Abstract
R topics documented:

'pkg.R' 'plot.R' 'plot_errorLocation.R' 'print.R' 'reduce.R'
'removeRedundant.R' 'softEdits.R' 'str.R' 'subsetting.R'
'substValue.R' 'summary.R' 'violatedEdits.R' 'writeELAsMip.R'
'zzz.R'

NeedsCompilation: no

Repository: CRAN

Date/Publication: 2013-07-03 16:35:58

R topics documented:

editrules-package ................................................. 3
adjacency .......................................................... 4
as.editmatrix ....................................................... 8
as.editset ......................................................... 9
as.matrix.editarray ............................................... 9
as.matrix.editmatrix ............................................. 12
backtracker ....................................................... 15
blocks ............................................................... 16
c.editset ........................................................ 18
checkDatamodel ................................................... 20
condition .......................................................... 21
datamodel .......................................................... 21
disjunct ............................................................. 22
echelon .............................................................. 23
editfile ............................................................. 24
editnames .......................................................... 24
editrules.plotting ............................................... 25
edits ............................................................... 29
editType ............................................................ 29
eliminate .......................................................... 29
errorLocalizer ..................................................... 32
errorLocalizer.mip ............................................... 36
termErrorLocation ............................................... 38
generateEdits ..................................................... 41
getA ............................................................... 42
getAb .............................................................. 43
getb ............................................................... 45
geth ............................................................... 46
getOps ............................................................. 47
getVars ............................................................. 48
is.editrules ....................................................... 49
isFeasible ......................................................... 50
isNormalized ..................................................... 50
isObviouslyInfeasible ........................................... 51
isObviouslyRedundant ........................................... 52
isSubset .......................................................... 53
localizeErrors .................................................... 53
The `editrules` package aims to provide an environment to conveniently define, read and check recordwise data constraints including

- Linear (in)equality constraints for numerical data,
- Constraints on value combinations of categorical data
- Conditional constraints on numerical and/or mixed data

In literature these constraints, or restrictions are refered to as “edits”. `editrules` can perform common rule set manipulations like variable elimination and value substitution, and offers error localization functionality based on the (generalized) paradigm of Fellegi and Holt. Under this paradigm, one determines the smallest (weighted) number of variables to adapt such that no (additional or derived) rules are violated. The paradigm is based on the assumption that errors are distributed randomly over the variables and there is no detectable cause of error. It also decouples the detection of corrupt variables from their correction. For some types of error, such as sign flips, typing errors or rounding errors, this assumption does not hold. These errors can be detected and are closely related to their resolution. The reader is referred to the `deducorrect` package for treating such errors.

**I. Define edits**

`editrules` provides several methods for creating edits from a character, expression, data.frame or a text file.

- **editfile** Read conditional numerical, numerical and categorical constraints from textfile
- **editset** Create conditional numerical, numerical and categorical constraints
- **editmatrix** Create a linear constraint matrix for numerical data
- **editarray** Create value combination constraints for categorical data

**II. Check and find errors in data**

`editrules` provides several method for checking data.frames with edits

- **violatedEdits** Find out which record violates which edit.
- **localizeErrors** Localize erroneous fields using Fellegi and Holt’s principle.
errorLocalizer  Low-level error localization function using B&B algorithm

Note that you can call plot, summary and print on results of these functions.

IV. Manipulate and check edits

editrules provides several methods for manipulating edits

- **substValue**: Substitute a value in a set of rules
- **eliminate**: Derive implied rules by variable elimination
- **reduce**: Remove unconstraint variables
- **isFeasible**: Check for contradictions
- **duplicated**: Find duplicated rules
- **blocks**: Decompose rules into independent blocks
- **disjunct**: Decouple conditional edits into disjunct edit sets
- **separate**: Decompose rules in blocks and decouple conditional edits
- **generateEdits**: Generate all nonredundant implicit edits (editarray only)

V. Plot and coerce edits

editrules provides several methods for plotting and coercion.

- **editrules.plotting**: Plot edit-variable connectivity graph
- **as.igraph**: Coerce to edit-variable connectivity igraph object
- **as.character**: Coerce edits to character representation
- **as.data.frame**: Store character representation in data.frame

adjacency  Derive adjacency matrix from collection of edits

Description

A set of edits can be represented as a graph where every vertex is an edit. Two vertices are connected if they have at least one variable in vars in common.

Usage

```r
adjacency(E, nodetype = c("all", "rules", "vars"),
  rules = rownames(E), vars = getVars(E), ...)
```

```r
## S3 method for class 'editmatrix'
adjacency(E,
  nodetype = c("all", "rules", "vars"),
```
rules = rownames(E), vars = getVars(E), ...)

## S3 method for class 'editarray'
adjacency(E,
    nodetype = c("all", "rules", "vars"),
    rules = rownames(E), vars = getVars(E), ...)

## S3 method for class 'editset'
adjacency(E,
    nodetype = c("all", "rules", "vars"),
    rules = c(rownames(E$num), rownames(E$mixcat)),
    vars = getVars(E), ...)

## S3 method for class 'editmatrix'
as.igraph(x,
    nodetype = c("all", "rules", "vars"),
    rules = editnames(E), vars = getVars(E),
    weighted = TRUE, ...)

## S3 method for class 'editarray'
as.igraph(x,
    nodetype = c("all", "rules", "vars"),
    rules = editnames(E), vars = getVars(E),
    weighted = TRUE, ...)

## S3 method for class 'editset'
as.igraph(x,
    nodetype = c("all", "rules", "vars"),
    rules = editnames(E), vars = getVars(E),
    weighted = TRUE, ...)

Arguments

E editmatrix, editarray or editset

nodetype adjacency between rules, vars or both?

rules selection of edits

vars selection of variables

... arguments to be passed to or from other methods

x An object of class editmatrix, editarray or editset

weighted see graph.adjacency

Details

adjacency returns the adjacency matrix. The elements of the matrix count the number of variables shared by the edits indicated in the row- and column names. The adjacency matrix can be converted to an igraph object with graph.adjacency from the igraph package.

as.igraph converts a set of edits to an igraph object directly.
Value

the adjacency matrix of edits in E with resect to the variables in vars

See Also

plot.editmatrix, plot.editarray, plot.editset

Examples

```r
## Examples with linear (in)equality edits

# load predefined edits from package
data(edits)
edits

# convert to editmatrix
E <- editmatrix(edits)

## Not run:
# (Note to reader: the Not run directive only prevents the example commands from
# running when package is built)

## Total edit graph
plot(E)

## Graph with dependent edits
plot(E, nodetype="rules")

## Graph with dependent variables
plot(E, nodetype="vars")

## Total edit graph, but with curved lines (option from igraph package)
plot(E, edge.curved=TRUE)

## graph, plotting just the connections caused by variable 't'
plot(E, vars='t')

## End(Not run)

## here's an example with a broken record.
r <- c(ct = 100, ch = 30, cp = 70, p=30, t=130)
violatedEdits(E,r)
errorLocalizer(E,r)$searchBest()$adapt

## we color the violated edits and the variables that have to be adapted

## Not run
set.seed(1) # (for reproducibility)
plot(E,
    adapt=errorLocalizer(E,r)$searchBest()$adapt,
```
### Examples with categorical edits

#### Generate an editarray:

```r
E <- editarray(expression(
  age %in% c('<15','16-65','>65'),
  employment %in% c('unemployed','employed','retired'),
  salary %in% c('none','low','medium','high'),
  if (age == '<15') employment=='unemployed',
  if (salary != 'none') employment != 'unemployed',
  if (employment == 'unemployed') salary == 'none'))
```

#### Not run:

- # plot total edit graph
  ```r
  plot(E)
  ```
- # plot with a different layout
  ```r
  plot(E,layout=layout.circle)
  ```
- # plot edit graph, just the connections caused by 'salary'
  ```r
  plot(E,vars='salary')
  ```

### Not run:

- # extract edit graph
  ```r
  as.igraph(E)
  ```
- # extract edit graph, just the connections caused by 'salary'
  ```r
  as.igraph(E,vars='salary')
  ```
- # extract adjacency matrix
  ```r
  adjacency(E)
  ```
- # extract adjacency matrix, only caused by 'employment'
  ```r
  adjacency(E,vars='employment')
  ```
as.editmatrix

Coerce a matrix to an edit matrix.

Description

as.editmatrix interpretes the matrix as an editmatrix. The columns of the matrix are the variables and the rows are the edit rules (contraints).

Usage

as.editmatrix(A, b = numeric(nrow(A)),
ops = rep("==", nrow(A)), ...)

Arguments

A matrix to be transformed into an editmatrix.
b Constant, a numeric of length(nrow(x)), defaults to 0
ops Operators, character of length(nrow(x)) with the equality operators, defaults to "=="
... further attributes that will be attached to the resulting editmatrix

Details

If only argument x is given (the default), the resulting editmatrix is of the form Ax = 0. This can be influenced by using the parameters b and ops.

Value

an object of class editmatrix.

See Also

editmatrix
as.editset  

Coerce x to an editset

Description

x may be an editset, editmatrix, editarray or character vector

Usage

as.editset(x, ...)

Arguments

x  
object or vector to be coerced to an editset

...  
extra parameters that will be passed to as.character, if necessary

as.matrix.editarray  

Parse textual, categorical edit rules to an editarray

Description

An editarray is a boolean array (with some extra attributes) where each row contains an edit restriction on purely categorical data.

Usage

## S3 method for class 'editarray'
as.matrix(x, ...)

## S3 method for class 'editarray'
c(...)

editarray(editrules, sep = "::", env = parent.frame())

## S3 method for class 'editarray'
as.character(x, useIf = TRUE, 
  datamodel = TRUE, ...)

## S3 method for class 'editarray'
as.data.frame(x, ...)

## S3 method for class 'editarray'
as.expression(x, ...)

## S3 method for class 'editarray'
str(object, ...)

## S3 method for class 'editarray'
summary(object, useBlocks = TRUE, 
    ...)

Arguments

editrules character or expression vector.
sep textual separator, to be used internally for separating variable from category
names.
env environment to evaluate the rhs of `==' or `%in%' in.
x editarray object
useIf logical. Use if( <condition> ) <statement> or !<condition> | <statement> ?
datamodel logical. Include datamodel explicitly?
... further arguments passed to or from other methods
object an R object
useBlocks logical. Summarize each block?

Details

The function `editarray` converts a vector of edit(s) in character or expression from to an
editarray object. Edits may also be read from a data.frame, in which case it must have at least a
character column with the name `edit`. It is not strictly necessary, but highly recommended that
the datamodel (i.e. the possible levels for a variable) is included explicitly in the edits using an %in% 
statement, as shown in the examples below. The function `editfile` can read categorical edits from
a free-form text file.

Value

as.matrix: The boolean matrix part of the editarray.
editarray: An object of class editarray
as.data.frame: data.frame with columns 'name', 'edit' and 'description'.

See Also

editrules.plotting, violatedEdits, localizeErrors, editfile, editset, editmatrix, getVars,
blocks, eliminate, substValue, isFeasible generateEdits, contains, is.editarray, isSubset

Examples

# Here is the prototypical categorical edit: men cannot be pregnant.
E <- editarray(expression(
    gender %in% c('male', 'female'),
    pregnant %in% c('yes', 'no'),
    if( gender == 'male' ) pregnant == 'no'
# an editarray has a summary method:
summary(E)

# A yes/no variable may also be modeled as a logical:
editarray(expression(
    gender %in% c('male','female'),
    pregnant %in% c(TRUE, FALSE),
    if( gender == 'male' ) pregnant == FALSE
  )
)

# or, shorter (and using a character vector as input):
editarray(c(
  "gender %in% c('male','female')",
  "pregnant %in% c(TRUE, FALSE)",
  "if( gender == 'male' ) !pregnant"
  )
)

# the %in% statement may be used at will
editarray(expression(
    gender %in% c('male','female'),
    pregnant %in% c(TRUE, FALSE),
    positionInHousehold %in% c('marriage partner', 'child', 'other'),
    maritalStatus %in% c('unmarried','married','widowed','divorced'),
    if( gender == 'male' ) !pregnant,
    if( maritalStatus %in% c(
      'unmarried',
      'widowed',
      'divorced'
    ) !positionInHousehold %in% c('marriage partner','child')
  )
)

# Here is the prototypical categorical edit: men cannot be pregnant.
E <- editarray(expression(
    gender %in% c('male','female'),
    pregnant %in% c('yes','no'),
    if( gender == 'male' ) pregnant == 'no'
  )
)

# an editarray has a summary method:
summary(E)
# A yes/no variable may also be modeled as a logical:

```r
editarray(expression(  
  gender %in% c('male', 'female'),  
  pregnant %in% c(TRUE, FALSE),  
  if( gender == 'male' ) pregnant == FALSE
))
```

# or, shorter (and using a character vector as input):

```r
editarray(c(  
  "gender \%in\% c('male', 'female')",  
  "pregnant \%in\% c(TRUE, FALSE) ",  
  "if( gender == 'male' ) !pregnant"
))
```

# the \%in\% statement may be used at will

```r
editarray(expression(  
  gender %in% c('male', 'female'),  
  pregnant %in% c(TRUE, FALSE),  
  positionInHousehold %in% c('marriage partner', 'child', 'other'),  
  maritalStatus %in% c('unmarried', 'married', 'widowed', 'divorced'),  
  if( gender == 'male' ) !pregnant,  
  if( maritalStatus %in% c(  
    'unmarried',  
    'widowed',  
    'divorced'  
  ) !positionInHousehold %in% c('marriage partner', 'child')
))
```

---

**as.matrix.editmatrix**

Create an editmatrix

**Description**

An editmatrix is a numerical matrix and a set of comparison operators representing a linear system of (in)equations.

**Usage**

```r
## S3 method for class 'editmatrix'
as.matrix(x, ...)
```

## S3 method for class 'editmatrix'
as.matrix.editmatrix

c(...) 
editmatrix(editrules, normalize = TRUE)

## S3 method for class 'editmatrix'
as.data.frame(x, ...)

## S3 method for class 'editmatrix'
as.character(x, ...)

## S3 method for class 'editmatrix'
as.expression(x, ...)

## S3 method for class 'editmatrix'
str(object, ...)

## S3 method for class 'editmatrix'
summary(object, useBlocks = TRUE, ...)

Arguments

editrules A character or expression vecotr with (in)equalities written in R syntax. Alternatively, a data.frame with a column named edits, see details.
normalize logical specifying if all edits should be transformed (see description)
x editmatrix object
object an R object
useBlocks logical Summarize each block?
... Arguments to pass to or from other methods

Details

The function editmatrix generates an editmatrix from a character vector, an expression vector or a data.frame with at least the column edit. The function editfile reads edits from a free-form textfile, function as.editmatrix converts a matrix, a vector of constants and a vector of operators to an editmatrix.

By default, the editmatrix is normalized, meaning that all comparison operators are converted to one of <, <=, or ==. Users may specify edits using any of the operators <, <=, ==, > (see examples below). However it is highly recommended to let editmatrix parse them into normal form as all functions operating on editmatrices expect or convert it to normal form anyway.

Value

as.matrix: Augmented matrix of editmatrix. (See also getAb).
editmatrix: An object of class editmatrix
as.data.frame a 3-column data.frame with columns 'name' and 'edit'. If the input editmatrix has a description attribute a third column is returned.
Note

Since version 2.0-0, the behaviour of `as.data.frame.editmatrix` changed to be more symmetrical with `editmatrix.data.frame` and `as.data.frame.editmatrix`. Use `editrules:::toDataFrame` (unsupported) for the old behaviour.

See Also

`editrules.plotting`, `violatedEdits`, `localizeErrors`, `normalize`, `contains`, `is.editmatrix`, `getA`, `getAb`, `getb`, `getOps`, `getVars`, `eliminate`, `substValue`, `isFeasible`

Examples

```r
# Using a character vector to define constraints
E <- editmatrix(c("x+3*y==2*z", "x==z"))
print(E)

# Using a expression vector to define constraints
E <- editmatrix(expression(x+3*y==2*z, x==z))
print(E)

# an editmatrix also has a summary method:
summary(E)

# select rows from an editmatrix:
E <- editmatrix(c("x+3*y==2*z", "x >= z"))
E[getOps(E) == "="]

# Using data.frame to define constraints
E.df <- data.frame(
  name =c("A","B","C"),
  edit = c("x == y",
           "z + w == y + x",
           "z == y + 2*w"),
  description = c("these variables should be equal","","")
)
print(E.df)

E <- editmatrix(E.df)
print(E)

# Using a character vector to define constraints
E <- editmatrix(c("x+3*y==2*z", "x==z"))
print(E)

# Using a expression vector to define constraints
E <- editmatrix(expression(x+3*y==2*z, x==z))
print(E)

# an editmatrix also has a summary method:
summary(E)
```
# select rows from an editmatrix:
E <- editmatrix(c("x+3*y==2*z", "x >= z"))
E[getOps(E) == "=="]

# Using data.frame to define constraints
E.df <- data.frame(
  name =c("A","B","C"),
  edit = c("x == y",
           "z + w == y + x",
           "z == y + 2*w"),
  description = c("these variables should be equal","","")
)
print(E.df)
E <- editmatrix(E.df)
print(E)

backtracker  

**Backtracker: a flexible and generic binary search program**

**Description**

backtracker creates a binary search program that can be started by calling the $searchNext function. It walks a binary tree depth first. For all left nodes $choiceLeft is evaluated, for all right nodes $choiceRight is evaluated. A solution is found if $isSolution evaluates to TRUE. In that case $searchNext will return all variables in the search environment in a list. If $isSolution evaluates to NULL, it will continue to search deeper. If $isSolution evaluates to FALSE, it stops at the current node and goes up the next search node.

**Usage**

backtracker($isSolution, $choiceLeft, $choiceRight,  
  $list = NULL, $maxdepth = Inf, $maxduration = Inf, ...)

**Arguments**

- isSolution  
  expression that should evaluate to TRUE when a solution is found.
- choiceLeft  
  expression that will be evaluated for a left node
- choiceRight  
  expression that will be evaluated for a right node
- list  
  list with variables that will be added to the search environment
- maxdepth  
  integer maximum depth of the search tree
- maxduration  
  integer Default maximum search time for $searchNext() and $searchAll()
- ...  
  named variables that will be added to the search environment
Details

Methods:

$\texttt{searchNext(...,\;\text{VERBOSE=FALSE})}$ Search next solution, can be called repeatedly until there is no solution left. Named variables will be added to the search environment, this feature can be used to direct the search in subsequent calls to $\texttt{searchNext}$. $\text{VERBOSE=TRUE}$ will print all intermediate search steps and results. It can be used to debug the expressions in the backtracker.

$\texttt{searchAll(...,\;\text{VERBOSE=FALSE})}$ Return all solutions as a list.

$\texttt{reset()}$ Resets the backtracker to its initial state.

Value

backtracker object, see Methods for a description of the methods.

Examples

```r
bt <- backtracker( isSolution = {
  if (y == 0) return(TRUE)
  if (x == 0) return(FALSE)
}

, choiceLeft = { x <- x - 1; y <- y }
, choiceRight = { y <- y - 1; x <- x }
# starting values for x and y
, x=2
, y=1
)

bt$\texttt{searchNext(VERBOSE=TRUE)}
bt$\texttt{searchNext(VERBOSE=TRUE)}

# next search will return NULL because there is no more solution
bt$\texttt{searchNext()}

bt$\texttt{reset()}
```

blocks

Decompose a matrix or edits into independent blocks

Description

blocks returns a list of independent blocks $M_i$ such that $M = M_1 \oplus M_2 \oplus \cdots \oplus M_n$.

blockIndex returns a list of row indices in a logical matrix $D$ designating independent blocks.
Usage
blocks(M)
blockIndex(D)

Arguments
M matrix, editmatrix, editarray or editset to be decomposed into independent blocks
D matrix of type logical

Value
list of independent subobjects of M.
list of row indices in D indicating independent blocks. Empty rows (i.e. every column FALSE) are ignored.

Examples
# three separate blocks
E <- editmatrix(expression(
  x1 + x2 == x3,
  x3 + x4 == x5,
  x5 + x6 == x7,
  y1 + y2 == y3,
  z1 + z2 == z3
))
blocks(E)

# four separate blocks
E <- editmatrix(expression(
  x1 + x2 == x3,
  x3 + x4 == x5,
  x8 + x6 == x7,
  y1 + y2 == y3,
  z1 + z2 == z3
))
blocks(E)

# two categorical blocks
E <- editarray(expression(
  x %in% c('a','b','c'),
  y %in% c('d','e'),
  z %in% c('f','g'),
  u %in% c('w','t'),
  if ( x == 'a') y != 'd',
  if ( z == 'f') u != 'w'
))
blocks(E)
c.editset  Read general edits

Description
An editset combines numerical (linear), categorical and conditional restrictions in a single object. Internally, it consists of two editmatrices and an editarray.

Usage

```r
## S3 method for class 'editset'
c(...)  
editset(editrules, env = new.env())  
## S3 method for class 'editset'
as.character(x, datamodel = TRUE,  
    useIf = TRUE, dummies = FALSE, ...)

## S3 method for class 'editset'
as.data.frame(x, ...)

## S3 method for class 'editset'
summary(object, useBlocks = TRUE, ...)
```

Arguments

- `editrules` character vector, expression vector or data.frame (see details) containing edits.
- `env` environment to parse categorical edits in (normally, users need not specify this)
- `x` an `editset`
- `datamodel` include datamodel?
- `useIf` return vectorized version?
- `dummies` return datamodel for dummy variables?
- `...` arguments to be passed to or from other methods
- `object` an R object
- `useBlocks` logical: Summarize each block?

Details
The function `editset` converts a character or expression vector to an editset. Alternatively, a data.frame with a column called `edit` can be supplied. Function `editfile` reads edits from a free-form textfile.
Value

editset: An object of class editset

as.data.frame: a data.frame with columns 'name' and 'edit'.

See Also

editrules.plotting, violatedEdits, localizeErrors, getVars, disjunct, eliminate, substValue, isFeasible, contains, is.editset

Examples

# edits can be read from a vector of expressions
E <- editset(expression(
    if ( x > 0 ) y > 0,
    x + y == z,
    A %in% letters[1:2],
    B %in% letters[2:3],
    if ( A == 'a') B == 'b',
    if ( A == 'b') x >= 0,
    u + v == w,
    if ( u >= 0 ) w >= 0
))
E
summary(E)
as.data.frame(E)
getVars(E)
getVars(E,type='cat')
getVars(E,type='num')

## see also editfile
E <- editfile(system.file('script/edits/mixedits.R', package='editrules'))
E
summary(E)
as.data.frame(E)
getVars(E)
getVars(E,type='cat')
getVars(E,type='num')

# edits can be read from a vector of expressions
E <- editset(expression(
    if ( x > 0 ) y > 0,
    x + y == z,
    A %in% letters[1:2],
    B %in% letters[2:3],
    if ( A == 'a') B == 'b',
    if ( A == 'b') x >= 0,
checkDatamodel

Check data against a datamodel

Description

Categorical variables in dat which also occur in E are checked against the datamodel for those variables. Numerical variables are checked against edits in E that contain only a single variable (e.g. \( x > 0 \)). Values violating such edits as well as empty values are set to adapt.

Usage

checkDatamodel(E, dat, weight = rep(1, ncol(dat)), ...)

Arguments

- E: an object of class editset, editarray, or editmatrix
- dat: a data.frame
- weight: vector of weights for every variable of dat or an array of weight of the same dimensions as dat.
- ...: arguments to be passed to or from other methods

Value

An object of class errorLocation.
condition

See Also

errorLocation, localizeErrors.

<table>
<thead>
<tr>
<th>condition</th>
<th>Get condition matrix from an editset.</th>
</tr>
</thead>
</table>

**Description**

Get condition matrix from an editset.

**Usage**

condition(E)

**Arguments**

E an editset

**Value**

an editmatrix, holding conditions under which the editset is relevant.

**See Also**

disjunct, separate, editset

<table>
<thead>
<tr>
<th>datamodel</th>
<th>Summarize data model of an editarray in a data.frame</th>
</tr>
</thead>
</table>

**Description**

Summarize data model of an editarray in a data.frame

**Usage**

datamodel(E)

**Arguments**

E editarray

**Value**

data.frame describing the categorical variables and their levels.
See Also

checkDatamodel

Examples

E <- editarray(expression(
  age %in% c('under aged', 'adult'),
  positionInHousehold %in% c('marriage partner', 'child', 'other'),
  maritalStatus %in% c('unmarried', 'married', 'widowed', 'divorced'),
  if (maritalStatus %in% c('married', 'widowed', 'divorced')) positionInHousehold != 'child',
  if (age == 'under aged') maritalStatus == 'unmarried'
)
)
datamodel(E)

disjunct

Decouple a set of conditional edits

Description

An editset is transformed to a list of editsets which do not contain any conditional numeric/categorical edits anymore. Each editset gains an attribute condition, which holds the series of assumptions made to decouple the original edits. This attribute will be printed when not NULL. Warning: this may be slow for large, highly entangled sets of edits.

Usage

disjunct(E, type = c("list", "env"))

Arguments

E Object of class editset
type Return type: list (default) for editlist, env for editenv.

Value

An object of class editlist (editenv), which is nothing more than a list (environment) of editsets with a class attribute. Each element has an attribute 'condition' showing which conditions were assumed to derive the editset.

See Also

separate, condition, blocks
echelon

Examples

\begin{verbatim}
E <- editset(expression(
  x + y == z,
  if ( x > 0 ) y > 0,
  x >= 0,
  y >= 0,
  z >= 0,
  A %in% letters[1:4],
  B %in% letters[1:4],
  if (A %in% c('a','b')) y > 0,
  if (A == 'c') B %in% letters[1:3]
))

disjunct(E)
\end{verbatim}

---

**echelon**

*Bring an (edit) matrix to reduced row echelon form.*

**Description**

If E is a matrix, a matrix in reduced row echelon form is returned. If E is an editmatrix the equality part of E is transformed to reduced row echelon form. For an editset, the numerical part is transformed to reduced row echelon form.

**Usage**

```
echelon(E, ...)
```

```
## S3 method for class 'editmatrix'
echelon(E, ...)
```

```
## S3 method for class 'matrix'
echelon(E,
  tol = sqrt(.Machine$double.eps), ...)
```

```
## S3 method for class 'editset'
echelon(E, ...)
```

**Arguments**

\begin{verbatim}
E            a matrix or editmatrix
...
options to pass on to further methods.
tol         tolerance that will be used to determine if a coefficient equals zero.
\end{verbatim}
See Also

eliminate, substValue

editfile

Read edits edits from free-form textfile

Description

This utility function allows for free editrule definition in a file. One can extract only the numerical (type='num'), only the categorical (type='cat') or all edits (default) in which case an editset is returned. The function first parses all assignments in the file, so it is possible to compute or read a list of categories defining a datamodel for example.

Usage

editfile(file, type = c("all", "num", "cat", "mix"), ...)

Arguments

file name of text file to read in
type type of edits to extract. Currently, only 'num' (numerical), 'cat' (categorical) and 'all' are implemented.
... extra parameters that are currently ignored

Value

editset with all edits if type=all, editarray if type='cat', editmatrix if type='num', editset with conditional edits if type='mix'. If the return value is a list, the elements are named numedits and catedits.

editnames

Names of edits

Description

Retrieve edit names from editset, -array or -matrix

Usage

editnames(E)

Arguments

E editset, editarray or editmatrix
editrules.plotting  Graphical representation of edits

Description

Plots a graph, showing which variables occur in what edits. By default, squares represent edits, circles represent variables and an edge connecting a variable with an edit indicates that the edit contains the variable.

plot method for editarray

plot method for editset

Usage

```r
## S3 method for class 'editmatrix'
plot(x, nodetype = "all",
     rules = editnames(x), vars = getVars(x),
     violated = logical(nedits(x)),
     adapt = logical(length(getVars(x))), nabbreviate = 5,
     layout = layout.fruchterman.reingold,
     edgecolor = "steelblue", rulecolor = "khaki1",
     varcolor = "lightblue1", violatedcolor = "sienna1",
     adaptcolor = "sienna1", ...)

## S3 method for class 'editarray'
plot(x, nodetype = "all",
     rules = editnames(x), vars = getVars(x),
     violated = logical(nedits(x)),
     adapt = logical(length(getVars(x))), nabbreviate = 5,
     layout = layout.fruchterman.reingold,
     edgecolor = "steelblue", rulecolor = "khaki1",
     varcolor = "lightblue1", violatedcolor = "sienna1",
     adaptcolor = "sienna1", ...)

## S3 method for class 'editset'
plot(x, nodetype = "all",
     rules = editnames(x), vars = getVars(x),
     violated = logical(nedits(x)),
     adapt = logical(length(getVars(x))), nabbreviate = 5,
     layout = layout.fruchterman.reingold,
     edgecolor = "steelblue", rulecolor = "khaki1",
     varcolor = "lightblue1", violatedcolor = "sienna1",
     adaptcolor = "sienna1", ...)
```

Arguments

- `x` object of class `editmatrix`
nodetype 'rules', 'vars' or 'all'.
rules selection of edits
vars selection of variables
violated A named logical vector of length nrow(E). Ignored when nodetype='vars'
adapt A named logical vector of length(getVars(E)). Ignored when nodetype='rules'
nabbreviate integer To how many characters should variable and edit names be abbrevi-
ated?
layout an igraph layout function. See ?igraph::layout
edgecolor Color of edges and node frames
rulecolor Color of rule nodes (ignored when nodetype='vars')
varcolor Color of variable nodes (ignored when nodetype='rules')
violetedcolor Color of nodes corresponding to violated edits (ignored when nodetype='vars')
adaptdcolor Color of nodes corresponding to variables to adapt (ignored when nodetype='rules')
... further arguments to be passed to plot.

Details

Depending on the chosen nodetype, this function can plot three types of graphs based on an edit set.

- If nodetype="all" (default), the full bipartite graph is plotted. Each variable is represented
  by a square node while each edit is represented by a circular node. An edge is drawn when a
  variable occurs in an edit.

- If nodetype="vars" the variable graph is drawn. Each node represents a variable, and an
  edge is drawn between two nodes if the variables occur together in at least one edit. The edge
  width relates to the number of edits connecting two variables.

- If nodetype="rules" the rule graph is drawn. Each node represents an edit rule and an edge
  is drawn between two nodes if they share at least one variable. The edge width relates to the
  number of edits connecting the two edit rules.

The boolean vectors violated and adapt can be used to color violated edits or variables which
have to be adapted. The vectors must have named elements, so variables and edit names can be
matched.

The function works by coercing an edimatrix to an igraph object, and therefore relies on the plot-
ing capabilities of the igraph package. For more finetuning, use as.igraph and see ?igraph.plotting.

The default layout generated by the Fruchterman-Reingold algorithm. The resulting layout is one of
several optimal layouts, generated randomly (using a attraction-repulsion model between the nodes).
To reproduce layouts, use fix a randseed before calling the plot function.

References

Csardi G, Nepusz T: The igraph software package for complex network research, InterJournal,
See Also

as.igraph, adjacency, igraph.plotting

Examples

```r
## Examples with linear (in)equality edits

# load predefined edits from package
data(edits)
eds

# convert to editmatrix
E <- editmatrix(eds)

## Not run:
# (Note to reader: the Not run directive only prevents the example commands from
# running when package is built)

# Total edit graph
plot(E)

# Graph with dependent edits
plot(E, nodetype="rules")

# Graph with dependent variables
plot(E, nodetype="vars")

# Total edit graph, but with curved lines (option from igraph package)
plot(E, edge.curved=TRUE)

# graph, plotting just the connections caused by variable 't'
plot(E, vars='t')

## End(Not run)

# here's an example with a broken record.
r <- c(ct = 100, ch = 30, cp = 70, p=30,t=130)
violatedEdits(E,r)
errorLocalizer(E,r)$searchBest()$adapt

# we color the violated edits and the variables that have to be adapted

## Not run
set.seed(1) # (for reproducibility)
plot(E, adapt=errorLocalizer(E,r)$searchBest()$adapt, violated=violatedEdits(E,r))
## End(Not run)
```
# extract total graph (as igraph object)
as.igraph(E)

# extract graph with edges related to variable 't' and 'ch'
as.igraph(E, vars = c('t', 'ch'))

# extract total adjacency matrix
adjacency(E)

# extract adjacency matrix related to variables t and 'ch'
adjacency(E, vars = c('t', 'ch'))

## Examples with categorical edits

# generate an editarray:
E <- editarray(expression(
    age %in% c('<15', '16-65', '>65'),
    employment %in% c('unemployed', 'employed', 'retired'),
    salary %in% c('none', 'low', 'medium', 'high'),
    if (age == '<15') employment == 'unemployed',
    if (salary != 'none') employment != 'unemployed',
    if (employment == 'unemployed') salary == 'none'))

## Not run:
# plot total edit graph
plot(E)

# plot with a different layout
plot(E, layout = layout.circle)

# plot edit graph, just the connections caused by 'salary'
plot(E, vars = 'salary')

## End(Not run)

# extract edit graph
as.igraph(E)

# extract edit graph, just the connections caused by 'salary'
as.igraph(E, vars = 'salary')

# extract adjacency matrix
adjacency(E)

# extract adjacency matrix, only caused by 'employment'
adjacency(E, vars = 'employment')
edits

Example editrules, used in vignette

Description
Some example editrules

editType

Determine edittypes in editset based on ‘contains(E)’

Description
Determines edittypes based on the variables they contain (not on names of edits).

Usage

editType(E, m = NULL)

Arguments

E editset
m if you happen to have contains(E) handy, it needs not be recalculated.

See Also
contains

eliminate

Eliminate a variable from a set of edit rules

Description
Eliminating a variable amounts to deriving all (non-redundant) edits not containing that variable. Geometrically, it can be seen as a projection of the solution space (records obeying all edits) along the eliminated variable's axis. If the solution space is non-convex (as is the usually case when conditional edits are involved), multiple projections of convex subregions are performed.

For objects of class editmatrix, Fourier-Motzkin elimination is used to eliminate a variable from the of linear (in)equality restrictions. An observation of Kohler (1967) is used to reduce the number of implied restrictions. Obvious redundancies of the type 0 < 1 are removed as well.

For categorical edits in an editarray, the elimination method is based on repeated logical reduction on categories. See Van der Loo (2012) for a description.

For an editset, E is transformed to an editlist. Each element of an editlist describes a convex subregion of the total solution space of the editset. After this, the elimination method for editlist is called.

For an editlist, the variable is eliminated from each constituting editset.
Usage

eliminate(E, var, ...)

## S3 method for class 'editmatrix'
eliminate(E, var, ...)

## S3 method for class 'editarray'
eliminate(E, var, ...)

## S3 method for class 'editset'
eliminate(E, var, ...)

## S3 method for class 'editlist'
eliminate(E, var, ...)

Arguments

E  editmatrix or editarray
var  name of variable to be eliminated
...  arguments to be passed to or from other methods

Value

If E is an editmatrix or editarray, an object of the same class is returned. A returned editmatrix contains an extra history attribute which is used to reduce the number of generated edits in consecutive eliminations (see getH). If E is an editset, an object of class editlist is returned.

References


See Also

substValue, isObviouslyInfeasible, isObviouslyRedundant, generateEdits

Examples

# The following is an example by Williams (1986). Eliminating all variables except z maximizes -4x1 + 5x2 +3x3:
P <- editmatrix(c(
  "4*x1 - 5*x2 - 3*x3 + z <= 0",
  "-x1 + x2 -x3 <= 2",
...
eliminate

"x1 + x2 + 2*x3 <= 3",

"-x1 <= 0",

"-x2 <= 0",

"-x3 <= 0")

# eliminate 1st variable
(P1 <- eliminate(P, "x1", fancynames=TRUE))

# eliminate 2nd variable. Note that redundant rows have been eliminated
(P2 <- eliminate(P1, "x2", fancynames=TRUE))

# finally, the answer:
(P3 <- eliminate(P2, "x3", fancynames=TRUE))

# check which original edits were used in deriving the new ones
geth(P3)

# check how many variables were eliminated
geth(P3)

# An example with an equality and two inequalities
# The only thing to do is solving for x in e1 and substitute in e3.
(E <- editmatrix(c(
    "2*x + y == 1",
    "y > 0",
    "x > 0"), normalize=TRUE))

eliminate(E,"x", fancynames=TRUE)

# This example has two equalities, and it's solution
# is the origin (x,y)=(0,0)
(E <- editmatrix(c(
    "y <= 1 - x",
    "y >= -1 + x",
    "x == y",
    "y == -2*x"), normalize=TRUE))

eliminate(E,"x", fancynames=TRUE)

# this example has no solution, the equalities demand (x,y) = (0,2)
# while the inequalities demand y <= 1
(E <- editmatrix(c(
    "y <= 1 - x",
    "y >= -1 + x",
    "y == 2 - x",
    "y == -2 + x"), normalize=TRUE))

# this happens to result in an obviously unfeasable system:
isObviouslyInfeasible(eliminate(E,"x"))

# for categorical data, elimination amounts to logical derivartions. For # example
E <- editarray(expression(
    age %in% c('under aged', 'adult'),
    positionInHousehold %in% c('marriage partner', 'child', 'other'),
    maritalStatus %in% c('unmarried', 'married', 'widowed', 'divorced'),
    ...))
if (maritalStatus %in% c('married','widowed','divorced'))
  positionInHousehold != 'child',
if (maritalStatus == 'unmarried')
  positionInHousehold != 'marriage partner',
if ( age == 'under aged') maritalStatus == 'unmarried'
)
)
E

# by eliminating 'maritalStatus' we can deduce that under aged persons cannot
# be partner in marriage.
eliminate(E,"maritalStatus")

E <- editarray(expression(
  age %in% c('under aged','adult'),
  positionInHousehold %in% c('marriage partner', 'child', 'other'),
  maritalStatus %in% c('unmarried','married','widowed','divorced'),
  if (maritalStatus %in% c('married','widowed','divorced'))
    positionInHousehold != 'child',
  if (maritalStatus == 'unmarried')
    positionInHousehold != 'marriage partner',
  if ( age == 'under aged')
    maritalStatus == 'unmarried'
)
)
E

# by eliminating 'maritalStatus' we can deduce that under aged persons cannot
# be partner in marriage.
eliminate(E,"maritalStatus")

---

errorLocalizer

Create a backtracker object for error localization

**Description**

Create a backtracker object for error localization

**Usage**

errorLocalizer(E, x, ...)

## S3 method for class 'editset'
errorLocalizer(E, x, ...)

## Arguments

- **E**
  
  A named numerical vector or list (if E is an editmatrix), a named character vector or list (if E is an editarray), or a named list if E is an editlist or editset. This is the record for which errors will be localized.

- **x**
  
  A named numerical vector or list (if E is an editmatrix), a named character vector or list (if E is an editarray), or a named list if E is an editlist or editset.

- **weight**
  
  A positive weight vector. The weights are assumed to be in the same order as the variables in x.

- **maxadapt**
  
  Maximum number of variables to adapt.

- **maxweight**
  
  Maximum weight of solution, if weights are not given, this is equal to the maximum number of variables to adapt.

- **maxduration**
  
  Maximum time (in seconds), for $searchNext(), searchAll() (not for searchBest, use searchBest(maxduration=duration) in stead)

- **tol**
  
  Tolerance passed to link(isObviouslyInfeasible) (used to check for bound conditions).

## Value

An object of class `backtracker`. Each execution of $searchNext() yields a solution in the form of a list (see details). Executing $searchBest() returns the lowest-weight solution. When multiple solutions with the same weight are found, $searchBest() picks one at random.

## Details

Generate a `backtracker` object for error localization in numerical, categorical, or mixed data. This function generates the workhorse program, called by `localizeErrors` with method=`localizer`.

The returned `backtracker` can be used to run a branch-and-bound algorithm which finds the least (weighted) number of variables in x that need to be adapted so that all restrictions in E can be satisfied. (Generalized principle of Fellegi and Holt (1976)).
The B&B tree is set up so that in one branch, a variable is assumed correct and its value substituted in \( E \), while in the other branch a variable is assumed incorrect and eliminated from \( E \). See De Waal (2003), chapter 8 or De Waal, Pannekoek and Scholtus (2011) for a concise description of the B&B algorithm.

Every call to `<backtracker>$\text{searchNext}()$` returns one solution list, consisting of

- \( w \): The solution weight.
- `adapt`: logical indicating whether a variable should be adapted (TRUE) or not

Every subsequent call leads either to `NULL`, in which case either all solutions have been found, or \( \text{maxduration} \) was exceeded. The property `<backtracker>$\text{maxdurationExceeded}$` indicates if this is the case. Otherwise, a new solution with a weight \( w \) not higher than the weight of the last found solution is returned.

Alternatively `<backtracker>$\text{searchBest}()$` will return the best solution found within \( \text{maxduration} \) seconds. If multiple equivalent solutions are found, a random one is returned.

The backtracker is prepared such that missing data in the input record \( x \) is already set to adapt, and missing variables have been eliminated already.

The backtracker will crash when \( E \) is an `editarray` and one or more values are not in the data-model specified by \( E \). The more user-friendly function `localizeErrors` circumvents this. See also `checkDatamodel`.

**Numerical stability issues**

For records with a large numerical range (eg 1-1E9), the error locations represent solutions that will allow repairing the record to within roundoff errors. We highly recommend that you round near-zero values (for example, everything \( \leq \text{sqrt(.Machine$double\_eps)} \)) and scale a record with values larger than or equal to 1E9 with a constant factor.

**Note**

This method is potentially very slow for objects of class `editset` that contain many conditional restrictions. Consider using `localizeErrors` with the option `method="mip"` in such cases.

**References**


**See Also**

`errorLocalizer.mip`, `localizeErrors`, `checkDatamodel`, `violatedEdits`, `errorLocalizer`
Examples

```r
### examples with numerical edits
# example with a single editrule
# p = profit, c = cost, t = turnover
E <- editmatrix(c("p + c == t"))
cp <- errorLocalizer(E, x=c(p=755, c=125, t=200))
# x obviously violates E. With all weights equal, changing any variable will do.
# first solution:
cp$searchNext()
# second solution:
cp$searchNext()
# third solution:
cp$searchNext()
# there are no more solution since changing more variables would increase the
# weight, so the result of the next statement is NULL:
cp$searchNext()

# Increasing the reliability weight of turnover, yields 2 solutions:
cp <- errorLocalizer(E, x=c(p=755, c=125, t=200), weight=c(1,1,2))
# first solution:
cp$searchNext()
# second solution:
cp$searchNext()
# no more solutions available:
cp$searchNext()

# A case with two restrictions. The second restriction demands that
# c/t >= 0.6 (cost should be more than 60% of turnover)
E <- editmatrix(c(
  "p + c == t",
  "c - 0.6*t >= 0")
)  
cp <- errorLocalizer(E,x=c(p=755,c=125,t=200))
# Now, there's only one solution, but we need two runs to find it (the 1st one
# has higher weight)
cp$searchNext()
cp$searchNext()

# With the searchBest() function, the lowest weight solution is found at once:
errorLocalizer(E,x=c(p=755,c=125,t=200))$searchBest()

# An example with missing data.
E <- editmatrix(c(
  "p + c1 + c2 == t",
  "c1 - 0.3*t >= 0",
  "p > 0",
  "c1 > 0",
  "c2 > 0",
  "t > 0")
)  
cp <- errorLocalizer(E,x=c(p=755, c1=NA, c2=50, t=200))
# (Note that e2 is violated.)
```

There are two solutions. Both demand that \( c2 \) is adapted:

\begin{verbatim}
cp$searchNext()
cp$searchNext()
\end{verbatim}

### Examples with categorical edits

# 3 variables, recording age class, position in household, and marital status:
# We define the datamodel and the rules

\[
E \leftarrow \text{editarray(expression(}
\begin{align*}
&\text{age } \%\text{in% c('under aged', 'adult'), } \\
&\text{maritalStatus } \%\text{in% c('unmarried', 'married', 'widowed', 'divorced'), } \\
&\text{positionInHousehold } \%\text{in% c('marriage partner', 'child', 'other'), } \\
&\text{if( age }==\text{ 'under aged' }\) \\
&\text{maritalStatus }==\text{ 'unmarried', } \\
&\text{if( maritalStatus }\%\text{in% c('married', 'widowed', 'divorced'))} \\
&\text{positionInHousehold }\%\text{in% c('marriage partner', 'child')} \\
\end{align*}
\]

\[
E
\]

# Let's define a record with an obvious error:

\[
r \leftarrow \text{c(}
\begin{align*}
&\text{age }=\text{ 'under aged', } \\
&\text{maritalStatus}=\text{'married', } \\
&\text{positionInHousehold}=\text{'child'}
\end{align*}
\]

# The age class and position in household are consistent, while the marital
# status conflicts. Therefore, changing only the marital status (in stead of
# both age class and position in household) seems reasonable.

\[
el \leftarrow \text{errorLocalizer(E,r)}
\]

Localize errors using a MIP approach.

**Description**

Localize errors using a MIP approach.
Usage

```r
errorLocalizer.mip(E, x, weight = rep(1, length(x)),
                   maxduration = 600, verbose = "neutral", ...)
```

Arguments

- `E`: an `editset`, `editmatrix`, or `editarray`
- `x`: named numeric with data
- `weight`: numeric with weights
- `maxduration`: number of seconds that is spent on finding a solution
- `verbose`: verbosity argument that will be passed on to `solve lpSolveAPI`
- `...`: other arguments that will be passed on to `solve`.

Value

- list with solution weight `w`, logical `adapt` stating what to adapt, `x_feasible` and the lp problem (an `lpExtPtr` object)

Details

`errorLocalizer.mip` uses `E` and `x` to define a mixed integer problem and solves this problem using `lpSolveAPI`. This function can be much faster than `errorLocalizer` but does not return the degeneracy of a solution. However it does return a bonus: `x_feasible`, a feasible solution.

Note

If the maximum absolute value of `x` ≥ 10^9, it is pre-scaled with a factor of √(max(|x|)).

References

E. De Jonge and Van der Loo, M. (2012) Error localization as a mixed-integer program in editrules (included with the package)


See Also

- `localizeErrors`
- `errorLocalizer`
- `errorLocation`
The errorLocation object

Description

Object storing information on error locations in a dataset.

Usage

```r
## S3 method for class 'errorLocation'
plot(x,
     topn = min(10, ncol(x$adapt)), ...)

## S3 method for class 'errorLocation'
summary(object, ...)
```

Arguments

- `x` errorLocation object
- `topn` Number of variables to show in 'errors per variable plot'. Only the top-n are are shown. By default the top-20 variables with the most errors are shown.
- `...` other arguments that will be transferred to `barplot`
- `object` an R object

Details

The errorlocation objects consists of the following slots which can be accessed with the dollar operator, just like with lists. Right now the only functions creating such objects are `localizeErrors` and `checkDatamodel`.

- `adapt` a logical array where each row/column shows which record/variable should be adapted.
- `status` A data.frame with the same number of rows as `adapt`. It contains the following columns
  - `weight` weight of the found solution
  - `degeneracy` number of equivalent solutions found
  - `user` user time used to generate solution (as in `sys.time`)
  - `system` system time used to generate solution (as in `sys.time`)
  - `elapsed` elapsed time used to generate solution (as in `sys.time`)
  - `maxDurationExceeded` Was the maximum search time reached?
  - `memfail` Indicates whether a branch was broken off due to memory allocation failure (branch and bound only)
- `method` The error localization method used, can be "mip", "localizer" or "checkDatamodel".
- `call` The R calls to the function generating the object.
- `user` character user who generated the object.
• timestamp character timestamp.

It is possible to plot objects of class `errorLocation`. An overview containing three or four graphs will be plotted in a new window. Axes in scatterplots are set to logarithmic if their scales maxima exceed 50.

**See Also**

`localizeErrors`, `checkDatamodel`

**Examples**

```r
# an editmatrix and some data:
E <- editmatrix(c(
    "x + y == z",
    "x > 0",
    "y > 0",
    "z > 0"))

dat <- data.frame(
    x = c(1,-1,1),
    y = c(-1,1,1),
    z = c(2,0,2))

# localize all errors in the data
err <- localizeErrors(E,dat)

summary(err)

# what has to be adapted:
err$adapt
# weight, number of equivalent solutions, timings,
err$status

## Not run

# Demonstration of verbose processing
# construct 2-block editmatrix
F <- editmatrix(c(
    "x + y == z",
    "x > 0",
    "y > 0",
    "z > 0",
    "w > 10"))

# Using 'dat' as defined above, generate some extra records
dd <- dat
for ( i in 1:5 ) dd <- rbind(dd,dd)

dd$w <- sample(12,nrow(dd),replace=TRUE)

# localize errors verbosely
(err <- localizeErrors(F,dd,verbose=TRUE))
```
# printing is cut off, use summary for an overview
summary(err)

# or plot (not very informative in this artificial example)
plot(err)

## End (Not run)

for (d in dir("../pkg/R",full.names=TRUE)) dmp <- source(d)
# Example with different weights for each record
E <- editmatrix('x + y == z')

dat <- data.frame(
    x = c(1,1),
    y = c(1,1),
    z = c(1,1))

# At equal weights, both records have three solutions (degeneracy): adapt x, y # or z:
localizeErrors(E,dat)$status

# Set different weights per record (lower weight means lower reliability):
w <- matrix(c(
    1,2,2,
    2,2,1),nrow=2,byrow=TRUE)

localizeErrors(E,dat,weight=w)

# an example with categorical variables
E <- editarray(expression(
    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( maritalStatus %in% c('married','widowed','divorced'))
        !positionInHousehold %in% c('marriage partner','child')
)

E

dat <- data.frame(
    age = c('under aged','adult','adult'),
    maritalStatus=c('married','unmarried','widowed'),
    positionInHousehold=c('child','other','marriage partner'))

dat
localizeErrors(E,dat)
# the last record of dat has 2 degenerate solutions. Running the last command # a few times demonstrates that one of those solutions is chosen at random.

# Increasing the weight of 'positionInHousehold' for example, makes the best
# solution unique again
localizeErrors(E,dat,weight=c(1,1,2))

# an example with mixed data:

E <- editset(expression(
    x + y %%z%%,  
    2*u + 0.5*v == 3*w,  
    w %>% 0,  
    if (x > 0) y > 0,  
    x %>% 0,  
    y %>% 0,  
    z %>% 0,  
    A %in% letters[1:4],  
    B %in% letters[1:4],  
    C %in% c(TRUE,FALSE),  
    D %in% letters[5:8],  
    if (A %in% c('a','b')) y > 0,  
    if (A == 'c') B %in% letters[1:3],  
    if (!C == TRUE) D %in% c('e','f')
))

set.seed(1)

dat <- data.frame(
    x = sample(-1:8),  
    y = sample(-1:8),  
    z = sample(10),  
    u = sample(-1:8),  
    v = sample(-1:8),  
    w = sample(10),  
    A = sample(letters[1:4],10,replace=TRUE),  
    B = sample(letters[1:4],10,replace=TRUE),  
    C = sample(c(TRUE,FALSE),10,replace=TRUE),  
    D = sample(letters[5:9],10,replace=TRUE),  
    stringsAsFactors=FALSE
)

(el <- localizeErrors(E,dat,verbose=TRUE))
**Description**

Implements the Field Code Forest (FCF) algorithm of Garfinkel et al (1986) to derive all essentially new implicit edits from an editarray. The FCF is really a single, highly unbalanced tree. This algorithm traverses the tree, pruning many unnecessary branches, uses blocks to divide and conquer, and optimizes traversing order. See Van der Loo (2012) for a description of the algorithms.

**Usage**

```r
generateEdits(E)
```

**Arguments**

- `E` An editarray

**Value**

A 3-element named list, where element `E` is an editarray containing all generated edits. `nodes` contains information on the number of nodes in the tree and vs the number of nodes traversed and `duration` contains user, system and elapsed time in seconds. The `summary` method for editarray prints this information.

**References**


M.P.J. Van der Loo (2012). Variable elimination and edit generation with a flavour of semigroup algebra (submitted)

---

**getA**

*Returns the coefficient matrix A of linear (in)equalities*

**Description**

Returns the coefficient matrix $A$ of linear (in)equalities

**Usage**

```r
getA(E)
```

**Arguments**

- `E` editmatrix

**Value**

numeric matrix $A$
getAb

See Also
editmatrix

Examples

E <- editmatrix(c("x+3*y == 2*z", "x > 2")
print(E)

# get editrules, useful for storing and maintaining the rules external from your script as.data.frame(E)

# get coefficient matrix of inequalities
geta(E)

# get augmented matrix of linear edit set
getab(E)

# get constants of inequalities (i.e. c(0, 2))
getb(E)

# get operators of inequalities (i.e. c("==",">"))
getops(E)

# get variables of inequalities (i.e. c("x","y","z"))
getvars(E)

# isNormalized
isNormalized(E)

#normalized E
E <- normalize(E)
E

# is het now normalized?
isNormalized(E)

getAb

Returns augmented matrix representation of edit set.

Description
For a system of linear (in)equations of the form $Ax \odot b$, $\odot \in \{<,\leq,=\}$, the matrix $A|b$ is called the augmented matrix.

Usage

getAb(E)
Arguments

E editmatrix

Value

numeric matrix A\|b

See Also

editmatrix as.matrix.editmatrix

Examples

E <- editmatrix(c("x+3*y == 2*z",
"x > 2")
print(E)

# get editrules, useful for storing and maintaining the rules external from your script
as.data.frame(E)

# get coefficient matrix of inequalities
getA(E)

# get augmented matrix of linear edit set
getAb(E)

# get constants of inequalities (i.e. c(0, 2))
getb(E)

# get operators of inequalities (i.e. c("==",">"))
getOps(E)

# get variables of inequalities (i.e. c("x","y","z"))
getVars(E)

# isNormalized
isNormalized(E)

#normalized E
E <- normalize(E)
E

# is het now normalized?
isNormalized(E)
getb

| getb | Returns the constant part b of a linear (in)equality |

**Description**

Returns the constant part b of a linear (in)equality

**Usage**

```r
getb(E)
```

**Arguments**

- `E`: editmatrix

**Value**

numeric vector b

**See Also**

- ```r
  editmatrix
  ```

**Examples**

```r
E <- editmatrix(c("x+3*y == 2*z",
                   "x > 2")
)
p
```

# get edirules, useful for storing and maintaining the rules external from your script
# as.data.frame(E)

# get coefficient matrix of inequalities
getA(E)

# get augmented matrix of linear edit set
getAb(E)

# get constants of inequalities (i.e. c(0, 2))
getb(E)

# get operators of inequalities (i.e. c("==",">"))
getOps(E)

# get variables of inequalities (i.e. c("x","y","z"))
getVars(E)

# isNormalized
getH

Returns the derivation history of an edit matrix or array

Description

Function `eliminate` tracks the history of edits in a logical array `H`. `H` has `nrow(E)` rows and the number of columns is the number of edits in the `editmatrix` as it was first defined. If `H[i,j1], H[i,j2],...,H[i,jn]` are TRUE, then `E[i,]` is some (positive, linear) combination of original edits `E[j1,], E[j2,],...,E[jn,]

`h` records the number of variables eliminated from `E` by `eliminate`

Usage

```r
getH(E)
geth(E)
```

Arguments

- `E` `editmatrix`

Details

Attributes `H` and `h` are used to detect redundant derived edits.

See Also

`editmatrix`, `eliminate`
getOps

Returns the operator part of a linear (in)equality editmatrix $E$

**Description**

Returns the operator part of a linear (in)equality editmatrix $E$

**Usage**

```
getOps(E)
```

**Arguments**

- `E` editmatrix

**Value**

character vector with the (in)equality operators.

**See Also**

- `editmatrix`

**Examples**

```r
E <- editmatrix(c("x + 3*y == 2*z",
                    "x > 2")
)
print(E)

# get editrules, useful for storing and maintaining the rules external from your script
as.data.frame(E)

# get coefficient matrix of inequalities
geta(E)

# get augmented matrix of linear edit set
getAb(E)

# get constants of inequalities (i.e. c(0, 2))
getb(E)

# get operators of inequalities (i.e. c("==",">"))
getOps(E)

# get variables of inequalities (i.e. c("x","y","z"))
getVars(E)

# isNormalized
```
getVars

get names of variables in a set of edits

Description

get names of variables in a set of edits

Usage

getVars(E, 

## S3 method for class 'editset'
getVars(E,
    type = c("all", "num", "cat", "mix", "dummy"), ...)

## S3 method for class 'NULL'
getVars(E, ...)

Arguments

E editset, editmatrix, or editarray
... Arguments to be passed to or from other methods
type (editset- or list only) select which variables to return. all means all (except dummies), num means all numericals, cat means all categoricals, mix means those numericals appearing in a logical constraint and dummy means dummy variables connecting the logical with numerical constraints.

Value

character vector with the names of the variables.

See Also

getA, getb, getAb, getOps
**Examples**

```r
E <- editmatrix(c("x+3*y == 2*z" ,"x > 2")
getVars(E)

E <- editarray(c("gender %in% c('male','female')", "pregnant %in% c(TRUE, FALSE)", "if( gender == 'male' ) pregnant == FALSE"
)
getVars(E)
```

---

**is.editrules**  
*Check object class*

**Description**

Check object class

**Usage**

- `is.editset(x)`
- `is.editmatrix(x)`
- `is.editarray(x)`

**Arguments**

- `x` object to be checked

**Value**

logical
isFeasible: Check consistency of set of edits

Description
When variables are eliminated one by one from a set of edits, eventually either no edits are left or an obvious contradiction is encountered. In the case no records can obey all edits in the set which is therefore inFeasible.

Usage
isFeasible(E, warn = FALSE)

Arguments
E an editmatrix, editarray or editset
warn logical: should a warning be emitted when system is infeasible?

Value
TRUE or FALSE

Note
This function can potentially take a long time to complete, especially when many connected (conditional) edits are present. Consider using blocks to check feasibility of independent blocks.

See Also
isObviouslyInfeasible, isObviouslyRedundant

isNormalized: Check if an editmatrix is normalized

Description
Check if an editmatrix is normalized

Usage
isNormalized(E)

Arguments
E editmatrix
isObviouslyInfeasible

Value

TRUE when all comparison operators of E are in \{<,\leq,=\}

See Also

editmatrix

---

isObviouslyInfeasible  Check for obvious contradictions in a set of edits

Description

Obvious contradictions are edits of the form 1 < 0, or categorical edits defining that a record fails for any value combination. If this function evaluates to TRUE, the set of edits is guaranteed infeasible. If it evaluates to FALSE, this does not guarantee feasibility. See isFeasible for a complete test.

Usage

isObviouslyInfeasible(E, ...)

## S3 method for class 'editmatrix'
isObviouslyInfeasible(E,
  tol = sqrt(.Machine$double.eps), ...)

## S3 method for class 'editarray'
isObviouslyInfeasible(E, ...)

## S3 method for class 'editset'
isObviouslyInfeasible(E, ...)

## S3 method for class 'editlist'
isObviouslyInfeasible(E, ...)

## S3 method for class 'editenv'
isObviouslyInfeasible(E, ...)

Arguments

E  An editset, editmatrix, editarray, editlist or editenv

...  Arguments to be passed to or from other methods.

tol  Tolerance for checking against zero.

Value

A logical for objects of class editset, editarray or editmatrix. A logical vector in the case of an editlist or editset.
See Also

`isObviouslyRedundant, isFeasible`
`eliminate editmatrix`

---

`isObviouslyRedundant`  *Find obvious redundancies in set of edits*

**Description**

Detect simple redundancies such as duplicates or edits of the form 0 < 1 or 0 == 0. For categorical edits, simple redundancies are edits that define an empty subregion of the space of all possible records (no record can ever be contained in such a region).

**Usage**

```r
isObviouslyRedundant(E, duplicates = TRUE, ...)
```

```r
## S3 method for class 'editmatrix'
isObviouslyRedundant(E, 
duplicates = TRUE, ...)
```

```r
## S3 method for class 'editarray'
isObviouslyRedundant(E, 
duplicates = TRUE, ...)
```

```r
## S3 method for class 'editset'
isObviouslyRedundant(E, 
duplicates = rep(TRUE, 2), ...)
```

```r
## S3 method for class 'editlist'
isObviouslyRedundant(E, 
duplicates = rep(TRUE, 2), ...)
```

```r
## S3 method for class 'editenv'
isObviouslyRedundant(E, 
duplicates = rep(TRUE, 2), ...)
```

**Arguments**

- **E** An `editset, editmatrix, editarray, editlist` or `editenv`
- **duplicates** logical: check for duplicate edits? For an `editset, editlist` or `editenv` this should be a logical 2-vector indicating which of the numerical or categorical edits should be checked for duplicates.
- **...** parameters to be passed to or from other methods.
**Value**

logical vector indicating which edits are (obviously) redundant

**See Also**

`isObviouslyInfeasible, isSubset`

---

**Description**

An edit defines a subregion of the space of all possible value combinations of a record. Records in this region are interpreted as invalid. An edit rule which defines a region equal to or contained in the region defined by another edit is redundant. (In data editing literature, this is often referred to as a domination relation.)

**Usage**

```r
isSubset(E)
```

**Arguments**

- `E` editarray

**Value**

logical vector indicating if an edit is a subset of at least one other edit.

---

**Description**

Locate errors on records in a `data.frame`.

**Usage**

```r
localizeErrors(E, dat, verbose = FALSE,
               weight = rep(1, ncol(dat)), maxduration = 600,
               method = c("bb", "mip", "localizer"), useBlocks = TRUE,
               retrieve = c("best", "first"), ...)
```
Arguments

- **E**: an object of class `editset editmatrix` or `editarray`
- **dat**: a data.frame with variables in E.
- **useBlocks**: DEPRECATED. Process error localization separately for independent blocks in E (always TRUE)?
- **verbose**: print progress to screen?
- **weight**: Vector of positive weights for every variable in dat, or an array or data.frame of weights with the same dimensions as dat.
- **method**: should `errorlocalizer` ("bb") or mix integer programming ("mip") be used?
- **retrieve**: Return the first found solution or the best solution? ("bb" method only).
- **maxduration**: maximum time for `$searchBest()` to find the best solution for a single record.
- **...**: Further options to be passed to `errorLocalizer`

Details

For performance purposes, the edits are split in independent blocks which are processed separately. Also, a quick vectorized check with `checkDatamodel` is performed first to exclude variables violating their one-dimensional bounds from further calculations.

By default, all weights are set equal to one (each variable is considered equally reliable). If a vector of weights is passed, the weights are assumed to be in the same order as the columns of dat. By passing an array of weights (of same dimensions as dat) separate weights can be specified for each record.

In general, the solution to an error localization problem need not be unique, especially when no weights are defined. In such cases, `localizeErrors` chooses a solution randomly. See `errorLocalizer` for more control options.

Error localization can be performed by the Branch and Bound method of De Waal (2003) (option `method="localizer"`, the default) or by rewriting the problem as a mixed-integer programming (MIP) problem (method="mip") which is parsed to the lpsolve library. The former case uses `errorLocalizer` and is very reliable in terms of numerical stability, but may be slower in some cases (see note below). The MIP approach is much faster, but requires that upper and lower bounds are set on each numerical variable. Sensible bounds are derived automatically (see the vignette on error localization as MIP), but could cause instabilities in very rare cases.

Value

- an object of class `errorLocation`

Note

The Branch and Bound method is potentially slow for large sets of connected edits, especially when conditional edits are involved. Consider using `method="mip"` in such cases. The run-time of the B&B algorithm is related to the number of equivalent solutions, so setting different weights (reducing the number of unique solutions) may reduce computation time as well.
localizeErrors

References

E. De Jonge and Van der Loo, M. (2012) Error localization as a mixed-integer program in editrules (included with the package)

See Also

tableLocalizer

Examples

# an editmatrix and some data:
E <- editmatrix(c(
  "x + y == z",
  "x > 0",
  "y > 0",
  "z > 0"))

dat <- data.frame(
  x = c(1,-1,1),
  y = c(-1,1,1),
  z = c(2,0,2))

# localize all errors in the data
err <- localizeErrors(E,dat)

summary(err)

# what has to be adapted:
err$adapt
# weight, number of equivalent solutions, timings,
err$status

## Not run

# Demonstration of verbose processing
# construct Z-block editmatrix
F <- editmatrix(c(
  "x + y == z",
  "x > 0",
  "y > 0",
  "z > 0",
  "w > 10")
)
# Using 'dat' as defined above, generate some extra records
dd <- dat
for (i in 1:5) dd <- rbind(dd,dd)

dd$w <- sample(12,nrow(dd),replace=TRUE)
# localize errors verbosely
(err <- localizeErrors(F, dd, verbose=TRUE))

# printing is cut off, use summary for an overview
summary(err)

# or plot (not very informative in this artificial example)
plot(err)

# End(Not run)

for ( d in dir("../pkg/R", full.names=TRUE)) dmp <- source(d)
# Example with different weights for each record
E <- editmatrix("x + y == z")
dat <- data.frame(
  x = c(1,1),
  y = c(1,1),
  z = c(1,1))

# At equal weights, both records have three solutions (degeneracy): adapt x, y
# or z:
localizeErrors(E,dat)$status

# Set different weights per record (lower weight means lower reliability):
  w <- matrix(c(1,2,2,
                2,2,1),nrow=2,byrow=TRUE)
localizeErrors(E,dat,weight=w)

# an example with categorical variables
E <- editarray(expression(
  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( maritalStatus %in% c('married','widowed','divorced')
    !positionInHousehold %in% c('marriage partner','child')
    )
)
E

dat <- data.frame(
  age = c('under aged','adult','adult'),
  maritalStatus=c('married','unmarried','widowed'),
  positionInHousehold=c('child','other','marriage partner')
)
dat
localizeErrors(E,dat)
# the last record of dat has 2 degenerate solutions. Running the last command
# a few times demonstrates that one of those solutions is chosen at random.

# Increasing the weight of 'positionInHousehold' for example, makes the best
# solution unique again
localizeErrors(E,dat,weight=c(1,1,2))

# an example with mixed data:

E <- editset(expression(
  x + y == z,
  2*u + 0.5*v == 3*w,
  w >= 0,
  if ( x > 0 ) y > 0,
  x >= 0,
  y >= 0,
  z >= 0,
  A %in% letters[1:4],
  B %in% letters[1:4],
  C %in% c(TRUE,FALSE),
  D %in% letters[5:8],
  if ( A %in% c('a','b') ) y > 0,
  if ( A == 'c' ) B %in% letters[1:3],
  if ( !C == TRUE) D %in% c('e','f')
))

set.seed(1)
dat <- data.frame(
  x = sample(-1:8),
  y = sample(-1:8),
  z = sample(10),
  u = sample(-1:8),
  v = sample(-1:8),
  w = sample(10),
  A = sample(letters[1:4],10,replace=TRUE),
  B = sample(letters[1:4],10,replace=TRUE),
  C = sample(c(TRUE,FALSE),10,replace=TRUE),
  D = sample(letters[5:9],10,replace=TRUE),
  stringsAsFactors=FALSE
)

(el <- localizeErrors(E,dat,verbose=TRUE))
**Description**

Number of edits Count the number of edits in a collection of edits.

**Usage**

```r
digits(E)
```

**Arguments**

- `E` `editset, editarray` or `editmatrix`

---

**normalize**

*Normalizes an editmatrix*

**Description**

An set of linear edits of the form \( a \cdot x \odot b \) with is called normalized when all \( \odot \in \{=, \leq, <\} \)

**Usage**

```r
normalize(E)
```

**Arguments**

- `E` `editmatrix`

**Value**

If \( E \) was normalized, the original editmatrix is returned, otherwise a new normalized editmatrix will be returned

**See Also**

- `editmatrix`

**Examples**

```r
E <- editmatrix(c("x+3*y == 2*z",
                   "x > 2")
)
print(E)

# get edirules, useful for storing and maintaining the rules external from your script
as.data.frame(E)

# get coefficient matrix of inequalities
geta(E)
```
# get augmented matrix of linear edit set
getAb(E)

# get constants of inequalities (i.e. c(0, 2))
getb(E)

# get operators of inequalities (i.e. c("=",">"))
getOps(E)

# get variables of inequalities (i.e. c("x","y","z"))
getVars(E)

# isNormalized
isNormalized(E)

# normalized E
E <- normalize(E)
E

# is het now normalized?
isNormalized(E)

---

**reduce**  
*Remove redundant variables and edits.*

**Description**

Remove variables which are not contained in any edit and remove edits which are *obviously redundant.*

**Usage**

```r
reduce(E, ...)
```

```r
tol = sqrt(.Machine$double.eps), ...)
```

```r
reduce(E, ...)  
reduce(E, ...)  
reduce(E, ...)
```

**Arguments**

- `E`  
  *editmatrix or editarray*

- `...`  
  *arguments to pass to other methods*

- `tol`  
  *elements of E with absolute value < tol are considered 0.*
See Also

`contains, eliminate, substValue`

---

**Description**

The input edits are separated into disjunct blocks, and simplified to `editmatrix` or `editarray` where possible. Remaining `editsets` are separated into `disjunct editlists`.

**Usage**

```r
separate(E)
```

**Arguments**

- `E` An `editset`

**Value**

A list where each element is either an `editmatrix`, an `editarray` or an object of class `editlist` which cannot be simplified further.

**References**


**See Also**

`blocks, disjunct, condition`

**Examples**

```r
E <- editset(expression(
  x + y == z,
  2*xu + 0.5*v == 3*w,
  w >= 0,
  if ( x > 0 ) y > 0,
  x >= 0,
  y >= 0,
  z >= 0,
  A %in% letters[1:4],
  B %in% letters[1:4],
  C %in% c(TRUE, FALSE),
```

```r
```

D %in% letters[5:8],
if ( A %in% c('a', 'b') ) y > 0,
if ( A == 'c' ) B %in% letters[1:3],
if ( !C == TRUE) D %in% c('e', 'f')
)

(L <- separate(E))
sapply(L, class)

 substValue

Replace a variable by a value in a set of edits.

Description
Replace a variable by a value in a set of edits.

Usage
substValue(E, var, value, ...)

## S3 method for class 'editmatrix'
substValue(E, var, value,
        reduce = FALSE, removeredundant = TRUE, ...)

## S3 method for class 'editarray'
substValue(E, var, value,
        reduce = FALSE, ...)

## S3 method for class 'editset'
substValue(E, var, value,
        simplify = TRUE, ...)

## S3 method for class 'editlist'
substValue(E, var, value, ...)

## S3 method for class 'editenv'
substValue(E, var, value, ...)

Arguments
E editset, editmatrix, editarray, editlist or editenv
`substValue`

var character with name(s) of variable(s) to substitute

value vector with value(s) of variable(s)

... arguments to be passed to or from other methods

reduce logical should the result be simplified? For `editmatrix` this has the same effect as calling the function `reduce`. For `editarray`, the datamodel of the substituted variable is reduced to a single value, and the variable itself is not removed.

removeredundant logical. Should empty rows be removed?

simplify Simplify editset by moving logical edits containing a single numerical statement to the pure numerical part? (This is mostly for internal purposes and overwriting the default should normally not be necessary for package users).

Value

E, with variables replaced by values

Note

At the moment, objects of class `editenv` are converted to `list` prior to processing (so no performance is gained there) and reconverted afterwards.

References

Value substitution is extensively described in the package vignettes.

See Also

`eliminate`

Examples

```r
E <- editmatrix(expression(
  x + y == z,
  2*y < 10,
  3*x + 1.5*u < 7,
  z >= 0
)
)

# single value
substValue(E,’z’,10)
# multiple values
substValue(E,c(’x’,’y’),c(1,3))
# remove substituted variable from edits
substValue(E,’z’,10,reduce=TRUE)
# do not remove redundant row
substValue(E,’z’,10,removeredundant=FALSE)
```
violatedEdits

Check data against constraints

Description

Determine which record violates which edits. Returns NA when edits cannot be checked because of missing values in the data.

- For rules of the form $Ax == b \land Ax - b \leq tol$ is returned.
- For rules of the form $Ax < b$, $Ax - b < tol$ is returned.
- For rules of the form $Ax \leq b$, $Ax - b \leq tol$ is returned.

For numerical records, the default tolerance is 0. When working with doubles, the square root of machine accuracy is a reasonable alternative ($\sqrt{\cdot \text{Machine}\$\text{double.e}\text{ps}}$). The editmatrix is normalized before checks are performed.

Plot summary statistics on violatedEdits

as.data.frame violatedEdits
Usage

violatedEdits(E, dat, ...)

## S3 method for class 'character'
violatedEdits(E, dat, name = NULL, ...

## S3 method for class 'editmatrix'
violatedEdits(E, dat, tol = 0, ...

## S3 method for class 'editarray'
violatedEdits(E, dat,
  datamodel = TRUE, ...)

## S3 method for class 'editset'
violatedEdits(E, dat,
  datamodel = TRUE, ...)

## S3 method for class 'violatedEdits'
plot(x, topn = min(10, ncol(x)), ...

## S3 method for class 'violatedEdits'
summary(object, E = NULL,
  minfreq = 1, ...)

## S3 method for class 'violatedEdits'
as.data.frame(x, ...)

Arguments

E character vector with constraintsm, editset, editmatrix or editarray.
dat data.frame with data that should be checked, if a named vector is supplied it
will converted internally to a data.frame
...
further arguments that can be used by methods implementing this generic function
name name of edits
tol tolerance to check rules against.
datamodel Also check against datamodel?
x violatedEdits object.
topn Top n edits to be plotted.
object violatedEdits object
minfreq minimum freq for edit to be printed
Value

An object of class `violatedEdits`, which is a logical `nrow(dat) x nedit(E)` matrix with an extra class attribute for overloading purposes.

Note

When summarizing an object of class `violatedEdits`, every empty value is counted as one edit violation when counting violations per record.

See Also

`checkDatamodel`

Examples

```R
# Using character vector to define constraints
E <- editmatrix(c("x+3*y==2*z",
               "x=z")
)

dat <- data.frame( x = c(0,2,1),
               y = c(0,0,1),
               z = c(0,1,1)
)
print(dat)

ve <- violatedEdits(E,dat)
print(ve)
summary(ve, E)
plot(ve)

# An example with categorical data:
E <- editarray(c(
               "gender %in% c('male','female')",
               "pregnant %in% c(TRUE, FALSE)",
               "if( gender == 'male' )!pregnant"
)
)
print(E)

dat <- data.frame( gender=c('male','male','female','cylon'),
               pregnant=TRUE,FALSE,TRUE,TRUE)
)
print(dat)
# Standard, the datamodel is checked as well,
violatedEdits(E,dat)
```
# but we may turn this off
violatedEdits(E,dat,datamodel=FALSE)
Index

+Topic datasets edits, 29
+Topic data edits, 29

adjacency, 4, 27
as.character.editarray (as.matrix.editarray), 9
as.character.editmatrix (as.matrix.editmatrix), 12
as.character.editset (c.editset), 18
as.data.frame.editarray (as.matrix.editarray), 9
as.data.frame.editmatrix (as.matrix.editmatrix), 12
as.data.frame.editset (c.editset), 18
as.data.frameviolatedEdits (violatedEditset), 63
as.editmatrix, 8, 13
as.editset, 9
as.expression.editarray (as.matrix.editarray), 9
as.expression.editmatrix (as.matrix.editmatrix), 12
as.igraph, 4, 26, 27
as.igraph.editarray (adjacency), 4
as.igraph.editmatrix (adjacency), 4
as.igraph.editset (adjacency), 4
as.matrix.editarray, 9
as.matrix.editmatrix, 12, 44
backtracker, 15, 33
blockIndex (blocks), 16
blocks, 4, 10, 16, 22, 42, 50, 54, 60
c.editarray (as.matrix.editarray), 9
c.editmatrix (as.matrix.editmatrix), 12
c.editset, 18
character, 64
checkDatamodel, 20, 22, 34, 38, 39, 54, 65
choicepoint (backtracker), 15
condition, 21, 22, 60
contains, 10, 14, 19, 29, 60
data.frame, 10
datamodel, 21
disjunct, 4, 19, 21, 22, 60
duplicated, 4
echelon, 23
editarray, 3–5, 18, 20, 21, 24, 29, 30, 33, 34, 37, 42, 48, 50–54, 58–62, 64
editarray (as.matrix.editarray), 9
editenv, 51, 52, 61, 62
editfile, 3, 10, 13, 18, 24
editlist, 29, 30, 33, 51, 52, 60, 61
editlists, 60
editmatrices, 18
editmatrix, 3, 5, 8, 10, 17, 20, 21, 23–25, 29, 30, 33, 37, 42–48, 50–52, 54, 58–62, 64
editmatrix (as.matrix.editmatrix), 12
editnames, 24
editrules-package, 3
editrules.plotting, 4, 10, 14, 19, 25
edits, 29
editset, 3, 5, 10, 18, 20–24, 29, 30, 33, 34, 37, 48, 50–52, 54, 58, 60, 61, 64
editset (c.editset), 18
editsets, 60
editType, 29
eliminate, 4, 10, 14, 19, 24, 29, 46, 52, 60, 62
eliminated, 34, 50
errorLocalizer, 4, 32, 37, 54, 55
errorLocalizer.mip, 34, 36
errorLocation, 20, 21, 37, 38, 54
generateEdits, 4, 10, 30, 41
getA, 14, 42, 48
going, 13, 14, 43, 48
getb, 14, 45, 48
getH, 30, 46
geth (getH), 46
getOps, 14, 47, 48
getVars, 10, 14, 19, 48
graph.adjacency, 5

is.editarray, 10
is.editarray (is.editrules), 49
is.editmatrix, 14
is.editmatrix (is.editrules), 49
is.editrules, 49
is.editset, 19
is.editset (is.editrules), 49
isFeasible, 4, 10, 14, 19, 50, 51, 52
isNormalized, 50
isObviouslyInfeasible, 30, 50, 51, 53
isObviouslyRedundant, 30, 50, 52, 52
isSubset, 10, 53, 53

localizeErrors, 3, 10, 14, 19, 21, 33, 34, 37–39, 53

nedit, 57
normalize, 14, 58
normalized, 63

plot.editarray, 6
plot.editarray (editrules.plotting), 25
plot.editmatrix, 6
plot.editmatrix (editrules.plotting), 25
plot.editset, 6
plot.editset (editrules.plotting), 25
plot.errorLocation (errorLocation), 38
plot.violatedEdits (violatedEdits), 63

reduce, 4, 59, 62

separate, 4, 21, 22, 60
str.editarray (as.matrix.editarray), 9
str.editmatrix (as.matrix.editmatrix), 12
substValue, 4, 10, 14, 19, 24, 30, 60, 61
summary, 42
summary.editarray (as.matrix.editarray), 9
summary.editmatrix (as.matrix.editmatrix), 12
summary.editset (c.editset), 18