Package ‘pm4py’

January 7, 2020

Type  Package
Title  Interface to the 'PM4py' Process Mining Library
Version  1.2.7
Description  Interface to 'PM4py' <http://pm4py.org>, a process mining library in 'Python'. This package uses the 'reticulate' package to act as a bridge between 'PM4py' and the 'R' package 'bupaR'. It provides several process discovery algorithms, evaluation measures, and alignments.
License  GPL-3
Encoding  UTF-8
LazyData  true
SystemRequirements  Python (>= 3.6)
Imports  reticulate (>= 1.11), bupaR, petrinetR, purrr, stringr
RoxygenNote  7.0.2
Suggests  testthat, eventdataR
URL  https://github.com/fmannhardt/pm4py
BugReports  https://github.com/fmannhardt/pm4py/issues
NeedsCompilation  no
Author  Felix Mannhardt [aut, cre]
Maintainer  Felix Mannhardt <felix.mannhardt@sintef.no>
Repository  CRAN
Date/Publication  2020-01-07 18:40:02 UTC

R topics documented:

  as_pm4py_marking ........................................... 2
  conformance ............................................... 3
  discovery .................................................. 4
  evaluation ............................................... 6
  install_pm4py ............................................. 7
as_pm4py_marking  

Convert to a PM4Py marking

Description
Converts a character vector of place identifiers to a PM4Py marking object.

Usage
as_pm4py_marking(x, petrinet)

Arguments
x A character vector with (possible duplicate) place identifiers.
petrinet A PM4Py Petri net.

Examples
if (pm4py_available()) {
  library(eventdataR)
  data(patients)

  # As Inductive Miner of PM4PY is not life-cycle aware, keep only 'complete' events:
  patients_completes <- patients[patients$registration_type == "complete", ]

  net <- discovery_inductive(patients_completes)
  as_pm4py_marking(c("sink"), r_to_py(net$petrinet))
}
**Conformance between an Event Log and a Petri net**

**Description**

Conformance between an Event Log and a Petri net

**Usage**

```r
conformance_alignment(
  eventlog,
  petrinet,
  initial_marking,
  final_marking,
  parameters = default_parameters(eventlog),
  variant = variant_state_equation_a_star(),
  convert = TRUE
)

variant_state_equation_a_star()

variant_dijkstra_no_heuristics()
```

**Arguments**

- `eventlog`: A bupaR or PM4PY event log.
- `petrinet`: A bupaR or PM4PY Petri net.
- `initial_marking`: A R vector with the place identifiers of the initial marking or a PM4PY marking. By default the initial marking of the bupaR Petri net will be used if available.
- `final_marking`: A R vector with the place identifiers of the final marking or a PM4PY marking.
- `parameters`: PM4PY conformance parameter. By default the activity_key from the bupaR event log is specified using `param_activity_key`.
- `variant`: The conformance variant to be used.
- `convert`: TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the `r-py-conversion` function.

**Value**

A data frame describing the conformance result. In case of `conformance_alignment` a data frame of log and model moves.
Examples

```r
if (pm4py_available()) {
  library(eventdataR)
  data(patients)

  # As Inductive Miner of PM4Py is not life-cycle aware, keep only 'complete' events:
  patients_completes <- patients[patients$registration_type == "complete", ]

  # Discover a Petri net
  net <- discovery_inductive(patients_completes)

  # Align event log and Petri net
  a <- conformance_alignment(patients_completes,
                             net$petrinet,
                             net$initial_marking,
                             net$final_marking)

  # Alignment is returned as data frame
  head(a)
}
```

---

### discovery

**Petri net discovery algorithms**

**Description**

PM4PY discovery algorithms that discover a Petri net and its initial and final marking. Currently the Inductive Miner and the Alpha Miner are implemented.

**Usage**

```r
discovery_inductive(
  eventlog,
  parameters = default_parameters(eventlog),
  variant = variant_inductive_imdfb(),
  convert = TRUE
)
```

```r
variant_inductive_imdfb()
```

```r
variant_inductive_only_dfg()
```

```r
discovery_alpha(
  eventlog,
  parameters = default_parameters(eventlog),
  variant = variant_alpha_classic(),
  convert = TRUE
)
```
discovery

)

variant_alpha_classic()

variant_alpha_plus()

Arguments

eventlog A bupaR event log or an R data frame.

parameters A named list of PM4PY parameters (see parameters) as required by the discovery method. By default, if the eventlog is a bupaR event log, the activity_key, timestamp_key, and caseid_key are automatically determined.

variant The variant of the discovery algorithm to be used. For Inductive Miner currently only variant_inductive_imdfb is supported.

convert TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the r-py-conversion function.

Value

A named list with elements petrinet, initial_marking, and final_marking or the original Python object.

Examples

if (pm4py_available()) {
  library(eventdataR)
data(patients)

  # As Inductive Miner of PM4PY is not life-cycle aware, keep only `complete` events:
  patients_completes <- patients[patients$registration_type == "complete", ]

  net <- discovery_inductive(patients_completes)

  # Show details of the obtained bupaR Petri net
  print(net$petrinet)

  # initial marking is a character vector
  print(net$initial_marking)

  # final marking is a character vector
  print(net$final_marking)

  # Petri net can be used with other bupaR functions
  petrinetR::render_PN(net$petrinet)

  # Keep an unconverted PM4PY Petri net for use in other PM4PY functions
  py_net <- discovery_inductive(patients_completes, convert = FALSE)
}


evaluation

Calculates evaluation measures for a Petri nets and an Event Log

Description

Calculates evaluation measures for a Petri nets and an Event Log

Usage

`evaluation_all(
    eventlog,
    petrinet,
    initial_marking,
    final_marking,
    parameters = default_parameters(eventlog),
    convert = TRUE
)

evaluation_precision(
    eventlog,
    petrinet,
    initial_marking,
    final_marking,
    parameters = default_parameters(eventlog),
    variant = variant_precision_etconformance(),
    convert = TRUE
)

variant_precision_etconformance()

evaluation_fitness(
    eventlog,
    petrinet,
    initial_marking,
    final_marking,
    parameters = default_parameters(eventlog),
    variant = variant_fitness_token_based(),
    convert = TRUE
)

variant_fitness_token_based()

variant_fitness_alignment_based()

Arguments

`eventlog`  A bupaR or PM4PY event log.
install_pm4py

petrinet A bupaR or PM4PY Petri net.
initial_marking A R vector with the place identifiers of the initial marking or a PM4PY marking. By default the initial marking of the bupaR Petri net will be used if available.
final_marking A R vector with the place identifiers of the final marking or a PM4PY marking.
parameters PM4PY alignment parameter. By default the activity_key from the bupaR event log is specified using param_activity_key.
convert TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the r-py-conversion function.
variant The evaluation variant to be used.

Value
A list with all available evaluation measures.

Examples

```r
if (pm4py_available()) {
  library(eventdataR)
  data(patients)

  # As Inductive Miner of PM4PY is not life-cycle aware, keep only 'complete' events:
  patients_completes <- patients[patients$registration_type == "complete", ]

  # Discover a Petri net
  net <- discovery_inductive(patients_completes)

  # Calculate evaluation measures for event log and Petri net
  evaluation_all(patients_completes,
                 net$petrinet,
                 net$initial_marking,
                 net$final_marking)
}
```

install_pm4py Install PM4PY library

Description
Installs the pm4py package and its dependencies using pip since no Conda package is available. Further information on the parameters can be found in the reticulate package documentation: https://rstudio.github.io/reticulate/ In some cases (multiple Python versions installed) it might be useful to specify the exact path to the conda binary.

Usage

```r
install_pm4py(method = "auto", conda = "auto", ...)
```
Arguments

method  Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows.

conda  Path to conda executable (or "auto" to find conda using the PATH and other conventional install locations).

...  Additional arguments passed to py_install().

Details

Additional requirements (a C++ compiler and GraphViz) of PM4PY might need to be installed to leverage all functionality; http://pm4py.pads.rwth-aachen.de/installation/

Examples

pm4py::install_pm4py()

# Specify path to conda
pm4py::install_pm4py(method = "conda", conda = "/home/user/miniconda3/bin/conda")

parameters  PM4Py parameter keys

| Description | Convenience methods to use as PM4Py parameter keys. |
| Usage | param_activity_key(value) |
| | param_attribute_key(value) |
| | param_timestamp_key(value) |
| | param_caseid_key(value) |
| | param_resource_key(value) |
| | default_parameters(eventlog) |

Arguments

value  The value to add to the list.

eventlog  A bupaR or PM4PY event log.
petrinet_check_relaxed_soundness

Value

a list with the parameter key/value pair

Examples

param_activity_key("activity")

library(eventdataR)
data(patients)
default_parameters(patients)

petrinet_check_relaxed_soundness

Check Relaxed soundness property

Description

Checks if the Petri net is relaxed sound

Usage

petrinet_check_relaxed_soundness(pn, im = NULL, fm = NULL, convert = TRUE)

Arguments

pn Petri net
im Initial marking of the Petri net (optional for workflow nets)
fm Final marking of the Petri net (optional for workflow nets)
convert TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the r-py-conversion function.

Value

A single logical

Examples

if (pm4py_available()) {
  library(eventdataR)
data(patients)

  # As Inductive Miner of PM4PY is not life-cycle aware, keep only `complete` events:
  patients_completes <- patients[patients$registration_type == "complete", ]

  net <- discovery_inductive(patients_completes)
  petrinet_check_relaxed_soundness(net$petrinet)
}


petrinet_check_wfnet  Check Workflow net property

Description

Checks if the Petri net is a Workflow net

Usage

petrinet_check_wfnet(pn, convert = TRUE)

Arguments

pn         Petri net
convert     TRUE to automatically convert Python objects to their R equivalent. If you pass
            FALSE you can do manual conversion using the r-py-conversion function.

Value

A single logical

Examples

if (pm4py_available()) {
  library(eventdataR)
  data(patients)

  # As Inductive Miner of PM4PY is not life-cycle aware, keep only 'complete' events:
  patients_completes <- patients[patients$registration_type == "complete", ]

  net <- discovery_inductive(patients_completes)
  petrinet_check_wfnet(net$petrinet)
}

petrinet_synchronous_product  Synchronous product Petri net

Description

Constructs the synchronous product net of two given Petri nets.
petrinet_synchronous_product

Usage

```r
petrinet_synchronous_product(
    pn1,
    im1,
    fm1,
    pn2,
    im2,
    fm2,
    skip = ">>",
    convert = TRUE
)
```

Arguments

- **pn1**: First Petri net
- **im1**: Initial marking of the first Petri net
- **fm1**: Final marking of the first Petri net
- **pn2**: Second Petri net
- **im2**: Initial marking of the second Petri net
- **fm2**: Final marking of the second Petri net
- **skip**: Symbol to be used as skip
- **convert**: TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the r-py-conversion function.

Value

A Petri net.

Examples

```r
if (pm4py_available()) {
    library(eventdataR)
    data(patients)

    # As Inductive Miner of PM4PY is not life-cycle aware, keep only 'complete' events:
    patients_completes <- patients[patients$registration_type == "complete", ]

    net <- discovery_inductive(patients_completes)
    petrinet_synchronous_product(net$petrinet,
                                  net$initial_marking,
                                  net$final_marking,
                                  net$petrinet,
                                  net$initial_marking,
                                  net$final_marking)
}
```
pm4py

PM4PY for R

Description
This package provides access to the Python Process Mining library PM4PY in R and provides conversion between bupaR and PM4PY data structures.

Usage
pm4py

Format
An object of class python.builtin.module (inherits from python.builtin.object) of length 2.

Details
To use this package, you need to have a Python environment (Conda or virtualenv) installed and install the PM4PY package and its dependencies. You can use the convenience function install_pm4py to let reticulate take care of install the right version. See the documentation of this function for further information.

When loaded, the object pm4py provides the low-level interface to the main PM4PY module. Use $ to access sub modules of PM4PY as described in the reticulate documentation:
  vignette("calling_python",package = "reticulate")

For parts of PM4PY wrapper functions are provided to transparently convert parameters and results to and from the corresponding bupaR S3 classes.

Examples
# Print the PM4PY version loaded
if (pm4py_available()) {
  print(pm4py$`__version__`)
}

pm4py_available

Is the PM4Py module available

Description
Is the PM4Py module available

Usage
pm4py_available()
Value

TRUE is PM4Py is installed

Examples

```python
if (pm4py_available()) {
    print(pm4py_version())
}
```

---

**pm4py_version**

*Returns PM4Py version used*

Description

Returns PM4Py version used

Usage

```python
pm4py_version()
```

Value

package_version S3 class

Examples

```python
if (pm4py_available()) {
    print(pm4py_version())
}
```

---

**version**

*This function is deprecated, please use pm4py_version.*

Description

This function is deprecated, please use pm4py_version.

Usage

```python
version()
```

Value

package_version S3 class
write_Pnml

Write Petri net as PNML

Description
Write Petri net as PNML

Usage
write_Pnml(petrinet, file, initial_marking = NULL, final_marking = NULL)

Arguments
- petrinet: A bupaR or PM4PY Petri net.
- file: File name of the PNML file
- initial_marking: A R vector with the place identifiers of the initial marking or a PM4PY marking. By default the initial marking of the bupaR Petri net will be used if available.
- final_marking: A R vector with the place identifiers of the final marking or a PM4PY marking.

Examples
# don't test automatically since this writes a file

if (pm4py_available()){
  library(eventdataR)
  data(patients)

  # As Inductive Miner of PM4PY is not life-cycle aware, keep only `complete` events:
  patients_completes <- patients[patients$registration_type == "complete", ]

  net <- discovery_inductive(patients_completes)
  write_Pnml(net$petrinet,
              "test.pnml",
              net$initial_marking,
              net$final_marking)

# Index

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasets</td>
<td>12</td>
</tr>
<tr>
<td>pm4py</td>
<td>12</td>
</tr>
<tr>
<td>as_pm4py_marking</td>
<td>2</td>
</tr>
<tr>
<td>conformance</td>
<td>3</td>
</tr>
<tr>
<td>conformance_alignment</td>
<td>3</td>
</tr>
<tr>
<td>default_parameters</td>
<td>8</td>
</tr>
<tr>
<td>discovery</td>
<td>4</td>
</tr>
<tr>
<td>discovery_alpha</td>
<td>4</td>
</tr>
<tr>
<td>discovery_inductive</td>
<td>4</td>
</tr>
<tr>
<td>evaluation</td>
<td>6</td>
</tr>
<tr>
<td>evaluation_all</td>
<td>6</td>
</tr>
<tr>
<td>evaluation_fitness</td>
<td>6</td>
</tr>
<tr>
<td>evaluation_precision</td>
<td>6</td>
</tr>
<tr>
<td>install_pm4py</td>
<td>7, 12</td>
</tr>
<tr>
<td>variant_alpha_classic</td>
<td>4</td>
</tr>
<tr>
<td>variant_alpha_plus</td>
<td>4</td>
</tr>
<tr>
<td>variant_dijkstra_no_heuristics</td>
<td>3</td>
</tr>
<tr>
<td>variant_fitness_alignment_based</td>
<td>6</td>
</tr>
<tr>
<td>variant_fitness_token_based</td>
<td>6</td>
</tr>
<tr>
<td>variant_inductive_imdfb</td>
<td>4</td>
</tr>
<tr>
<td>variant_inductive_only_dfg</td>
<td>4</td>
</tr>
<tr>
<td>variant_precision_etconformance</td>
<td>6</td>
</tr>
<tr>
<td>variant_state_equation_a_star</td>
<td>3</td>
</tr>
<tr>
<td>version</td>
<td>13</td>
</tr>
<tr>
<td>write_pnml</td>
<td>14</td>
</tr>
<tr>
<td>petrinet_check_relaxed_soundness</td>
<td>9</td>
</tr>
<tr>
<td>petrinet_check_wfnet</td>
<td>10</td>
</tr>
<tr>
<td>petrinet_synchronous_product</td>
<td>10</td>
</tr>
<tr>
<td>pm4py</td>
<td>12</td>
</tr>
<tr>
<td>pm4py_available</td>
<td>12</td>
</tr>
<tr>
<td>pm4py_version</td>
<td>13, 13</td>
</tr>
<tr>
<td>r-py-conversion</td>
<td>3, 5, 7, 9–11</td>
</tr>
<tr>
<td>variant_fitness_alignment_based (evaluation)</td>
<td>6</td>
</tr>
<tr>
<td>variant_fitness_token_based (evaluation)</td>
<td>6</td>
</tr>
<tr>
<td>variant_inductive_imdfb (discovery)</td>
<td>4</td>
</tr>
<tr>
<td>variant_inductive_only_dfg (discovery)</td>
<td>4</td>
</tr>
<tr>
<td>variant_precision_etconformance (evaluation)</td>
<td>6</td>
</tr>
<tr>
<td>variant_state_equation_a_star (conformance)</td>
<td>3</td>
</tr>
<tr>
<td>version</td>
<td>13</td>
</tr>
<tr>
<td>write_pnml</td>
<td>14</td>
</tr>
</tbody>
</table>