Automaton Documentation

Release 1.3.1

Federico Ficarelli

Jan 23, 2019

Contents

Automaton is a minimal and easy to use finite-state machine definition package for Python. The focus here is on *minimalism* and *not* on feature-richness.

Contents:

CHAPTER 1

Reference

A minimal finite-state machine implementation.

class automaton.**Event**

Bases: *automaton.Event*

Class that represents a state transition.

Important: This is a read-only data descriptor type.

Parameters

- **source_states** (*any*) – The transition source state.
- **dest_state** (*any*) – The transition destination state.

edges(*data=False*)

Provides all the single transition edges associated to this event.

Note: Since an event can have multiple source states, in graph terms it represents a set of state-to-state edges.

Parameters **data** (*bool, optional*) – If set, data associated to the corresponding edge will be added to the edge tuple. Defaults to *False*.

Yields *(any, any) or (any, any, dict)* – All the *(source, dest)* tuples representing the graph edges associated to the event. If *data* parameter is set, each tuple will be appended with the *dict* containing edge data (by default the *'event'* key containing the event name).

name

The actual user-defined name of the event as an *Automaton* class member. If None, the Event isn't bounded to any specific *Automaton* subclass.

class automaton.**Automaton**(*initial_state=None*, *initial_event=None*, *accepting_states=None*)

Bases: object

Base class for automaton types.

In order to define an automaton, this class must be the base for the provided machine definition:

```
>>> class MyClass(Automaton):
...     transition = Event("state_a", "state_b")
```

Parameters

- **initial_state** (*any, optional*) – The initial state for this automaton instance. Defaults to None. Note that if automaton type has no default initial state (specified via __default_initial_state__), this argument must be specified.
- **initial_event** (*any, optional*) – The initial event to be fired to deduce the initial state. Defaults to *None*. Please note that *initial_state* and and *initial_event* arguments *are mutually exclusive*, specifying both of them will raise a *TypeError*.

Note: Since the *destination state* of an event is a *single state* and due to the fact that events are class attributes (so each event name is unique), *deducing the initial state through a initial event is a well defined operation*.

- **accepting_states** (*iterable, optional*) – The accepting states for this automaton instance. Defaults to None. Note that if automaton type has default accepting states (specified via __default_accepting_states__), this argument takes precedence over that.

Raises

- *DefinitionError* – The automaton type has no default initial state while no custom initial state specified during construction *or* the specified initial state is unknown.
- TypeError – Both *initial_state* and *initial_event* arguments have been specified while they are mutually exclusive.

event(*event_name*)

Signals the occurrence of an event and evolves the automaton to the destination state.

Parameters **do_event** (*any*) – The event to be performed on the automaton.

Raises *InvalidTransitionError* – The specified event is unknown.

classmethod events()

Gives the automaton events set.

Yields *any* – The iterator over the events set.

classmethod get_default_initial_state()

Gives the automaton default initial state.

Returns The automaton default initial state.

Return type any

classmethod in_events(**states*)

Retrieves all the inbound events entering the specified states with no duplicates.

Parameters **states** (*tuple(any)*) – The states subset.

Raises KeyError – When an unknown state is found while iterating.

Yields *any* – The events entering the specified states.

is_accepted
Gets the current automaton's acceptance state.

Returns True if the current state is an accepting state, False otherwise.

Return type bool

classmethod out_events(**states*)
Retrieves all the outbound events leaving the specified states with no duplicates.

Raises KeyError – When an unknown state is found while iterating.

Parameters **states** (*tuple(any)*) – The states subset.

Yields *any* – The events that exit the specified states.

state
Gets the current state of the automaton instance.

Returns The current state.

Return type any

classmethod states()
Gives the automaton state set.

Yields *any* – The iterator over the state set.

exception automaton.**AutomatonError**
Bases: Exception

Exception representing a generic error occurred in the automaton.

exception automaton.**DefinitionError**
Bases: *automaton.AutomatonError*

Exception representing an error occurred during the automaton definition.

exception automaton.**InvalidTransitionError**
Bases: *automaton.AutomatonError*

Exception representing an invalid transition.

automaton.**transitiontable**(*automaton*, *header=None*, *fmt=None*, *traversal=None*)
Render an automaton's transition table in text format.

The transition table has three columns: source node, destination node and the name of the event.

A simple example will be formatted as follows:

```
>>> class TrafficLight(Automaton):
...     go = Event('red', 'green')
...     slowdown = Event('green', 'yellow')
...     stop = Event('yellow', 'red')
>>> tabulate(TrafficLight)
```

```
========  ======  ========
Source    Dest    Event
========  ======  ========
green     yellow  slowdown
yellow    red     stop
```

(continues on next page)

(continued from previous page)

```
red       green   go
========  ======  ========
```

Parameters

- **automaton** (*Automaton*) – The automaton to be rendered. It can be both a class and an instance.
- **header** (*list[str, str, str]*) – An optional list of fields to be used as table headers. Defaults to a predefined header.
- **str, optional** (*fmt*) – Specifies the output format for the table. All formats supported by tabulate package are supported (e.g.: `rst` for reStructuredText, `pipe` for Markdown). Defaults to `rst`.
- **traversal** (*callable(graph), optional*) – An optional callable used to sort the events. It has the same meaning as the `traversal` parameter of *automaton.transition_table*.

Returns Returns the formatted transition table.

Return type str

automaton.**stategraph**(*automaton*, *fmt=None*, *traversal=None*)

Render an automaton's state-transition graph.

A simple example will be formatted as follows:

```
>>> class TrafficLight(Automaton):
...     go = Event('red', 'green')
...     slowdown = Event('green', 'yellow')
...     stop = Event('yellow', 'red')
>>> print( plantuml(TrafficLight) )
```

```
@startuml
    green --> yellow : slowdown
    yellow --> red : stop
    red --> green : go
@enduml
```

Parameters

- **automaton** (*Automaton*) – The automaton to be rendered. It can be both a class and an instance.
- **fmt** (*str, optional*) – Specifies the output format for the graph. Currently, the only supported format is PlantUML. Defaults to `plantuml`, unknown format specifiers are ignored.
- **traversal** (*callable(graph), optional*) – An optional callable used to sort the events. It has the same meaning as the `traversal` parameter of *automaton.transition_table*.

Returns Returns the formatted state graph.

Return type str

automaton.**get_table**(*automaton*, *traversal=None*)

Build the transition table of the given automaton.

Parameters

- **automaton** (*Automaton*) – The automaton to be rendered. It can be both a class and an instance.
- **traversal** (*callable(graph), optional*) – An optional callable used to yield the edges of the graph representation of the automaton. It must accept a *networkx.MultiDiGraph* as the first positional argument and yield one edge at a time as a tuple in the form `(source_state, destination_state)`. The default traversal sorts the states in ascending order by inbound grade (number of incoming events).

Yields *tuple(source, dest, event)* – Yields one row at a time as a tuple containing the source and destination node of the edge and the name of the event associated with the edge.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index