Package 'processcheckR'

May 9, 2026

Type Package

Title Rule-Based Conformance Checking of Business Process Event Data

Version 0.1.5

Description Check compliance of event-data from (business) processes with respect to specified rules. Rules supported are of three types: frequency (activities that should (not) happen x number of times), order (succession between activities) and exclusiveness (and and exclusive choice between activities).

License MIT + file LICENSE

Encoding UTF-8

Depends R(>= 3.5.0)

Imports dplyr, bupaR (>= 0.5.1), rlang, edeaR (>= 0.9.0), stringr, stringi, glue, lifecycle, tidyr

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, eventdataR, covr, compare, testthat (>= 3.0.0)

VignetteBuilder knitr

URL https://bupar.net/, https://github.com/bupaverse/processcheckr, https://bupaverse.github.io/processcheckR/

BugReports https://github.com/bupaverse/processcheckr/issues

Config/testthat/edition 3

NeedsCompilation no

Author Gert Janssenswillen [aut, cre], Gerard van Hulzen [ctb]

Maintainer Gert Janssenswillen <gert.janssenswillen@uhasselt.be>

Repository CRAN

Date/Publication 2025-09-08 13:40:02 UTC

Contents

`absent` *Absent*

Description

Check if the specified activity is absent from a case.

The `absent` rule can be used to check whether an activity is absent in a case or not. The `n` parameter can be configured to create a different level of *absence*. When `n = 0`, an activity is not allowed to occur even a single time. The maximum number of times it is allowed to occur is `n`.

Usage

```
absent(activity, n = 0)
```

Arguments

`activity`	`character`: The activity to check. This should be an activity of the log supplied to `check_rule`.
`n`	`numeric` (default `0`): The allowed number of occurences of the activity, e.g. `n = 0` means the activity should be absent, `n = 1` means it is allowed to occur once.

See Also

Other Cardinality rules: `contains()`, `contains_between()`, `contains_exactly()`

Examples

```
library(bupaR)
library(eventdataR)

# Check for which patients the activity "MRI SCAN" is absent.
patients %>%
 check_rule(absent("MRI SCAN"))

# Check for which patients the activity "Blood test" occurs maximum a single time,
# but not 2 times or more.
patients %>%
 check_rule(absent("Blood test", n = 1))
```

---

and *AND*

Description

Check for co-existence of two activities.

Theand rule checks whether two activities both occur in a case (or are both absent). If `activity_a` exists, `activity_b` should also exist, and vice versa.

Usage

```
and(activity_a, activity_b)
```

Arguments

`activity_a`	`character`: Activity A. This should be an activity of the log supplied to `check_rule`.
`activity_b`	`character`: Activity B. This should be an activity of the log supplied to `check_rule`.

See Also

Other Exclusiveness rules: `xor()`

Examples

```
library(bupaR)
library(eventdataR)

# Check that if a patients is registered, he's also checked-out, and vice versa.
patients %>%
 check_rule(and("Registration","Check-out"))
```

check_rule *Check Declarative Rule(s)*

Description

This function can be used to check rules or constraint templates on event data. It needs a log (object of class log or derivatives, e.g. grouped_log, eventlog, activitylog, etc.). and (a) rule(s). Rules can be made with the following templates:

- *Cardinality*:
  - absent: Check if the specified activity is absent from a case,
  - contains: Check if the specified activity is present (contained) in a case,
  - contains_between: Check if the specified activity is present (contained) in a case between the minimum and maximum number of times,
  - contains_exactly: Check if the specified activity is present (contained) in a case for exactly n times.
- *Relation*:
  - ends: Check if cases end with the specified activity,
  - starts: Check if cases start with the specified activity.
  - precedence: Check for precedence between two activities,
  - response: Check for response between two activities,
  - responded_existence: Check for responded existence between two activities,
  - succession: Check for succession between two activities.
- *Exclusiveness*:
  - and: Check for co-existence of two activities,
  - xor: Check for exclusiveness of two activities.

Usage

```
check_rule(log, rule, label = NULL, eventlog = deprecated())

## S3 method for class 'log'
check_rule(log, rule, label = NULL, eventlog = deprecated())

check_rules(log, ..., eventlog = deprecated())

## S3 method for class 'log'
check_rules(log, ..., eventlog = deprecated())
```

Arguments

log	log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).
rule	A rule created by a rule function.

label	character (default NULL): Optionally, the column name under which the result of the rule should be stored.
eventlog	**[Deprecated]**; please use log instead.
...	Name-rule pairs created by rule functions.

Details

The rules or constraint templates in this package are (partially) based on *DecSerFlow* (*Declarative Service Flow Language*). For more information, see the **References** below.

Grouped Logs:

When applied to a grouped_log, the grouping variables are ignored but retained in the returned log.

Value

An annotated log (of same type as input), where – for every rule – a new column indicates whether the rule holds or not. The name of the new column can optionally be set using the label argument, or by the name of each rule in the name-rule pairs.

Methods (by class)

- check_rule(log): Check rule on a log.

Functions

- check_rules(log): Check rules on a log.

References

van der Aalst, W. M. P., & Pesic, M. (2006). DecSerFlow: Towards a Truly Declarative Service Flow Language. In M. Bravetti, M. Núñez, & G. Zavattaro (Eds.), Proceedings of the 3rd International Workshop on Web Services and Formal Methods (Vol. 4184, pp. 1–23). Springer. doi:10.1007/11841197_1

See Also

filter_rules

Examples

```
library(bupaR)
library(eventdataR)

# Check whether MRI Scan is preceded by Blood test.
patients %>%
 check_rule(precedence("Blood test","MRI SCAN"))

# Check whether MRI Scan is preceded by Blood test, and the case starts with Registration.
patients %>%
```

```
check_rules(rule1 = precedence("Blood test","MRI SCAN"),
            rule2 = starts("Registration"))
```

`contains` *Contains*

Description

Check if the specified activity is present (contained) in a case.

The `contains` rule examines whether the supplied `activity` is present in a case or not. The argument `n` can be used to set a minimum number of occurences that should be present in each case.

Usage

```
contains(activity, n = 1)
```

Arguments

`activity`	`character`: The activity to check. This should be an activity of the log supplied to `check_rule`.
`n`	`numeric` (default `1`): The minimum number of times the activity should be present. Should be greater than or equal to `1`. Use `absent` instead to check for absent (i.e. `n = 0`) activities.

See Also

Other Cardinality rules: `absent()`, `contains_between()`, `contains_exactly()`

Examples

```
library(bupaR)
library(eventdataR)

# Each patient should be registered at least once.
patients %>%
 check_rule(contains("Registration"))

# Check whether some patients have received 2 or more blood tests.
patients %>%
 check_rule(contains("Blood test", n = 2))
```

contains_between *Contains Between*

Description

Check if the specified activity is present (contained) in a case between the minimum and maximum number of times.

The `contains_between` rule examines whether the supplied `activity` is present in a case for a certain interval of times. The arguments `min` and `max` can be used to specify the allowed interval of occurences.

Usage

```
contains_between(activity, min = 1, max = 1)
```

Arguments

`activity`	`character`: The activity to check. This should be an activity of the log supplied to `check_rule`.
`min`	`numeric` (default `1`): The minimum number of times the activity should be present (inclusive). Should be greater than or equal to `0`.
`max`	`numeric` (default `1`): The maximum number of times the activity should be present (inclusive). Should be greater than or equal to `min`.

See Also

Other Cardinality rules: `absent()`, `contains()`, `contains_exactly()`

Examples

```
library(bupaR)
library(eventdataR)

# A patients should have between 0 and 4 blood tests (including 0 and 4).
patients %>%
 check_rule(contains_between("Blood test", min = 0, max = 4))
```

contains_exactly *Contains Exactly*

Description

Check if the specified activity is present (contained) in a case for exactly n times.

The contains_exactly rule examines whether the supplied activity is present in a case for an exact number of n times.

Usage

```
contains_exactly(activity, n = 1)
```

Arguments

activity	character: The activity to check. This should be an activity of the log supplied to check_rule.
n	numeric (default 1): The exact number of times the activity should be present. Should be greater than or equal to 1. Use absent instead to check for absent (i.e. n = 0) activities.

See Also

Other Cardinality rules: absent(), contains(), contains_between()

Examples

```
library(bupaR)
library(eventdataR)

# Each patient should have exactly one registration activity instance.
patients %>%
 check_rule(contains_exactly("Registration", n = 1))
```

ends *Ends*

Description

Check if cases end with the specified activity.

Usage

```
ends(activity)
```

Arguments

activity	character: The end activity. This should be an activity of the log supplied to check_rule.

See Also

Other Ordering rules: precedence(), responded_existence(), response(), starts(), succession()

Examples

```
library(bupaR)
library(eventdataR)

# A patient's last activity should be the Check-out
patients %>%
 check_rule(ends("Check-out"))
```

---

filter_rules *Filter Using Declarative Rules*

Description

This function can be used to filter event data using declaritive rules or constraint templates. It needs a log (object of class log or derivatives, e.g. grouped_log, eventlog, activitylog, etc.). and a set of rules. Rules can be made with the following templates:

- *Cardinality*:
  - absent: Check if the specified activity is absent from a case,
  - contains: Check if the specified activity is present (contained) in a case,
  - contains_between: Check if the specified activity is present (contained) in a case between the minimum and maximum number of times,
  - contains_exactly: Check if the specified activity is present (contained) in a case for exactly n times.
- *Relation*:
  - ends: Check if cases end with the specified activity,
  - starts: Check if cases start with the specified activity.
  - precedence: Check for precedence between two activities,
  - response: Check for response between two activities,
  - responded_existence: Check for responded existence between two activities,
  - succession: Check for succession between two activities.
- *Exclusiveness*:
  - and: Check for co-existence of two activities,
  - xor: Check for exclusiveness of two activities.

Usage

```
filter_rules(log, ..., eventlog = deprecated())

## S3 method for class 'log'
filter_rules(log, ..., eventlog = deprecated())
```

Arguments

log	log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).
...	Name-rule pairs created by rule functions.
eventlog	**[Deprecated]**; please use log instead.

Details

The rules or constraint templates in this package are (partially) based on *DecSerFlow* (*Declarative Service Flow Language*). For more information, see the **References** below.

Grouped Logs:

When applied to a grouped_log, the grouping variables are ignored but retained in the returned log.

Value

A filtered log (of same type as input) that satisfied the specified rules.

Methods (by class)

- filter_rules(log): Filter a log using declaritive rules.

References

van der Aalst, W. M. P., & Pesic, M. (2006). DecSerFlow: Towards a Truly Declarative Service Flow Language. In M. Bravetti, M. Núñez, & G. Zavattaro (Eds.), Proceedings of the 3rd International Workshop on Web Services and Formal Methods (Vol. 4184, pp. 1–23). Springer. doi:10.1007/11841197_1

See Also

check_rules

Examples

```
library(bupaR)
library(eventdataR)

# Filter where Blood test precedes MRI SCAN and Registration is the start of the case.
patients %>%
 filter_rules(precedence("Blood test","MRI SCAN"),
              starts("Registration"))
```

precedence *Precedence*

Description

Check for precedence between two activities.

If `activity_b` occured, it should be preceded by `activity_a` in the same case, i.e., if `B` was executed, it could not have been executed before `A` was executed. For example, the trace `[A,C,B,B,A]` satisfies the `precedence` relation.

Usage

```
precedence(activity_a, activity_b)
```

Arguments

`activity_a`	`character`: Activity A. This should be an activity of the log supplied to `check_rule`.
`activity_b`	`character`: Activity B. This should be an activity of the log supplied to `check_rule`.

See Also

Other Ordering rules: `ends()`, `responded_existence()`, `response()`, `starts()`, `succession()`

Examples

```
library(bupaR)
library(eventdataR)

# A MRI Scan should be preceeded by a Blood test.

patients %>%
 check_rule(precedence("Blood test","MRI SCAN"))
```

responded_existence *Responded Existence*

Description

Check for responded existence between two activities.

If `activity_a` occurs in a case, `activity_b` should also occur (before or after).

Usage

```
responded_existence(activity_a, activity_b)
```

Arguments

`activity_a`	`character`: Activity A. This should be an activity of the log supplied to `check_rule`.
`activity_b`	`character`: Activity B. This should be an activity of the log supplied to `check_rule`.

See Also

Other Ordering rules: `ends()`, `precedence()`, `response()`, `starts()`, `succession()`

Examples

```
library(bupaR)
library(eventdataR)

# When a Blood test occurs, a MRI Scan should also have
# happened for this patient (before or after the test).

patients %>%
 check_rule(responded_existence("Blood test","MRI SCAN"))
```

response *Response*

Description

Check for response between two activities.

If `activity_a` is executed, it should be (eventually) followed by `activity_b`. The `response` relation is very relaxed, because `B` does not have to be executed immediately after `A`, and multiple `A`s can be executed between the first `A` and the subsequent `B`. For example, the trace `[B,A,A,A,C,B]` satisfies the `response` relation.

Usage

```
response(activity_a, activity_b)
```

Arguments

activity_a	character: Activity A. This should be an activity of the log supplied to check_rule.
activity_b	character: Activity B. This should be an activity of the log supplied to check_rule.

See Also

Other Ordering rules: ends(), precedence(), responded_existence(), starts(), succession()

Examples

```
library(bupaR)
library(eventdataR)

# A blood test should eventually be followed by Discuss Results.
patients %>%
 check_rule(response("Blood test","Discuss Results"))
```

---

starts *Starts*

Description

Check if cases start with the specified activity.

Usage

```
starts(activity)
```

Arguments

activity	character: The start activity. This should be an activity of the log supplied to check_rule.

See Also

Other Ordering rules: ends(), precedence(), responded_existence(), response(), succession()

Examples

```
library(bupaR)
library(eventdataR)

# Each patients should first be registered.
patients %>%
 check_rule(starts("Registration"))
```

---

succession *Succession*

Description

Check for succession between two activities.

succession checks the bi-directional execution order of activity_a and activity_b, i.e., both response and precedence relations have to hold: every A has to be (eventually) followed by B, and there has to be an A before every B. For example, the trace [A,C,A,B,B] satisfies the succession relation.

Usage

```
succession(activity_a, activity_b)
```

Arguments

activity_a	character: Activity A. This should be an activity of the log supplied to check_rule.
activity_b	character: Activity B. This should be an activity of the log supplied to check_rule.

See Also

Other Ordering rules: ends(), precedence(), responded_existence(), response(), starts()

Examples

```
library(bupaR)
library(eventdataR)

# Blood test should always happen before a MRI Scan,
# and both should happen when one of them happens.
patients %>%
 check_rule(succession("Blood test","MRI SCAN"))
```

xor *XOR*

Description

Check for exclusiveness of two activities.

If `activity_a` exists, `activity_b` should not exist, and vice versa.

Usage

```
xor(activity_a, activity_b)
```

Arguments

`activity_a`	`character`: Activity A. This should be an activity of the log supplied to `check_rule`.
`activity_b`	`character`: Activity B. This should be an activity of the log supplied to `check_rule`.

See Also

Other Exclusiveness rules: `and()`

Examples

```
library(bupaR)
library(eventdataR)

# A patient should not receive both an X-Ray and MRI Scan.
patients %>%
 check_rule(xor("X-Ray","MRI SCAN"))
```

Index