logical

Release 2.0.0

Reity LLC

Oct 10, 2022

CONTENTS

Callable subclass of the tuple type for representing logical operators/connectives based on their truth tables.

CHAPTER

ONE

INSTALLATION AND USAGE

This library is available as a package on PyPI:

```
python -m pip install logical
```

The library can be imported in the usual ways:

```
import logical
from logical import *
```

1.1 Examples

Each instance of the `logical` class (derived from the built-in `tuple` class) represents a boolean function that accepts `n` inputs by specifying its output values across all possible inputs. In other words, an instance represents the *output column* of a truth table for a function (under the assumption that the input vectors to which each output value corresponds are sorted in ascending order). Thus, each instance representing a function that accepts `n` inputs must have length `2**n`.

For example, consider the truth table below for a boolean function f that accepts three inputs:

x	y	z	$f(x, y, z)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Notice that the input vectors (*i.e.*, the left-most three column values in each row) are sorted in ascending order from top to bottom. If we always assume this order for input vectors, the entire function f can be represented using the right-most column. For the example function f defined by the table above, this can be done in the manner illustrated below:

```
>>> from logical import *
>>> f = logical((1, 0, 1, 0, 0, 1, 1, 0))
```

It is then possible to apply the instance `f` defined above to any three-component input vector:

```
>>> f(0, 1, 1)
0
>>> f(1, 1, 0)
1
```

It is also possible to create a new logical instance that has a function attribute corresponding to a compiled Python function that has the same behavior as the __call__ method (at least, on valid inputs). This Python function does not check that inputs are of the correct type and format, but has an execution time that is usually at most half of the execution time of the __call__ method:

```
>>> f = logical((1, 0, 0, 1, 0, 1, 0, 1))
>>> g = f.compiled()
>>> g.function(0, 0, 0)
1
>>> g.function(1, 1, 0)
0
```

Pre-defined instances are provided for all nullary, unary, and binary boolean functions. These are available both as constants and as attributes of the logical class:

```
>>> logical.xor_(1, 0)
1
>>> and_(1, 0)
0
```

The constants nullary, unary, and binary are also defined. Each is a set containing exactly those instances of logical that represent functions having that arity:

```
>>> unary
{(0, 0), (1, 0), (1, 1), (0, 1)}
>>> len(binary)
16
```

For convenience, the constant every is defined as the union of nullary, unary, and binary.

CHAPTER

TWO

DEVELOPMENT

All installation and development dependencies are fully specified in `pyproject.toml`. The `project.optional-dependencies` object is used to specify optional requirements for various development tasks. This makes it possible to specify additional options (such as `docs`, `lint`, and so on) when performing installation using pip:

```
python -m pip install .[docs,lint]
```

2.1 Documentation

The documentation can be generated automatically from the source files using Sphinx:

```
python -m pip install .[docs]
cd docs
sphinx-apidoc -f -E --templatedir=_templates -o _source .. && make html
```

2.2 Testing and Conventions

All unit tests are executed and their coverage is measured when using pytest (see the `pyproject.toml` file for configuration details):

```
python -m pip install .[test]
python -m pytest
```

Alternatively, all unit tests are included in the module itself and can be executed using doctest:

```
python src/logical/logical.py -v
```

Style conventions are enforced using Pylint:

```
python -m pip install .[lint]
python -m pylint src/logical
```

2.3 Contributions

In order to contribute to the source code, open an issue or submit a pull request on the GitHub page for this library.

2.4 Versioning

The version number format for this library and the changes to the library associated with version number increments conform with Semantic Versioning 2.0.0.

2.5 Publishing

This library can be published as a package on PyPI by a package maintainer. First, install the dependencies required for packaging and publishing:

```
python -m pip install .[publish]
```

Ensure that the correct version number appears in `pyproject.toml`, and that any links in this README document to the Read the Docs documentation of this package (or its dependencies) have appropriate version numbers. Also ensure that the Read the Docs project for this library has an automation rule that activates and sets as the default all tagged versions. Create and push a tag for this version (replacing `?.?.?` with the version number):

```
git tag ?.?.?
git push origin ?.?.?
```

Remove any old build/distribution files. Then, package the source into a distribution archive:

```
rm -rf build dist src/*.egg-info
python -m build --sdist --wheel .
```

Finally, upload the package distribution archive to PyPI:

```
python -m twine upload dist/*
```

2.5.1 logical module

Callable subclass of the built-in `tuple` type for representing logical operators and connectives based on their truth tables.

The two nullary, four unary, and sixteen binary operators are available as attributes of the *`logical`* class, and also as constants. Likewise, the four sets of operators *`logical.nullary`*, *`logical.unary`*, *`logical.binary`*, and *`logical.every`* are available both as attributes of *`logical`* and as exported top-level constants.

class `logical.logical.`**`logical`**(*iterable*)

Bases: `tuple`

Each instance of this class represents a boolean function of `n` inputs by specifying its output values across all possible inputs. In other words, an instance represents the *output column* of a truth table for a function (under the assumption that the input vectors to which each output value corresponds are sorted in ascending order). Each instance representing a function that accepts `n` inputs must have length `2 ** n`.

For example, consider the truth table below for a boolean function f that accepts two inputs:

x	*y*	*f* (*x*, *y*)
0	0	1
0	1	0
1	0	1
1	1	0

The entire function *f* can be represented using the right-most column. For the example function *f* defined by the table above, this can be done in the manner illustrated below.

```
>>> f = logical((1, 0, 1, 0))
>>> f(0, 1)
0
>>> f(1, 0)
1
```

Pre-defined instances are defined for all nullary, unary and binary functions, and are available as attributes of this class and as top-level constants:

- `()` = *`logical.undef_`* represents **UNDEFINED** (*i.e.*, no inputs and no defined output)
- `(0,)` = *`logical.nf_`* represents **NULLARY FALSE** (*i.e.*, no inputs and a constant output)
- `(1,)` = *`logical.nt_`* represents **NULLARY TRUE** (*i.e.*, no inputs and a constant output)
- `(0, 0)` = *`logical.uf_`* represents **UNARY FALSE** (*i.e.*, a constant output for any one input)
- `(0, 1)` = *`logical.id_`* represents **IDENTITY**
- `(1, 0)` = *`logical.not_`* represents **NOT**
- `(1, 1)` = *`logical.ut_`* represents **UNARY TRUE** (*i.e.*, a constant output for any one input)
- `(0, 0, 0, 0)` = *`logical.bf_`* represents **BINARY FALSE**
- `(0, 0, 0, 1)` = *`logical.and_`* represents **AND**
- `(0, 0, 1, 0)` = *`logical.nimp_`* represents **NIMP** (*i.e.*, >)
- `(0, 0, 1, 1)` = *`logical.fst_`* represents **FST** (*i.e.*, first/left-hand input)
- `(0, 1, 0, 0)` = *`logical.nif_`* represents **NIF** (*i.e.*, <)
- `(0, 1, 0, 1)` = *`logical.snd_`* represents **SND** (*i.e.*, second/right-hand input)
- `(0, 1, 1, 0)` = *`logical.xor_`* represents **XOR** (*i.e.*, !=)
- `(0, 1, 1, 1)` = *`logical.or_`* represents **OR**
- `(1, 0, 0, 0)` = *`logical.nor_`* represents **NOR**
- `(1, 0, 0, 1)` = *`logical.xnor_`* represents **XNOR** (*i.e.*, ==)
- `(1, 0, 1, 0)` = *`logical.nsnd_`* represents **NSND** (*i.e.*, negation of second input)
- `(1, 0, 1, 1)` = *`logical.if_`* represents **IF** (*i.e.*, >=)
- `(1, 1, 0, 0)` = *`logical.nfst_`* represents **NFST** (*i.e.*, negation of first input)
- `(1, 1, 0, 1)` = *`logical.imp_`* represents **IMP** (*i.e.*, <=)
- `(1, 1, 1, 0)` = *`logical.nand_`* represents **NAND**
- `(1, 1, 1, 1)` = *`logical.bt_`* represents **BINARY TRUE**

```
>>> logical.xor_(1, 0)
1
>>> and_(1, 0)
0
```

Because this class is derived from the `tuple` type, all methods and functions that operate on tuples also work with instances of this class.

```
>>> logical((1, 0)) == logical((1, 0))
True
>>> logical((1, 0)) == logical((0, 1))
False
>>> logical((1, 0))[1]
0
```

If an attempt is made to create an instance using an iterable that cannot be interpreted as a truth table, an exception is raised.

```
>>> logical(('a', 'b'))
Traceback (most recent call last):
  ...
TypeError: all entries in supplied truth table must be integers
>>> logical((-1, 2))
Traceback (most recent call last):
  ...
ValueError: all integers in supplied truth table must be 0 or 1
>>> logical((1, 0, 1))
Traceback (most recent call last):
  ...
ValueError: number of elements in supplied truth table must be zero or a power of 2
```

names: dict = {(): 'undef', (0,): 'nf', (0, 0): 'uf', (0, 0, 0, 0): 'bf', (0, 0, 0, 1): 'and', (0, 0, 1, 0): 'nimp', (0, 0, 1, 1): 'fst', (0, 1): 'id', (0, 1, 0, 0): 'nif', (0, 1, 0, 1): 'snd', (0, 1, 1, 0): 'xor', (0, 1, 1, 1): 'or', (1,): 'nt', (1, 0): 'not', (1, 0, 0, 0): 'nor', (1, 0, 0, 1): 'xnor', (1, 0, 1, 0): 'nsnd', (1, 0, 1, 1): 'if', (1, 1): 'ut', (1, 1, 0, 0): 'nfst', (1, 1, 0, 1): 'imp', (1, 1, 1, 0): 'nand', (1, 1, 1, 1): 'bt'}

Typical concise names for all nullary, unary, and binary operators.

nullary: frozenset = frozenset({(0,), (1,)})

Set of all nullary operators.

unary: frozenset = frozenset({(0, 0), (0, 1), (1, 0), (1, 1)})

Set of all unary operators.

binary: frozenset = frozenset({(0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 1, 0), (0, 0, 1, 1), (0, 1, 0, 0), (0, 1, 0, 1), (0, 1, 1, 0), (0, 1, 1, 1), (1, 0, 0, 0), (1, 0, 0, 1), (1, 0, 1, 0), (1, 0, 1, 1), (1, 1, 0, 0), (1, 1, 0, 1), (1, 1, 1, 0), (1, 1, 1, 1)})

Set of all binary operators.

every: frozenset = frozenset({(0,), (0, 0), (0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 1, 0), (0, 0, 1, 1), (0, 1), (0, 1, 0, 0), (0, 1, 0, 1), (0, 1, 1, 0), (0, 1, 1, 1), (1,), (1, 0), (1, 0, 0, 0), (1, 0, 0, 1), (1, 0, 1, 0), (1, 0, 1, 1), (1, 1), (1, 1, 0, 0), (1, 1, 0, 1), (1, 1, 1, 0), (1, 1, 1, 1)})

Set of all nullary, unary, and binary operators.

`__call__`(**arguments: Union[Tuple[int, ...], Tuple[Iterable[int]]]*) → int

Apply the function represented by this instance to zero or more integer arguments (where the arguments collectively represent an individual input row within a truth table) or to a single iterable of integers (where the entries of the iterable represent an individual input row within a truth table).

```
>>> logical((1,))()
1
>>> logical((1, 0))(1)
0
>>> logical((1, 0, 0, 1))(0, 0)
1
>>> logical((1, 0, 0, 1))(1, 1)
1
>>> logical((1, 0, 0, 1))(1, 0)
0
>>> logical((1, 0, 0, 1))(0, 1)
0
>>> logical((1, 0, 0, 1, 0, 1, 0, 1))(1, 1, 0)
0
>>> logical((1, 0, 0, 1, 0, 1, 0, 1))([1, 1, 0])
0
>>> logical((1, 0, 0, 1, 0, 1, 0, 1))((1, 1, 0))
0
>>> logical((1, 0, 0, 1, 0, 1, 0, 1))((1, 1, 0))
0
```

The supplied iterable of integers can be an iterator, as well.

```
>>> t = iter([1, 1, 0])
>>> logical((1, 0, 0, 1, 0, 1, 0, 1))(t)
0
```

The instance corresponding to the nullary function with no defined output raises an exception when applied to an input.

```
>>> logical(())()
Traceback (most recent call last):
  ...
ValueError: no defined output
```

Any attempt to apply an instance to an invalid input raises an exception.

```
>>> logical((1, 0))(2.3)
Traceback (most recent call last):
  ...
TypeError: expecting zero or more integers or a single iterable of integers
>>> logical((1, 0))(['abc'])
Traceback (most recent call last):
  ...
TypeError: expecting zero or more integers or a single iterable of integers
>>> logical((1, 0))(2)
Traceback (most recent call last):
  ...
ValueError: expecting an integer that is 0 or 1
```

name() → str

Return the typical concise name for this operator.

```
>>> logical((0,)).name()
'nf'
>>> logical((1, 0, 0, 1)).name()
'xnor'
>>> len([o.name for o in logical.nullary])
2
>>> len([o.name for o in logical.unary])
4
>>> len([o.name for o in logical.binary])
16
```

arity() → int

Return the arity of this operator.

```
>>> logical(()).arity()
0
>>> logical((1,)).arity()
0
>>> logical((1, 0)).arity()
1
>>> logical((1, 0, 0, 1)).arity()
2
```

compiled() → *logical.logical.logical*

Return a new instance (representing the same logical function) that has a new `function` attribute corresponding to a compiled version of the logical function it represents.

```
>>> (logical((1,)).compiled()).function()
1
>>> (logical((1, 0)).compiled()).function(1)
0
>>> f = logical((1, 0, 0, 1))
>>> g = f.compiled()
>>> g.function(1, 1)
1
>>> f = logical((1, 0, 0, 1, 0, 1, 0, 1))
>>> g = f.compiled()
>>> g.function(0, 0, 0)
1
>>> g.function(1, 1, 0)
0
```

The function is constructed by translating the truth table into an abstract syntax tree of a corresponding Python function definition (using the `ast` module), compiling that function definition (using the built-in `compile` function), executing that function definition (using `exec`), and then assigning that function to the `function` attribute.

While the compiled function increases the amount of memory consumed by an instance, the execution time of the compiled function on an input is usually at most half of the execution time of the `__call__` method.

undef_: *logical.logical.logical* = ()

Nullary operation with no defined output.

nf_: *logical.logical.logical* = (0,)

Nullary **FALSE** (constant) operation.

nf_()
0

nt_: *logical.logical.logical* = (1,)

Nullary **TRUE** (constant) operation.

nt_()
1

uf_: *logical.logical.logical* = (0, 0)

Unary **FALSE** (constant) operation.

x	uf_(x)
0	0
1	0

id_: *logical.logical.logical* = (0, 1)

Unary **IDENTITY** operation.

x	id_(x)
0	0
1	1

not_: *logical.logical.logical* = (1, 0)

Unary **NOT** operation (*i.e.*, negation).

x	not_(x)
0	1
1	0

ut_: *logical.logical.logical* = (1, 1)

Unary **TRUE** (constant) operation.

x	ut_(x)
0	1
1	1

bf_: *logical.logical.logical* = (0, 0, 0, 0)

Binary **FALSE** (constant) operation.

(x, y)	bf_(x, y)
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	0

and_: *logical.logical.logical* **= (0, 0, 0, 1)**
Binary **AND** operation (*i.e.*, conjunction).

(x, y)	and_(x, y)
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	1

nimp_: *logical.logical.logical* **= (0, 0, 1, 0)**
Binary **NIMP** operation (*i.e.*, >).

(x, y)	nimp_(x, y)
(0, 0)	0
(0, 1)	0
(1, 0)	1
(1, 1)	0

fst_: *logical.logical.logical* **= (0, 0, 1, 1)**
Binary **FST** operation (*i.e.*, first/left-hand input).

(x, y)	fst_(x, y)
(0, 0)	0
(0, 1)	0
(1, 0)	1
(1, 1)	1

nif_: *logical.logical.logical* **= (0, 1, 0, 0)**
Binary **NIF** operation (*i.e.*, <).

(x, y)	nif_(x, y)
(0, 0)	0
(0, 1)	1
(1, 0)	0
(1, 1)	0

snd_: *logical.logical.logical* **= (0, 1, 0, 1)**
Binary **SND** operation (*i.e.*, second/right-hand input).

(x, y)	snd_(x, y)
(0, 0)	0
(0, 1)	1
(1, 0)	0
(1, 1)	1

xor_: *logical.logical.logical* **= (0, 1, 1, 0)**
Binary **XOR** operation (*i.e.*, !=).

(x, y)	xor_(x, y)
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0

or_: *logical.logical.logical* **= (0, 1, 1, 1)**

Binary **OR** operation (*i.e.*, disjunction).

(x, y)	or_(x, y)
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	1

nor_: *logical.logical.logical* **= (1, 0, 0, 0)**

Binary **NOR** operation.

(x, y)	nor_(x, y)
(0, 0)	1
(0, 1)	0
(1, 0)	0
(1, 1)	0

xnor_: *logical.logical.logical* **= (1, 0, 0, 1)**

Binary **XNOR** operation (*i.e.*, ==).

(x, y)	xnor_(x, y)
(0, 0)	1
(0, 1)	0
(1, 0)	0
(1, 1)	1

nsnd_: *logical.logical.logical* **= (1, 0, 1, 0)**

Binary **NSND** operation (*i.e.*, negation of second/right-hand input).

(x, y)	nsnd_(x, y)
(0, 0)	1
(0, 1)	0
(1, 0)	1
(1, 1)	0

if_: *logical.logical.logical* **= (1, 0, 1, 1)**

Binary **IF** operation.

(x, y)	if_(x, y)
(0, 0)	1
(0, 1)	0
(1, 0)	1
(1, 1)	1

nfst_: *logical.logical.logical* **= (1, 1, 0, 0)**
Binary **NFST** operation (*i.e.*, negation of first/left-hand input).

(x, y)	nfst_(x, y)
(0, 0)	1
(0, 1)	1
(1, 0)	0
(1, 1)	0

imp_: *logical.logical.logical* **= (1, 1, 0, 1)**
Binary **IMP** operation (*i.e.*, implication or <=).

(x, y)	imp_(x, y)
(0, 0)	1
(0, 1)	1
(1, 0)	0
(1, 1)	1

nand_: *logical.logical.logical* **= (1, 1, 1, 0)**
Binary **NAND** operation (*i.e.*, negation of conjunction).

(x, y)	nand_(x, y)
(0, 0)	1
(0, 1)	1
(1, 0)	1
(1, 1)	0

bt_: *logical.logical.logical* **= (1, 1, 1, 1)**
Binary **TRUE** (constant) operation.

(x, y)	bt_(x, y)
(0, 0)	1
(0, 1)	1
(1, 0)	1
(1, 1)	1

PYTHON MODULE INDEX

l

INDEX