also discovered these properties. The following property shows that the resolution operator leaves redundancy relations untouched.

Lemma 3.3.3. *If $\mathbf{b} \preceq \mathbf{c}$, then $\mathbf{a}\,\mathfrak{R}_k\,\mathbf{b} \preceq \mathbf{a}\,\mathfrak{R}_k\,\mathbf{c}$.*

Proof. We may write $\mathbf{a} = \mathbf{a}_k \oplus \mathbf{a}'$ and similarly for $\mathbf{b}$ and $\mathbf{c}$. This gives

$$\begin{aligned} \mathbf{a}\,\mathfrak{R}_k\,\mathbf{b} &= \mathbf{a}\,\mathfrak{R}_k(\mathbf{b} \wedge \mathbf{c}) \\ &= \mathbf{a}_k \vee (\mathbf{b}_k \wedge \mathbf{c}_k) \oplus \mathbf{a}' \wedge (\mathbf{b}' \wedge \mathbf{c}') \\ &= (\mathbf{a}_k \vee \mathbf{b}_k) \wedge (\mathbf{a}_k \vee \mathbf{c}_k) \oplus (\mathbf{a}' \wedge \mathbf{b}') \wedge (\mathbf{a}' \wedge \mathbf{c}') \\ &= \mathbf{a}\,\mathfrak{R}_k\,\mathbf{b} \wedge \mathbf{a}\,\mathfrak{R}_k\,\mathbf{c} \preceq \mathbf{a}\,\mathfrak{R}_k\,\mathbf{c}, \end{aligned}$$

and we are done. □

This lemma is important because it shows that removing redundant edits (as shown in Algorithm 3) does not affect the outcome of ELIMINATE in the sense that the resulting edit set covers the same subset of D with or without the redundancy removal step.

We now define more formally the notion of variable elimination.

Definition 3.3.4. *Given a function $f : \mathcal{P}(D) \to \mathcal{P}(D)$, and a set of edits $E \subset \mathcal{P}(D)$. If none of the edits in $f(E)$ contain variable k, we say that f eliminates that variable from E.*

Remember that in the boolean representation, this means that $\mathbf{a}_k = (\text{TRUE})^{d_k}$. The following theorem shows that every essentially new implied edit, generated by k is found by Algorithm 3.

Theorem 3.3.5. *If E is a set of edits, the function ELIMINATE(E, k) generates every edit derived from E from which variable k has been eliminated. Moreover, these edits are mutually non-redundant.*

Proof. The rows are mutually non-redundant since redundant rows are removed explicitly in line 8 of the algorithm. The fact that the removing these rows does not alter the result (in the sense that the resulting edits will cover the same subdomain of D) is a consequence of Lemma 3.3.3.

Denote by $\mathbf{A}(j)$ the state of $\mathbf{A}$ after j iterations. We have

$$\mathbf{A}(1) = \mathbf{A}^{+}(0) \cup \mathbf{A}^{+}(0)\,\mathfrak{R}_k\,\mathbf{A}^{-}(0). \tag{30}$$

Here $\mathbf{A}(1)$ contains every nonredundant derived edit with column $\mathbf{ind}(k,1)$ equal to TRUE . For, if there is another derived edit, it must be in $\mathbf{A}^{+}(0)\,\mathfrak{R}_k\,\mathbf{A}^{+}(0)$, since $\mathbf{A}^{-}(0)\,\mathfrak{R}_k\,\mathbf{A}^{-}(0)$ only generates edits where column number $\mathbf{ind}(k,1)$ is FALSE . Now, consider two edits $\mathbf{a}^{+}$ and $\mathbf{a}'^{+} \in \mathbf{A}^{+}(0)$. We have $\mathbf{a}^{+}\,\mathfrak{R}_k\,\mathbf{a}'^{+} \preceq \mathbf{a}^{+} \cup \mathbf{a}'^{+} \preceq \mathbf{A}^{+}(0) \preceq \mathbf{A}(1)$ so each element of $\mathbf{A}^{+}(0)\,\mathfrak{R}_k\,\mathbf{A}^{+}(0)$ is redundant (remember Remarks 2.1.2 and 2.1.7). Now consider$\mathbf{A}(j)$. It follows from the above that if $\mathbf{A}(j-1)$ contains every nonredundant edit with columns $\mathbf{ind}(k,1),\ldots,\mathbf{ind}(k,j-1)$ equal to TRUE then $\mathbf{A}(j)$ contains all nonredundant derived edits where columns $\mathbf{ind}(k,1),\ldots,\mathbf{ind}(k,j)$ equal TRUE . Since the algorithm iterates over all $j \in \mathbf{ind}(k)$ the result follows. □

It is interesting to compare the procedure in Algorithm 3 with the procedure of Fellegi and Holt (1976) for generating implied edits. This procedure, which is also described by De Waal et al. (2011) may be summarized as follows.

1: **function** FH(E,k)
2: Find every $E_j \subseteq E$, with $j \in 1, 2, \ldots, s$ such that
- $F_k(E_j)$ eliminates k.
- E_j is minimal in the sense that there is no $E'_j \subset E$ such that $F_j(E'_j)$ eliminates k.
- The E_j are mutually non-redundant in the sense $F_k(E_j) \not\subseteq F_k(E_l)$.

3: $E \leftarrow \cup_{j=1}^{s} F_k(E_j)$
4: **return** E

Here, E is a set of categorical edits in some form, and k the variable to eliminate. Most of the computational complexity is contained in line 2, where the search space is determined by the power set of E, yielding exponential complexity in the number of edits.

The complexity of the ELIMINATE algorithm is determined by the 7th line in Algorithm 3, which is quadratic in the current number of edits in the for-loop. This recurrence relation also yields exponential complexity in the number of edits. However, by removing the redundant edits at every iteration (a quadratic operation in itself), the actual number of edits can be kept to a minimum which reduces the complexity encountered in practice.

3.4 An example with eliminate

The purpose of the eliminate function is to derive all possible non-redundant edits from an edit set that do not contain a certain variable. For categorical data edits, this amounts to logical resolution. For example, consider the syllogism which was also discussed in Example 3.2.2:

P_1	Under-aged people cannot be married
P_2	A marriage partner has to be married
C	An under-aged subject cannot be a marriage partner.

Here, the conclusion C is derived from premises P_1 and P_2 by eliminating *marital status*. In editrules the above operation can be performed as follows. We first define a data model and edit rules:

```
> E <- editarray(c(
+     "age %in% c('under-aged','adult')",
+     "maritalStatus %in% c('married','not married')",
+     "positionInHousehold %in% c('partner','child','other')",
+     "if (age == 'under-aged') maritalStatus != 'married'",
+     "if (positionInHousehold == 'partner') maritalStatus == 'married'"
+ ))
```

We may derive the conclusion by eliminating the *marital status* variable:

```
> eliminate(E,'maritalStatus')

Edit array:
     levels
edits age:adlt age:und- mrtS:mrrd mrtS:ntmr psIH:chld psIH:othr psIH:prtn
   e1    FALSE     TRUE      TRUE      TRUE     FALSE     FALSE      TRUE

Edit rules:
d1  :  age %in% c('adult', 'under-aged')
d2  :  maritalStatus %in% c('married', 'not married')
d3  :  positionInHousehold %in% c('child', 'other', 'partner')
e1  :  if( age == 'under-aged' ) positionInHousehold != 'partner'
```

This indeed yields the right conclusion. Alternatively, we may eliminate *age*:

```
> eliminate(E,'age')

Edit array:
     levels
edits age:adlt age:und- mrtS:mrrd mrtS:ntmr psIH:chld psIH:othr psIH:prtn
   e1     TRUE     TRUE     FALSE      TRUE     FALSE     FALSE      TRUE

Edit rules:
d1  :  age %in% c('adult', 'under-aged')
d2  :  maritalStatus %in% c('married', 'not married')
d3  :  positionInHousehold %in% c('child', 'other', 'partner')
e1  :  if( maritalStatus == 'not married' ) positionInHousehold != 'partner'
```

This deletes the only rule actually involving *age*. That is, no new rules not involving *age* can be derived.

4 Error localization in categorical data

4.1 A Branch and bound algorithm

The editrules package implements an error localization algorithm, based on the branch-and-bound algorithm of De Waal and Quere (2003). The algorithm has been extensively described in De Waal (2003) and De Waal et al. (2011). The algorithm is similar to the branch-and-bound algorithm used for error localization in numerical data in the editrules package as described in De Jonge and Van der Loo (2011), except that the elimination and substitution subroutines are implemented for categorical data.

In short, a binary tree is created with the full set of edits and an erroneous record at the root node. Two child nodes are created. In the first child node the first variable of the record is assumed correct, and it's values is substituted in the edits. In the second child node the variable is assumed incorrect and it is eliminated from the set of edits. The tree is continued recursively until choices are made for each variable. Branches are pruned when they cannot lead to a solution, leaving a partial binary tree where each path from root to leaf represents a solution to the error localization problem. Computational complexity is reduced further by pruning branches leading to higher-weight solutions then solutions found earlier.

Recall the datamodel of Example 2.1.1, with variables *marital status*, *age* and *position in household*. We define the following two edits:

$e^{(1)}$	An under-aged subject cannot be married
$e^{(2)}$	A (marriage) partner has to be married

As an example we treat the following record with the branch-and-bound algorithm to localize the errors:

$$v = (\mathsf{married}, \mathsf{under\text{-}aged}, \mathsf{partner}). \tag{31}$$

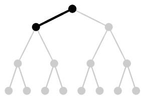
At the beginning of the algorithm, only the root node is filled. The situation may be represented as follows:

$\mathbf{v}$	1	0	0	0	1	0	1	0	0
$\mathbf{a}^{(1)}$	1	0	0	0	1	0	1	1	1
$\mathbf{a}^{(2)}$	0	1	1	1	1	1	1	0	0

Root node, $w = 0$,

where $\mathbf{v} = \rho(v)$, and $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$ are the boolean representations of $e^{(1)}$ and $e^{(2)}$ respectively. The record and edits are denoted in boolean representation as shown in Example 2.1.1. The weight w counts the number of variables that are assumed to be incorrect, which at the root node is zero.

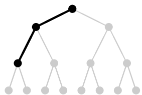
The tree is traversed in depth-first fashion. In the first step, we substitute $\mathsf{married}$ in *marital status*, yielding



$\mathbf{v}$	1				1	0	1	0	0
$\mathbf{a}^{(1)}$	1				1	0	1	1	1

Subst. mar. stat., $w = 0$.

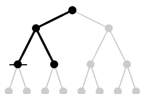
Here, $\mathbf{a}^{(2)}$ is removed, since it has no meaning for $\mathbf{v}$ anymore. The positions for the categories unmarried, widowed and divorced are left empty here to signify that the datamodel has a fixed marital status now. The dark part of the tree on the left shows which nodes have been treated. Continuing we find



$\mathbf{v}$	1				1	1	0	0
$\mathbf{a}^{(1)}$	1				1	1	1	1

Subst. age., $w = 0$.

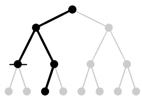
At this point we have fixed the value for *marital status* and *age*. It can be seen from the value of $\mathbf{a}^{(1)}$ for *position in household* that no matter what value is chosen for that field, the value $\mathbf{v} \cdot \mathbf{a}^{(1)} = 3$. This shows that this path will never lead to a valid solution. We therefore prune the tree here, go up one node and turn right.



$\mathbf{v}$	1		1 0 0

Elim. age, $w = 1$.

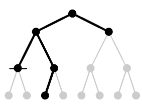
Eliminating the *age* variable yields an empty edit set. We may continue down and substitute the value partner for *position in household*.



$\mathbf{v}$	1		1

Subst. hh. pos., $w = 1$.

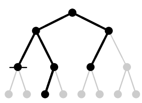
This yields the first solution: only the age variable needs to be changed. In search for more solutions, we move up the tree and try to eliminate *position in household*. However, since eliminating *position in household* would increase the weight to 2 we will prune the tree at this point. Moving up to the root node and eliminating *marital status* gives



$\mathbf{v}$					1	0	1	0	0
$\mathbf{a}^{(3)}$	1	1	1	1	1	0	1	0	0

Elim. mar. stat., $w = 1$.

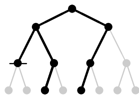
Here $\mathbf{a}^{(3)} = \mathbf{a}^{(1)} \,\mathfrak{R}_1\, \mathbf{a}^{(2)}$. It is interpreted as the rule that under-aged people cannot be a partner in the household (no matter what the value of *marital status* is). Creating the next child node by substituting *age*, we get



$\mathbf{v}$					1	1	0	0
$\mathbf{a}^{(3)}$	1	1	1	1	1	1	0	0

Subst. age, $w = 1$.

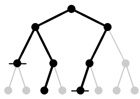
Going down the tree and substituting *position in household* yields



$\mathbf{v}$					1	1
$\mathbf{a}^{(3)}$	1	1	1	1	1	1

Subst. hh. pos., $w = 1$.

However, whatever value we would choose for *marital status*, it would always result in an erroneous record since $\mathbf{a}^{(3)}$ has TRUE on all categories of that variable. Therefore, we go up one step in the tree. Eliminating *position in household* would increase the weight to 2, but since we already found a solution with weight equal to 1, this path need not be followed. We go up another node and bound on the fact that eliminating *age* would yield the same problem. The final tree may be represented as follows:



Here, every evaluated node is colored black, and a node is crossed when a bound condition was encountered. The only (minimal) solution created is represented by the path substitute *marital status* → eliminate *age* → substitute *position in household*. This corresponds to the solution where *age* has to be altered to fix the record, and indeed changing *age* from under-aged to adult will make the record fully valid. Note that the branch-and-bound algorithm reduced the number of nodes to be evaluated from 15 to 8 in this case.

4.2 Error localization with localizeErrors

The function localizeErrors applies the branch-and-bound algorithm to determine the minimal weight error location for every record in a data.frame. The columns may be in character or factor format. The function has an identical interface for numerical data under linear edits and categorical data under categorical edits. It is implemented as an S3 generic function, accepting either an editmatrix or an editarray as the first argument and a data.frame as the second argument. Further arguments are a vector of variable weights, a maximum search time (in seconds) to spend on a single record, a maximum weight and the maximum number of variables which may be changed. The latter two arguments introduce extra bound conditions in the branch-and-bound algorithm.

Even when variables are weighted, the solution to the error localization problem may not be unique. In those cases localizeErrors will draw uniformly from the set of lowest-weight solutions. The degeneracy (number of equivalent solutions found) is reported in the output.

The result of a call to localizeErrors is an object of class errorLocation. It contains a boolean matrix with error locations for each record as well as a status report containing degeneracies, solution weights run times and whether the maximum runtime was exceeded. It also contains a timestamp

```
> E <- editarray(c(
+     "age %in% c('under-aged','adult')",
+     "maritalStatus %in% c('unmarried','married','widowed','divorced')",
+     "positionInHousehold %in% c('marriage partner', 'child', 'other')",
+     "if( age == 'under-aged' ) maritalStatus == 'unmarried'",
+     "if(positionInHousehold == 'marriage partner') maritalStatus == 'married'"
+     )
+ )
> (dat <- data.frame(
+     maritalStatus=c('married','unmarried','widowed' ),
+     age = c('under-aged','adult','adult' ),
+     positionInHousehold=c('marriage partner','other','marriage partner')
+ ))

  maritalStatus        age positionInHousehold
1       married under-aged    marriage partner
2     unmarried      adult               other
3       widowed      adult    marriage partner

> set.seed(1)
> localizeErrors(E,dat)

Object of class 'errorLocation' generated at Wed Nov 16 15:25:27 2011
call : localizeErrors(E, dat)
slots: $adapt $status $call $user $timestamp

Values to adapt:
      adapt
record maritalStatus   age positionInHousehold
     1         FALSE  TRUE               FALSE
     2         FALSE FALSE               FALSE
     3         FALSE FALSE                TRUE

 Status:
  weight degeneracy  user system elapsed maxDurationExceeded
1      1          1 0.004  0.000   0.003               FALSE
2      0          1 0.000  0.004   0.002               FALSE
3      1          2 0.012  0.000   0.009               FALSE
```

Figure 2: Localizing errors in a data.frame of records. The data model is as defined in Example 2.1.1. The randseed is set before calling localizeErrors to make results reproducible. The third record has degeneracy 2, which means that the chosen solution was drawn uniformly from two equivalent solutions with weight 1.

(in the form of a Date object) and the name of the user running R. Table 2 gives an overview of the slots involved.

In Figure 2 an example of the use of localizeErrors is given. The data model and rules are as in subsection 4.1. The records are given by

```
  maritalStatus        age positionInHousehold
1       married under-aged    marriage partner
2     unmarried      adult               other
3       widowed      adult    marriage partner
```

The edits state that under-aged persons cannot be married and that one cannot be a marriage partner if one is unmarried. Clearly, the first and

Table 2: Slots in the errorLocation object

Slot	description.
$adapt	boolean array, stating which variables must be adapted for each record.
$status	A data.frame, giving solution weights, number of equivalent solutions, timings and whether the maximum search time was exceeded.
$user	Name of user running R during the error localization
$timestamp	date() at the end of the run.
$call	The call to localizeErrors

third record disobey some of these rules while the second record is valid. The first record can be repaired by adapting age and the second record can be made consistent by changing either *position in household* or *marital status*. In the latter case, both solutions have equal weight and localizeErrors has drawn one solution.

4.3 Error localization with errorLocalizer

Just like for linear edits, the function errorLocalizer gives more control over the error localization process since it allows to parameterize the search separately for each record. This can be useful, for example when reliability weights are calculated for each record.

The errorLocalizer function is described extensively in De Jonge and Van der Loo (2011), so here we will discuss the example shown in Figure 3.

The data model and edits are the same as in Figure 2. The difference here is that a record must be offered as a named character vector. A call to errorLocalizer generates a backtracker object which contains all information necessary to start searching the binary tree. After calling $searchNext() the weight and first found solution are returned, while the backtracker object stores some meta-information about the process, most significantly the duration of the search. A second call yields an equivalent solution and the third call returns NULL, indicating that all minimal weight solutions have been found.

```
> # Define a record
> r <- c(age = 'under-aged', maritalStatus='married', positionInHousehold='child')
> el <- errorLocalizer(E,r)
> el$searchNext()

$w
[1] 1

$adapt
               age       maritalStatus positionInHousehold
             FALSE                TRUE               FALSE

> el$duration

   user  system elapsed
  0.004   0.000   0.003

> el$maxdurationExceeded

[1] FALSE

> el$searchNext()

$w
[1] 1

$adapt
               age       maritalStatus positionInHousehold
              TRUE               FALSE               FALSE

> el$searchNext()

NULL
```

Figure 3: Finding errors with errorLocalizer. The data model and edits in E are as in Figure 2.

5 Conclusions

This paper describes the theory and implementation of categorical edit manipulation of the editrules package. Categorical restrictions may be defined textually in standard R syntax. New edits may be derived with the resolution method. A new formulation of the elimination method in terms of the resolution operator was developed which facilitated the development of a fast elimination algorithm which seems to be new in the field of data editing.

The package offers functionality to check records against rules and can determine the location of errors based on the generalized principle of Fellegi and Holt.

References

De Jonge, E. and M. Van der Loo (2011). Manipulation of linear edits and error localization with the editrules package. Technical Report 201120, Statistics Netherlands, The Hague.

De Waal, T. (2003). *Processing of erroneous and unsafe data.* Ph. D. thesis, Erasmus University Rotterdam.

De Waal, T., J. Pannekoek, and S. Scholtus (2011). *Handbook of statistical data editing and imputation.* Wiley handbooks in survey methodology. John Wiley & Sons.

De Waal, T. and R. Quere (2003). A fast and simple algorithm for automatic editing of mixed data. *Journal of Official Statistics 19*, 383–402.

Fellegi, I. P. and D. Holt (1976). A systematic approach to automatic edit and imputation. *Journal of the Americal Statistical Association 71*, 17–35.

Husson, F., S. Lê, and J. Pagès (2010). *Exploratory Multivariate Analysis by Example Using R.* Computer Sciences and Data Analysis. Chapman & Hall/CRC.

Robinson, J. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM 12*, 23–41.

Willenborg, L. (1988). *Computational aspects of data processing*, Volume 54 of *CWI Tract.* Amsterdam: Centre for Mathematics and Computer Science.

Winkler, W. E. (1999). State of statistical data editing and current research problems. In *Working paper no. 29. UN/ECE Work Session on Statistical Data editing*, Rome.

A Notation

Symbol	Explanation and reference
$\oplus$	Direct vector sum.
$\mathbf{a}$	An edit, in boolean representation: $\mathbf{a} = \rho(e)$, Eq. (13).
$\mathbf{a}_k$	Boolean representation of one variable in $\mathbf{a}$. $\mathbf{a} = \oplus_{k=1}^{n}\mathbf{a}_k$.
$\mathbf{A}$	Set of edits, in $m \times d$ boolean representation.
D	Set (domain) of all possible categorical records, Eq. (1).
D_k	Set of possible categories for variable k. Eq. (2).
d	Number of categories (in total), Eq. (6).
d_k	Number of categories in D_k.
e	An edit, in set representation: $e \subseteq D$, [Eq. (11)].
E	An editarray, Eq. (19), or a set of edits in set representation.
ind	Function relating categories c of variable k to columns in $\mathbf{A}$, Eqs.(20) and (21).
i	Row index in $\mathbf{A}$ (labeling edits).
j	Column index in $\mathbf{A}$ (labeling categories).
k	Labels a categorical variable.
m	Number of edits.
n	Number of variables.
$\mathfrak{R}_k$	Resolution operator Eq. (22).
ρ	Map, sending set representation to boolean representation.
v	Categorical record, in set representation: $v = (v_1, \ldots, v_n) \in D$.
v_k	Label for a single category of D_k.
$\mathbf{v}$	Categorical record, in boolean representation: $\mathbf{v} = \rho(v)$.
$\mathbf{v}_k$	Boolean representation of a single variable $\mathbf{v} = \oplus_{k=1}^{n}\mathbf{v}_k$.

Index