Tutorial Rule Checking with bupaR

This tutorial will provide you with the necessary information to perform rule checking using the bupaR library, created by Gert Janssenswillen from Hasselt University. You can check Chapter 4 of the book **Conformance Checking - Relating Processes and Models** for an introduction to rule-based conformance checking.

Rule checking can be done with bupaR using the processcheckR package. To start, we most load both packages.

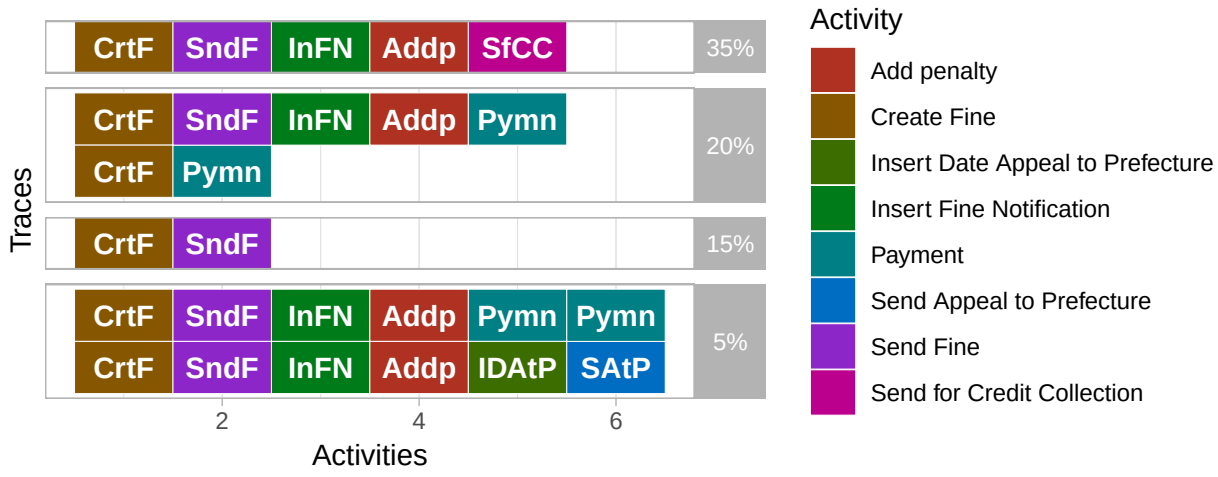
```
library(bupaR)
library(processcheckR)
```

In this tutorial, we will use a small sample of 20 cases of the traffic fines event log provided by bupaR.

```
set.seed(123)
log <- sample_n(traffic_fines, 20)
```

This sample contains 6 different variants, as shown below

```
log %>%
    trace_explorer(coverage = 1)
```



The rules which can be checked are of three types: cardinality, ordering and exclusiveness rules.

Cardinality rules:

- `contains`: activity occurs n times or more
- `contains_exactly`: activity occurs exactly n times
- `contains_between`: activity occurs between min and max times
- `absent`: activity does not occur more than n - 1 times

Ordering rules:

- `starts`: case starts with activity
- `ends`: case ends with activity
- `succession`: if activity A happens, B should happen after. If B happens, A should have happened before.
- `response`: if activity A happens, B should happen after
- `precedence`: if activity B happens, A should have happend before
- `responded_existence`: if activity A happens, B should also (have) happen(ed) (i.e. before or after A)

Exclusiveness:

- `and`: two activities always exist together
- `xor`: two activities are not allowed to exist together

Each rule requires the specification of one (unary rules) or two (binary rules) activities, together with information on cardinality for the first type of rules. To check a rule, we use the `check_rule` function.

For example, we can test whether the cases end with the "Payment" activity as follows. We input the log to the check_rule function where we specific the rule `ends("Payment")`. The optional label argument allows us the give a custom name to the check results variable. We store the checked log as `check_log`.

```
log %>%
    check_rule(ends("Payment"),
               label = "ends_with_payment") -> checked_log
```

The `check_rule` function will create a new case attribute which indicate whether the rule holds for a specific case. We can now count the number of cases for which it holds or not as follows. We group the log based on the check result, and count the number of cases in each group.

```
checked_log %>%
    group_by(ends_with_payment) %>%
    n_cases()
```

```
## # A tibble: 2 x 2
##   ends_with_payment n_cases
##   <lgl>                <int>
## 1 FALSE                   11
## 2 TRUE                     9
```

Here, FALSE indicates that the rule is violated, whereas TRUE means the rule is not violated. As you can see, 11 out of the 20 cases violate this rule and thus end with another activity.

Another rule we can test is whether or not the "Send Fine" activity is always followed with a "Insert Fine Notification". This is a `response` rule. We can add this rule to the `checked_log`.

```
checked_log %>%
    check_rule(response("Send Fine", "Insert Fine Notification"),
               label = "fine_notification") -> checked_log
```

Again, we can count the cases where the rule holds or not. In this case, the rule is violated in 3 cases.

```
checked_log %>%
    group_by(fine_notification) %>%
    n_cases()
```

```
## # A tibble: 2 x 2
##   fine_notification n_cases
##   <lgl>                <int>
## 1 FALSE                    3
## 2 TRUE                    17
```

Using the newly created rule attributes, we can use them in a flexible way to look at the rule violations. For instance, we could look at the traces (process variants) for which a rule, or a combination of rules is violated.

```
checked_log %>%
    group_by(ends_with_payment, fine_notification) %>%
    traces
```

```
## # A tibble: 6 x 5
##   ends_with_payment fine_notificati~ trace                absolute_freque~
##   <lgl>             <lgl>            <chr>                           <int>
```

```
## 1 TRUE              TRUE             Create Fine,Payment                4
## 2 TRUE              TRUE             Create Fine,Send Fi~               4
## 3 TRUE              TRUE             Create Fine,Send Fi~               1
## 4 FALSE             FALSE            Create Fine,Send Fi~               3
## 5 FALSE             TRUE             Create Fine,Send Fi~               7
## 6 FALSE             TRUE             Create Fine,Send Fi~               1
## # ... with 1 more variable: relative_frequency <dbl>
```

Also, it is very easy to combine different rules logically. For example, we want to make sure that cases contain exactly one payment or either Send for Credit Collection or Appeal to Prefecture.

```
log %>%
    check_rule(contains_exactly("Payment", 1), "r1") %>%
    check_rule(contains("Send for Credit Collection"), "r2") %>%
    check_rule(contains("Send Appeal to Prefecture"), "r3") -> checked_log
```

Rule 1 (`r1`) checks that there is exactly one payment. Rule 2 (`r2`) and 3 (`r3`) check whether activities Send for Credit Collection and Send Appeal to Prefecture exist, respectively. We can now make sure that if a payment exists, the other activities should be absent, by adding a new attribute which is a logical combination of the specified rules.

```
checked_log %>%
    # It is not allowed that rule 1 holds and rule 2 or 3 hold
    mutate(r4 = !(r1 & (r2|r3))) %>%
    group_by(r4) %>%
    traces
```

```
## # A tibble: 6 x 4
##   r4    trace                              absolute_freque~ relative_freque~
##   <lgl> <chr>                                       <int>            <dbl>
## 1 TRUE  Create Fine,Payment                             4           0.200
## 2 TRUE  Create Fine,Send Fine,Insert Fi~                4           0.200
## 3 TRUE  Create Fine,Send Fine,Insert Fi~                1           0.0500
## 4 TRUE  Create Fine,Send Fine                           3           0.150
## 5 TRUE  Create Fine,Send Fine,Insert Fi~                7           0.350
## 6 TRUE  Create Fine,Send Fine,Insert Fi~                1           0.0500
```

In the last table, we can see that this rule holds for all the cases in the data, i.e. there are no cases with Payments as well as a Send activity.