past-mtl-monitors

Release 0.0.1

Marcell Vazquez-Chanlatte

Apr 21, 2021

CONTENTS:

A library for creating past metric temporal logic monitors.

Basic Usage:

```
from past_mtl_monitors import atom

x, y, z = atom('x'), atom('y'), atom('z')

# Monitor that historically, x has been equal to y.
monitor = (x == y).hist().monitor()

#                    time         values
assert monitor.send((0    , {'x': 1, 'y': 1}))  ==  1   # sat
assert monitor.send((1.1  , {'x': 1, 'y': -1})) == -1   # unsat
assert monitor.send((1.5  , {'x': 1, 'y': 1}))  == -1   # unsat

monitor2 = x.once().monitor()  # Monitor's x's maximum value.
assert monitor2.send((0 , {'x': -10, 'y': 1})) == -10
assert monitor2.send((0 , {'x': 100, 'y': 2})) == 100
assert monitor2.send((0 , {'x': -100, 'y': -1})) == 100

# Monitor that x & y have been true since the last
# time that z held for 3 time units.
monitor3 = (x & y).since(z.hist(0, 3)).monitor()
```

CHAPTER

ONE

INSTALLATION

If you just need to use *past-mtl-monitors*, you can just run:

```
$ pip install past-mtl-monitors
```

For developers, note that this project uses the [poetry](https://poetry.eustace.io/) python package/dependency management tool. Please familarize yourself with it and then run:

```
$ poetry install
```

CHAPTER

TWO

API

The past-mtl-monitor API centers around the MonitorFactory type with atom as the primary entrypoint into the API.

past_mtl_monitors.**atom**(*var*)

Main entry point to monitor construction DSL.

Takes a variable name and produces a monitor factory.

Parameters **var** (str) –

Return type MonitorFact

class past_mtl_monitors.**MonitorFact**(*factory*)

Parameters **factory** (*Factory*) –

hist(*start=0*, *end=inf*)

Monitors if the child monitor was historically true over the interval [t-end, t-start] where t is the current time.

Return type MonitorFact

once(*start=0*, *end=inf*)

Monitors if the child monitor was once true in the interval [t-end, t-start] where t is the current time.

Return type MonitorFact

since(*other*)

Monitors the minimum value since the last time other's value was greater than 0.

Note: other's value is assumed to have been previously greater than zero when the monitor starts.

Parameters **other** (MonitorFact) –

Return type MonitorFact

implies(*other*)

Monitors if child monitor self implies child monitor other.

Parameters **other** (MonitorFact) –

Return type MonitorFact

__and__(*other*)

Combines child monitors using min. If values are {1, -1} for True and False resp., then this corresponds to logical And.

Parameters **other** (MonitorFact) –

Return type MonitorFact

__or__(*other*)
Combines child monitors using min. If values are {1, -1} for True and False resp., then this corresponds to logical Or.

Parameters **other** (MonitorFact) –

Return type MonitorFact

__invert__()
Negates result of child monitors.

Return type MonitorFact

__eq__(*other*)
Monitors if child monitors self and other return same values.

Parameters **other** (MonitorFact) –

Return type MonitorFact

CHAPTER

THREE

EXTENDING

The *MonitorFact* type is a simple wrapper around python co-routines. This makes is easy to write your own monitors that integrate well with the *past-mtl-monitor* library. For example: consider the following monitor which aggregates the child's monitor's results:

```
from past_mtl_monitors import MonitorFact


def aggregate_monitor(child_factory):
    """Sums the child values using piecewise constant interpolation."""

    def avg_factory():
        child_monitor = child_factory.monitor()
        payload = (time, _) = yield    # Get initial payload.
                                       # payload = (time, child input).

        total = prev_val = 0
        while True:
            child_val = child_monitor.send(payload)

            prev_time = time
            payload = (time, _) = yield total

            total += prev_val * (time - prev_time)
            prev_val = child_val

    return MonitorFact(avg)
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

H

I

M

O

S