sympy-addons

Release 0.0.5

Matthias Baer

Oct 19, 2022

CONTENTS:

Useful tools for working with SymPy.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

sympy-addons can be installed using pip:

```
pip install --upgrade sympy-addons
```

1.2 Usage

1.2.1 The Query API

The query API allows to select specific subexpressions within SymPy expressions. The examples in this section use the following expression for performing queries:

$$(x-1)^2 + \frac{(x-4)^3 + (x+2)^2 + \sin(z)}{\sqrt{(x-1)^2 + (x+3)^2}}$$

or in SymPy:

```
from sympy import *
from sympy.abc import x, z
expr = (x - 1) ** 2 + ((x + 2) ** 2 + (x - 4) ** 3 + sin(z)) / sqrt((x - 1) ** 2 + (x +␣
↪3) ** 2)
```

Querying for types

To get all subexpression matching a given type, e.g. *Pow*:

```
from sympy_addons import Query

# define the query:
query = Query(type=Pow)

# execute it on an expression:
result = query.run(expr)

# result is an instance of QueryResult. You can iterate over it:
```

(continues on next page)

(continued from previous page)

```
for item in result:
    print(item)

# or get the result as a list:
this_is_a_list_of_matching_expressions = result.all()

# or just the first/last:
first_matching_expr = result.first()
last_matching_expr = result.last()
```

Querying for inherited types

To find all subexpressions that are instances of, say, *Atom* and all classes inheriting from *Atom*, use the *isinstance* keyword:

```
query = Query(isinstance=Atom)
```

Querying for arguments

Non-atomic types in SymPy have an *args* attribute. You can query for subexpression which have exactly the *args* that you look for (order does not matter). For example,

will return all subexpressions with *args*==*(x, 1)* or *args*==*(1, x)*.

If you don't want to specify all *args*, use *args__contains* instead:

will return all subexpression with *args* attribute containing *x*.

Custom tests

You can define your own predicates to query for. For instance, to query for subexpressions with exactly three arguments, you could write

Chaining queries

Each query is defined as one predicate. But you can concatenate queries to combine them logically.

For a logical OR, use the | operator:

For an AND operation, use the *filter* method:

1.3 Getting *EPaths*

The *epath* function in SymPy allows to work directly on subexpressions. As input, it needs the path to the subexpression, which is often cumbersome to get. The *get_paths* and *get_path* facilitate getting those paths.

For example,

returns

```
['/[0]', '/[1]/[0]/[0]/[0]']
```

To only expand the $(x-1)^2$ under the square root, we would need the second path:

The *get_path* function works just as the *get_paths* function, but it will raise an exception if the expression is not found or not unique.

CHAPTER

TWO

API REFERENCE

2.1 Module *query*

class sympy_addons.query.**Node**(*parent*, *expr*, *path*)

A node in the expression tree.

Wraps around the SymPy expression node and contains path information. Makes only sense in the context of expression and path queries, where we have a containing expression.

exception sympy_addons.query.**NotFoundException**

exception sympy_addons.query.**NotUniqueException**

class sympy_addons.query.**Query**(***kwargs*)

A class for querying SymPy expression.

run(*expr*)

Run the query on an expression.

exception sympy_addons.query.**QueryException**

sympy_addons.query.**get_epath**(*subexpr*, *containing_expr*)

Get the unique epath for a subexpression within a given expression.

Parameters

- **subexpr** (*Basic*) – The subexpression to get epaths for.
- **containing_expr** (*Basic*) – The containing expression.

Returns

out – The epath string for the matching subexpression.

Return type

str

Raises

- *NotUniqueException* – if the subexpression is not unique in the containing expression.
- *NotFoundException* – if the subexpression is not found in the containing expression.

sympy_addons.query.**get_epaths**(*subexpr*, *containing_expr*)

Get all epaths for a subexpression within a given expression.

Parameters

- **subexpr** (*Basic*) – The subexpression to get epaths for.

- **containing_expr** (*Basic*) – The containing expression.

Returns
out – A list of epath-strings matching the subexpression.

Return type
list

sympy_addons.query.**make_expression_tree**(*expr*)

Build the expression tree with path information.

Parameters
expr (*Basic*) – The root expression for the expression tree to build.

Returns
out – The root node of the expression tree.

Return type
Node

sympy_addons.query.**walk_tree**(*root_node*)

Returns a list of all nodes in the expression tree.

2.2 Module *graphviz*

sympy_addons.graphviz.**plot_graph**(*expr*)

Make a graph plot of the internal representation of SymPy expression.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX