

# Package: **validatetools** (via r-universe)

June 14, 2024

**Title** Checking and Simplifying Validation Rule Sets

**Version** 0.5.2

**Description** Rule sets with validation rules may contain redundancies or contradictions. Functions for finding redundancies and problematic rules are provided, given a set a rules formulated with 'validate'.

**Depends** validate

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**URL** <https://github.com/data-cleaning/validatetools>

**BugReports** <https://github.com/data-cleaning/validatetools/issues>

**Imports** methods, stats, utils, lpSolveAPI

**Suggests** testthat, covr

**RoxygenNote** 7.2.3

**Repository** <https://edwindj.r-universe.dev>

**RemoteUrl** <https://github.com/data-cleaning/validatetools>

**RemoteRef** HEAD

**RemoteSha** 3cf7df5211d626c591163bc9a794d97d2f6e26d0

## Contents

detect_boundary_cat . . . . .	2
detect_boundary_num . . . . .	3
detect_fixed_variables . . . . .	4
detect_infeasible_rules . . . . .	5
detect_redundancy . . . . .	6
expect_values . . . . .	7
is_categorical . . . . .	8
is_conditional . . . . .	8

is_contradicted_by . . . . .	9
is_implied_by . . . . .	10
is_infeasible . . . . .	11
is_linear . . . . .	12
make_feasible . . . . .	12
remove_redundancy . . . . .	13
simplify_conditional . . . . .	14
simplify_fixed_variables . . . . .	15
simplify_rules . . . . .	16
substitute_values . . . . .	17
translate_mip_lp . . . . .	18
validatetools . . . . .	18

<b>Index</b>	<b>20</b>
--------------	-----------

---

detect_boundary_cat	<i>Detect viable domains for categorical variables</i>
---------------------	--

---

## Description

Detect viable domains for categorical variables

## Usage

```
detect_boundary_cat(x, ..., as_df = FALSE)
```

## Arguments

x	<a href="#">validator</a> object with rules
...	not used
as_df	return result as data.frame (before 0.4.5)

## Value

data.frame with columns \$variable, \$value, \$min, \$max. Each row is a category/value of a categorical variable.

## See Also

Other feasibility: [detect\\_boundary\\_num\(\)](#), [detect\\_infeasible\\_rules\(\)](#), [is\\_contradicted\\_by\(\)](#), [is\\_infeasible\(\)](#), [make\\_feasible\(\)](#)

## Examples

```
rules <- validator(  
  x >= 1,  
  x + y <= 10,  
  y >= 6  
)  
  
detect_boundary_num(rules)  
  
rules <- validator(  
  job %in% c("yes", "no"),  
  if (job == "no") income == 0,  
  income > 0  
)  
  
detect_boundary_cat(rules)
```

---

`detect_boundary_num` *Detect the range for numerical variables*

---

## Description

Detect for each numerical variable in a validation rule set, what its maximum and minimum values are. This allows for manual rule set checking: does rule set x overly constrain numerical values?

## Usage

```
detect_boundary_num(x, eps = 1e-08, ...)
```

## Arguments

<code>x</code>	<code>validator</code> object, rule set to be checked
<code>eps</code>	detected fixed values will have this precision.
<code>...</code>	currently not used

## Details

This procedure only finds minimum and maximum values, but misses gaps.

## Value

`data.frame` with columns "variable", "lowerbound", "upperbound".

## References

Statistical Data Cleaning with R (2017), Chapter 8, M. van der Loo, E. de Jonge  
Simplifying constraints in data editing (2015). Technical Report 2015|18, Statistics Netherlands, J. Daalmans

**See Also**

[detect\\_fixed\\_variables](#)

Other feasibility: [detect\\_boundary\\_cat\(\)](#), [detect\\_infeasible\\_rules\(\)](#), [is\\_contradicted\\_by\(\)](#), [is\\_infeasible\(\)](#), [make\\_feasible\(\)](#)

**Examples**

```
rules <- validator(  
  x >= 1,  
  x + y <= 10,  
  y >= 6  
)
```

```
detect_boundary_num(rules)
```

```
rules <- validator(  
  job %in% c("yes", "no"),  
  if (job == "no") income == 0,  
  income > 0  
)
```

```
detect_boundary_cat(rules)
```

---

`detect_fixed_variables`

*Detect fixed variables*

---

**Description**

Detects variables that have a fixed value in the rule set. To simplify a rule set, these variables can be substituted with their value.

**Usage**

```
detect_fixed_variables(x, eps = x$options("lin.eq.eps"), ...)
```

**Arguments**

x	<a href="#">validator</a> object with the validation rules.
eps	detected fixed values will have this precision.
...	not used.

**See Also**

[simplify\\_fixed\\_variables](#)

Other redundancy: [detect\\_redundancy\(\)](#), [is\\_implied\\_by\(\)](#), [remove\\_redundancy\(\)](#), [simplify\\_fixed\\_variables\(\)](#), [simplify\\_rules\(\)](#)

## Examples

```
library(validate)
rules <- validator( x >= 0
                  , x <= 0
                  )
detect_fixed_variables(rules)
simplify_fixed_variables(rules)

rules <- validator( x1 + x2 + x3 == 0
                  , x1 + x2 >= 0
                  , x3 >= 0
                  )
simplify_fixed_variables(rules)
```

---

detect\_infeasible\_rules

*Detect which rules cause infeasibility*

---

## Description

Detect which rules cause infeasibility. This methods tries to remove the minimum number of rules to make the system mathematically feasible. Note that this may not result in your desired system, because some rules may be more important to you than others. This can be mitigated by supplying weights for the rules. Default weight is 1.

## Usage

```
detect_infeasible_rules(x, weight = numeric(), ...)
```

## Arguments

x	<a href="#">validator</a> object with rules
weight	optional named <a href="#">numeric</a> with weights. Unnamed variables in the weight are given the default weight 1.
...	not used

## Value

character with the names of the rules that are causing infeasibility.

## See Also

Other feasibility: [detect\\_boundary\\_cat\(\)](#), [detect\\_boundary\\_num\(\)](#), [is\\_contradicted\\_by\(\)](#), [is\\_infeasible\(\)](#), [make\\_feasible\(\)](#)

### Examples

```
rules <- validator( x > 0)

is_infeasible(rules)

rules <- validator( rule1 = x > 0
                    , rule2 = x < 0
                    )

is_infeasible(rules)

detect_infeasible_rules(rules)
make_feasible(rules)

# find out the conflict with this rule
is_contradicted_by(rules, "rule1")
```

---

detect\_redundancy      *Detect redundant rules without removing.*

---

### Description

Detect redundancies in a rule set.

### Usage

```
detect_redundancy(x, ...)
```

### Arguments

x                    [validator](#) object with the validation rules.  
...                   not used.

### Note

For removal of duplicate rules, [simplify](#)

### See Also

Other redundancy: [detect\\_fixed\\_variables\(\)](#), [is\\_implied\\_by\(\)](#), [remove\\_redundancy\(\)](#), [simplify\\_fixed\\_variable](#)  
[simplify\\_rules\(\)](#)

**Examples**

```
rules <- validator( rule1 = x > 1
                    , rule2 = x > 2
                    )

# rule1 is superfluous
remove_redundancy(rules)

# rule 1 is implied by rule 2
is_implied_by(rules, "rule1")

rules <- validator( rule1 = x > 2
                    , rule2 = x > 2
                    )

# standout: rule1 and rule2, oldest rules wins
remove_redundancy(rules)

# Note that detection signifies both rules!
detect_redundancy(rules)
```

---

expect_values	<i>expect values</i>
---------------	----------------------

---

**Description**

expect values

**Usage**

```
expect_values(values, weights, ...)
```

**Arguments**

values	named list of values.
weights	named numeric of equal length as values.
...	not used

---

is_categorical	<i>Check if rules are categorical</i>
----------------	---------------------------------------

---

**Description**

Check if rules are categorical

**Usage**

```
is_categorical(x, ...)
```

**Arguments**

x	validator object
...	not used

**Value**

logical indicating which rules are purely categorical/logical

**Examples**

```
v <- validator( A %in% c("a1", "a2")
               , B %in% c("b1", "b2")
               , if (A == "a1") B == "b1"
               , y > x
               )

is_categorical(v)
```

---

is_conditional	<i>Check if rules are conditional rules</i>
----------------	---

---

**Description**

Check if rules are conditional rules

**Usage**

```
is_conditional(rules, ...)
```

**Arguments**

rules	validator object containing validation rules
...	not used



**Value**

logical indicating which rules are conditional

**Examples**

```
v <- validator( A %in% c("a1", "a2")
               , B %in% c("b1", "b2")
               , if (A == "a1") x > 1 # conditional
               , if (y > 0) x >= 0 # conditional
               , if (A == "a1") B == "b1" # categorical
               )

is_conditional(v)
```

---

is\_contradicted\_by      *Find out which rules are conflicting*

---

**Description**

Find out for a contradicting rule which rules are conflicting. This helps in determining and assessing conflicts in rule sets. Which of the rules should stay and which should go?

**Usage**

```
is_contradicted_by(x, rule_name)
```

**Arguments**

x                      [validator](#) object with rules.  
rule\_name              character with the names of the rules that are causing infeasibility.

**Value**

character with conflicting rules.

**See Also**

Other feasibility: [detect\\_boundary\\_cat\(\)](#), [detect\\_boundary\\_num\(\)](#), [detect\\_infeasible\\_rules\(\)](#), [is\\_infeasible\(\)](#), [make\\_feasible\(\)](#)

**Examples**

```
rules <- validator( x > 0)

is_infeasible(rules)

rules <- validator( rule1 = x > 0
                   , rule2 = x < 0
```

```

    )

is_infeasible(rules)

detect_infeasible_rules(rules)
make_feasible(rules)

# find out the conflict with this rule
is_contradicted_by(rules, "rule1")

```

---

is\_implied\_by                      *Find which rule(s) make rule\_name redundant*

---

### Description

Find out which rules are causing rule\_name(s) to be redundant.

### Usage

```
is_implied_by(x, rule_name, ...)
```

### Arguments

x	<a href="#">validator</a> object with rule
rule_name	character with the names of the rules to be checked
...	not used

### Value

character with the names of the rule that cause the implication.

### See Also

Other redundancy: [detect\\_fixed\\_variables\(\)](#), [detect\\_redundancy\(\)](#), [remove\\_redundancy\(\)](#), [simplify\\_fixed\\_variables\(\)](#), [simplify\\_rules\(\)](#)

### Examples

```

rules <- validator( rule1 = x > 1
                   , rule2 = x > 2
                   )

# rule1 is superfluous
remove_redundancy(rules)

# rule 1 is implied by rule 2
is_implied_by(rules, "rule1")

rules <- validator( rule1 = x > 2

```

```
    , rule2 = x > 2
  )

# standout: rule1 and rule2, oldest rules wins
remove_redundancy(rules)

# Note that detection signifies both rules!
detect_redundancy(rules)
```

---

is_infeasible	<i>Check the feasibility of a rule set</i>
---------------	--

---

### Description

An infeasible rule set cannot be satisfied by any data because of internal contradictions. This function checks whether the record-wise linear, categorical and conditional rules in a rule set are consistent.

### Usage

```
is_infeasible(x, ...)
```

### Arguments

x	validator object with validation rules.
...	not used

### Value

TRUE or FALSE

### See Also

Other feasibility: [detect\\_boundary\\_cat\(\)](#), [detect\\_boundary\\_num\(\)](#), [detect\\_infeasible\\_rules\(\)](#), [is\\_contradicted\\_by\(\)](#), [make\\_feasible\(\)](#)

### Examples

```
rules <- validator( x > 0)

is_infeasible(rules)

rules <- validator( rule1 = x > 0
                    , rule2 = x < 0
                    )

is_infeasible(rules)
```

```

detect_infeasible_rules(rules)
make_feasible(rules)

# find out the conflict with this rule
is_contradicted_by(rules, "rule1")

```

---

`is_linear`                      *Check which rules are linear rules.*

---

### Description

Check which rules are linear rules.

### Usage

```
is_linear(x, ...)
```

### Arguments

`x`                      [validator](#) object containing data validation rules  
`...`                      not used

### Value

logical indicating which rules are (purely) linear.

---

`make_feasible`                      *Make an infeasible system feasible.*

---

### Description

Make an infeasible system feasible, by removing the minimum (weighted) number of rules, such that the remaining rules are not conflicting. This function uses [detect\\_infeasible\\_rules](#) for determining the rules to be removed.

### Usage

```
make_feasible(x, ...)
```

### Arguments

`x`                      [validator](#) object with the validation rules.  
`...`                      passed to [detect\\_infeasible\\_rules](#)

**Value**

`validator` object with feasible rules.

**See Also**

Other feasibility: `detect_boundary_cat()`, `detect_boundary_num()`, `detect_infeasible_rules()`, `is_contradicted_by()`, `is_infeasible()`

**Examples**

```
rules <- validator( x > 0)

is_infeasible(rules)

rules <- validator( rule1 = x > 0
                    , rule2 = x < 0
                    )

is_infeasible(rules)

detect_infeasible_rules(rules)
make_feasible(rules)

# find out the conflict with this rule
is_contradicted_by(rules, "rule1")
```

---

remove_redundancy	<i>Remove redundant rules</i>
-------------------	-------------------------------

---

**Description**

Simplify a rule set by removing redundant rules

**Usage**

```
remove_redundancy(x, ...)
```

**Arguments**

<code>x</code>	<code>validator</code> object with validation rules.
<code>...</code>	not used

**Value**

simplified `validator` object, in which redundant rules are removed.

**See Also**

Other redundancy: [detect\\_fixed\\_variables\(\)](#), [detect\\_redundancy\(\)](#), [is\\_implied\\_by\(\)](#), [simplify\\_fixed\\_variable](#)  
[simplify\\_rules\(\)](#)

**Examples**

```
rules <- validator( rule1 = x > 1
                   , rule2 = x > 2
                   )

# rule1 is superfluous
remove_redundancy(rules)

# rule 1 is implied by rule 2
is_implied_by(rules, "rule1")

rules <- validator( rule1 = x > 2
                   , rule2 = x > 2
                   )

# stdout: rule1 and rule2, oldest rules wins
remove_redundancy(rules)

# Note that detection signifies both rules!
detect_redundancy(rules)
```

---

simplify\_conditional *Simplify conditional statements*

---

**Description**

Conditional rules may be constrained by the others rules in a validation rule set. This procedure tries to simplify conditional statements.

**Usage**

```
simplify_conditional(x, ...)
```

**Arguments**

x [validator](#) object with the validation rules.  
... not used.

**Value**

[validator](#) simplified rule set.

## References

TODO non-constraining, non-relaxing

## Examples

```
library(validate)

# non-relaxing clause
rules <- validator( r1 = if (x > 1) y > 3
                   , r2 = y < 2
                   )
# y > 3 is always FALSE so r1 can be simplified
simplify_conditional(rules)

# non-constraining clause
rules <- validator( r1 = if (x > 0) y > 0
                   , r2 = if (x < 1) y > 1
                   )
simplify_conditional(rules)

rules <- validator( r1 = if (A == "a1") x > 0
                   , r2 = if (A == "a2") x > 1
                   , r3 = A == "a1"
                   )
simplify_conditional(rules)
```

---

simplify\_fixed\_variables

*Simplify fixed variables*

---

## Description

Detect variables of which the values are restricted to a single value by the rule set. Simplify the rule set by replacing fixed variables with these values.

## Usage

```
simplify_fixed_variables(x, eps = 1e-08, ...)
```

## Arguments

x	<a href="#">validator</a> object with validation rules
eps	detected fixed values will have this precision.
...	passed to <a href="#">substitute_values</a> .

## Value

[validator](#) object in which

**See Also**

Other redundancy: [detect\\_fixed\\_variables\(\)](#), [detect\\_redundancy\(\)](#), [is\\_implied\\_by\(\)](#), [remove\\_redundancy\(\)](#), [simplify\\_rules\(\)](#)

**Examples**

```
library(validate)
rules <- validator( x >= 0
                   , x <= 0
                   )
detect_fixed_variables(rules)
simplify_fixed_variables(rules)

rules <- validator( x1 + x2 + x3 == 0
                   , x1 + x2 >= 0
                   , x3 >= 0
                   )
simplify_fixed_variables(rules)
```

---

simplify\_rules

*Simplify a rule set*

---

**Description**

Simplifies a rule set set by applying different simplification methods. This is a convenience function that works in common cases. The following simplification methods are executed:

- [substitute\\_values](#): filling in any parameters that are supplied via `.values` or `...`
- [simplify\\_fixed\\_variables](#): find out if there are fixed values. If this is the case, they are substituted.
- [simplify\\_conditional](#): Simplify conditional statements, by removing clauses that are superfluous.
- [remove\\_redundancy](#): remove redundant rules.

For more control, these methods can be called separately.

**Usage**

```
simplify_rules(.x, .values = list(...), ...)
```

**Arguments**

`.x` [validator](#) object with the rules to be simplified.  
`.values` optional named list with values that will be substituted.  
`...` parameters that will be used to substitute values.



**See Also**

Other redundancy: [detect\\_fixed\\_variables\(\)](#), [detect\\_redundancy\(\)](#), [is\\_implied\\_by\(\)](#), [remove\\_redundancy\(\)](#), [simplify\\_fixed\\_variables\(\)](#)

**Examples**

```
rules <- validator( x > 0
  , if (x > 0) y == 1
  , A %in% c("a1", "a2")
  , if (A == "a1") y > 1
)

simplify_rules(rules)
```

---

substitute_values	<i>substitute a value in a rule set</i>
-------------------	---

---

**Description**

Substitute values into expression, thereby simplifying the rule set. Rules that evaluate to TRUE because of the substitution are removed.

**Usage**

```
substitute_values(.x, .values = list(...), ..., .add_constraints = TRUE)
```

**Arguments**

<code>.x</code>	validator object with rules
<code>.values</code>	(optional) named list with values for variables to substitute
<code>...</code>	alternative way of supplying values for variables (see examples).
<code>.add_constraints</code>	logical, should values be added as constraints to the resulting validator object?

**Examples**

```
library(validate)
rules <- validator( rule1 = z > 1
  , rule2 = y > z
)
# rule1 is dropped, since it always is true
substitute_values(rules, list(z=2))

# you can also supply the values as separate parameters
substitute_values(rules, z = 2)

# you can choose to not add substituted values as a constraint
```

```

substitute_values(rules, z = 2, .add_constraints = FALSE)

rules <- validator( rule1 = if (gender == "male") age >= 18 )
substitute_values(rules, gender="male")
substitute_values(rules, gender="female")

```

---

translate_mip_lp	<i>translate linear rules into an lp problem</i>
------------------	--

---

### Description

translate linear rules into an lp problem

### Usage

```
translate_mip_lp(rules, objective = NULL, eps = 0.001)
```

### Arguments

rules	mip rules
objective	function
eps	accuracy for equality/inequality

---

validatetools	<i>Tools for validation rules</i>
---------------	-----------------------------------

---

### Description

validatetools is a utility package for managing validation rule sets that are defined with [validate](#). In production systems validation rule sets tend to grow organically and accumulate redundant or (partially) contradictory rules. 'validatetools' helps to identify problems with large rule sets and includes simplification methods for resolving issues.

### Problem detection

The following methods allow for problem detection:

- [is\\_infeasible](#) checks a rule set for feasibility. An infeasible system must be corrected to be useful.
- [detect\\_boundary\\_num](#) shows for each numerical variable the allowed range of values.
- [detect\\_boundary\\_cat](#) shows for each categorical variable the allowed range of values.
- [detect\\_fixed\\_variables](#) shows variables whose value is fixated by the rule set.
- [detect\\_redundancy](#) shows which rules are already implied by other rules.

**Simplifying rule set**

The following methods detect possible simplifications and apply them to a rule set.

- `substitute_values`: replace variables with constants.
- `simplify_fixed_variables`: substitute the fixed variables with their values in a rule set.
- `simplify_conditional`: remove redundant (parts of) conditional rules.
- `remove_redundancy`: remove redundant rules.

**References**

Statistical Data Cleaning with Applications in R, Mark van der Loo and Edwin de Jonge, ISBN: 978-1-118-89715-7

# Index

- \* **feasibility**
  - detect\_boundary\_cat, 2
  - detect\_boundary\_num, 3
  - detect\_infeasible\_rules, 5
  - is\_contradicted\_by, 9
  - is\_infeasible, 11
  - make\_feasible, 12
- \* **redundancy**
  - detect\_fixed\_variables, 4
  - detect\_redundancy, 6
  - is\_implied\_by, 10
  - remove\_redundancy, 13
  - simplify\_fixed\_variables, 15
  - simplify\_rules, 16
- data.frame, 3
- detect\_boundary\_cat, 2, 4, 5, 9, 11, 13, 18
- detect\_boundary\_num, 2, 3, 5, 9, 11, 13, 18
- detect\_fixed\_variables, 4, 4, 6, 10, 14, 16–18
- detect\_infeasible\_rules, 2, 4, 5, 9, 11–13
- detect\_redundancy, 4, 6, 10, 14, 16–18
- expect\_values, 7
- is\_categorical, 8
- is\_conditional, 8
- is\_contradicted\_by, 2, 4, 5, 9, 11, 13
- is\_implied\_by, 4, 6, 10, 14, 16, 17
- is\_infeasible, 2, 4, 5, 9, 11, 13, 18
- is\_linear, 12
- make\_feasible, 2, 4, 5, 9, 11, 12
- numeric, 5
- remove\_redundancy, 4, 6, 10, 13, 16, 17, 19
- simplify\_conditional, 14, 16, 19
- simplify\_fixed\_variables, 4, 6, 10, 14, 15, 16, 17, 19
- simplify\_rules, 4, 6, 10, 14, 16, 16
- substitute\_values, 15, 16, 17, 19
- translate\_mip\_lp, 18
- validate, 18
- validatetools, 18
- validator, 2–6, 9, 10, 12–16