callgraph Documentation

Release 1.0.0

Oliver Steele

Jun 14, 2020

Contents:

Callgraph is a Python package that defines a decorator, and Jupyter magic, to draw dynamic call graphs of Python function calls.

It's intended for classroom use, but may also be useful for self-guided exploration.

The package defines a Jupyter IPython magic, `%callgraph`, that displays a call graph within a Jupyter cell:

```
from functools import lru_cache

@lru_cache()
def lev(a, b):
    if "" in (a, b):
        return len(a) + len(b)

    candidates = []
    if a[0] == b[0]:
        candidates.append(lev(a[1:], b[1:]))
    else:
        candidates.append(lev(a[1:], b[1:]) + 1)
    candidates.append(lev(a, b[1:]) + 1)
    candidates.append(lev(a[1:], b) + 1)
    return min(candidates)

%callgraph -w10 lev("big", "dog"); lev("dig", "dog")
```

It also provides a Python decorator, `callgraph.decorator`, that instruments a function to collect call graph information and render the result.

CHAPTER 1

Jupyter / IPython Usage

```
$ pip install callgraph
```

In a Jupyter IPython notebook:

```
%load_ext callgraph

def nchoosek(n, k):
    if k == 0:
        return 1
    if n == k:
        return 1
    return nchoosek(n - 1, k - 1) + nchoosek(n - 1, k)

%callgraph nchoosek(4, 2)
```

As an alternative to including `%load_ext callgraph` in each notebook that uses `%callgraph`, you can add the extension to the Notebook configuration file in your IPython profile.

Your configuration file is probably called `~/.ipython/profile_default/ipython_config.py`. (You can run `ipython profile locate` to find it.) Edit this file to include the following line:

```
c.InteractiveShellApp.extensions = ["callgraph.extension"]
```

(If your configuration file already includes an uncommented statement `c.InteractiveShellApp.extensions = [...]`, edit the list of extensions in that line to include `"callgraph.extension"`.

See extension example notebook for additional examples.

CHAPTER 2

Decorator Usage

```
$ pip install callgraph
```

```
from functools import lru_cache
import callgraph.decorator as callgraph

@callgraph()
@lru_cache()
def nchoosek(n, k):
    if k == 0:
        return 1
    if n == k:
        return 1
    return nchoosek(n - 1, k - 1) + nchoosek(n - 1, k)

nchoosek(5, 2)

nchoosek.__callgraph__.view()
```

See the API documentation for additional documentation.

See the decorator example notebook for additional instructions and examples.

CHAPTER 3

Development

Install dev tools, and set up a Jupyter kernel for the current python enviromnent:

```
$ pip install -r requirements-dev.txt
$ python -m ipykernel install --user
```

Install locally:

```
flit install --symlink
```

CHAPTER 4

Acknowledgements

Callgraph uses the Python graphviz package. Python graphviz uses the Graphviz package.

CHAPTER 5

License

MIT

CHAPTER 6

API

This package defines decorators and IPython magic to display a dynamic call graph.

`callgraph.`**`load_ipython_extension`**(*ipython*)

Register the IPython magic.

Jupyter / IPython calls this when the extension is loaded. You don't need to.

See the package documentation for instructions on how to tell Jupyter to load the extension.

`callgraph.`**`decorator`**(*fn=None*, *recorder=None*, *label_returns=False*, *graph_attrs=None*)

Instrument a function to record calls for the call graph.

Decorator that wraps a function with instrumentation to record calls to it, for use in constructing a call graph.

Parameters

- **`recorder`** (`CallGraphRecorder`, *`optional`*) – An CallGraphRecorder. If this is not supplied, a new recorder is created with the specified values for `label_returns` and `graph_attrs`, and attached to the decorated function as `fn.__callgraph__`.
- **`label_returns`** (*`bool`*) – If true, arrows are draw from callee to caller, and labeled with the return value.
- **`graph_attrs`** (*`dict`*) – Graphviz graph attributes. These are passed to `graphviz.Digraph.attr()`. A new `graphviz.Digraph`.

Examples

```
import callgraph.decorator as callgraph

@callgraph()
def nchoosek(n, k):
    if k == 0:
        return 1
    if n == k:
```

(continues on next page)

(continued from previous page)

```
        return 1
    return nchoosek(n - 1, k - 1) + nchoosek(n - 1, k)
```

class callgraph.**CallGraphRecorder**(*equal=False*, *label_returns=False*, *graph_attrs=None*)

Record function calls into a Graphviz diagraph.

graph

A `graphviz.Digraph`.

Type Digraph

record(*fn*, *args*, *kwargs*)

Return a context manager that records a function call.

Returns A context manager that records a function call.

Return type CallGraphCallRecorder

Examples

```
with recorder.record(fn, args, kwargs) as record_return:
    result = fn(*args, **kwargs)
    record_return(result)
```

wrap(*fn*)

Wrap fn with instrumentation to record calls to it.

You probably want `decorator()` instead.

Python Module Index

Index