tt

Release 0.4.1

December 20, 2016

Contents

Welcome to the documentation site for tt.

Warning: tt is heavily tested and fully usable, but is still pre-1.0/stable software with **no guarantees** of avoiding breaking API changes until hitting version 1.0.

CHAPTER 1

Synopsis

tt is a Python library and command-line tool for working with Boolean expressions. Please check out the project site for more information.

CHAPTER 2

Installation

tt has been tested with Python 2.7, 3.3, 3.4, and 3.5 and is written in pure Python with no dependencies, so it only requires a compatible Python installation to run. You can get the latest release from PyPI with:

```
pip install ttable
```

CHAPTER 3

Basic Usage

Below are a couple of examples to show you the kind of things tt can do. For more examples and further documentation, take a look at the project site.

3.1 As a Library

tt aims to provide a Pythonic interface for working with Boolean expressions. Here are some simple examples from the REPL:

```
>>> from tt import BooleanExpression, TruthTable
>>> b = BooleanExpression('A xor (B and 1)')
>>> b.tokens
['A', 'xor', '(', 'B', 'and', '1', ')']
>>> b.symbols
['A', 'B']
>>> print(b.tree)
xor
`----A
`----and
     `----B
     `----1
>>> b.evaluate(A=True, B=False)
True
>>> t = TruthTable(b)
>>> print(t)
+---+---+---+
| A | B |   |
+---+---+---+
| 0 | 0 | 0 |
+---+---+---+
| 0 | 1 | 1 |
+---+---+---+
| 1 | 0 | 1 |
+---+---+---+
| 1 | 1 | 0 |
+---+---+---+
>>> t = TruthTable('A or B', fill_all=False)
>>> print(t)
Empty!
>>> t.fill(A=0)
>>> print(t)
+---+---+---+
```

```
| A | B |   |
+---+---+---+
| 0 | 0 | 0 |
+---+---+---+
| 0 | 1 | 1 |
+---+---+---+
>>> t.fill(A=1)
>>> print(t)
+---+---+---+
| A | B |   |
+---+---+---+
| 0 | 0 | 0 |
+---+---+---+
| 0 | 1 | 1 |
+---+---+---+
| 1 | 0 | 1 |
+---+---+---+
| 1 | 1 | 1 |
+---+---+---+
```

3.2 From the Command Line

tt also provides a command-line interface for working with expressions. Here are a couple of examples:

```
$ tt tokens "(op1 nand op2) xnor op3"
(
op1
nand
op2
)
xnor
op3

$ tt table A or B
+---+---+---+
| A | B |   |
+---+---+---+
| 0 | 0 | 0 |
+---+---+---+
| 0 | 1 | 1 |
+---+---+---+
| 1 | 0 | 1 |
+---+---+---+
| 1 | 1 | 1 |
+---+---+---+

$ tt tree A or or B
Error! Unexpected binary operator "or":
A or or B
     ^
```

CHAPTER 4

License

tt uses the MIT License.

4.1 Development

4.1.1 Managing with `ttasks.py`

tt ships with a script `ttasks.py` (tt + tasks = ttasks) in the project's top-level directory, used to manage common project tasks. You will see it referenced below.

4.1.2 Dependencies

All development requirements for tt are stored in the `dev-requirements.txt` file in the project's top-level directory. You can install all of these dependencies with:

```
pip install -r dev-requirements.txt
```

4.1.3 Testing

Testing is done with Python's unittest module. All tests can be run using the `ttasks.py` script:

```
python ttasks.py test
```

Cross Python version testing is achieved through tox, and the project's CI is run on Travis CI and AppVeyor. To run changes against the reference and style tests, simply invoke `tox` from the top-level directory of the project

4.1.4 Style

tt aims to be strictly PEP8 compliant, enforcing this compliance via Flake8. This project includes an editorconfig file to help with formatting issues, as well. Google style docstrings are used in the source code code documentation and processed via napoleon.

4.1.5 The Todo List

Below are features I'd like to add eventually, roughly ordered in anticipated schedule of completion. A new release will be cut every so often down the list.

- For the CLI
  - Functional testing, capturing stdout/stderr
  - Option for interfacing with the truth table's `fill` method
  - Option for interfacing with the truth table's `ordering` attribute
  - Option for specifying output delimiters for token-listing commands
- For the project as a whole
  - A *Getting Started* section with a tutorial-style guide to the library and CLI
  - Clean up API documentation (with valid cross-references)
  - Karnaugh map support
  - Optimizations in tree evaluation
  - Interface for substituting/transforming expression symbols
  - Functionality for optimizing/simplifying expressions

4.2 Prior Art

There are some great projects operating in the same problem space as tt. Most of the listed libraries are more mature and feature-rich than tt, so they may be a better choice for the problems you're working on. If you think that your library should be listed here, please let me know or submit a PR.

4.2.1 Python libraries

- boolean.py
- PyEDA

4.2.2 Other languages

- LogicNG (Java)
- BoolExpr (C++)
- EvalEx (Java)

4.3 Special Thanks

A lot of free services and open source libraries have helped this project become possible. This page aims to give credit where its due; if you were left out, I'm sorry! Please let me know!

4.3.1 Services

Thank you to the free hosting provided by these services!

- Github
- Travis CI

- AppVeyor
- Read the Docs

4.3.2 Open Source Projects & Libraries

tt relies on some well-written and well-documented projects and libraries for its development, listed below. Thank you!

- Alabaster
- Babel
- Colorama
- Docutils
- Flake8
- imagesize
- Jinja2
- MarkupSafe
- McCabe
- pep8
- pluggy
- py
- pyflakes
- Pygments
- Python
- pytz
- Requests
- six
- snowballstemmer
- Sphinx
- Sphinx napoleon extension
- tox
- virtualenv

4.4 Author

tt is written by Brian Welch. If you'd like to discuss anything about this library, Python, or software engineering in general, please feel free to reach out via one of the below channels.

- Personal website
- Github

CHAPTER 5

API Docs

Feel free to peruse through the source, or take a look through the auto-generated api docs below.

5.1 cli

tt's command-line interface.

5.1.1 cli.core module

Core command-line interface for tt.

tt.cli.core.**get_parsed_args**(*args=None*)
Get the parsed command line arguments.

Parameters **args** (*List[str], optional*) – The list of command line args to parse; if left as None, sys.argv will be used.

Returns The Namespace object returned by the `ArgumentParser.parse_args` function.

Return type argparse.Namespace

tt.cli.core.**main**(*args=None*)
The main routine to run the tt command-line interface.

Parameters **args** (*List[str], optional*) – The `args` argument to be passed to the `get_parsed_args` function.

Returns The exit code of the program.

Return type int

5.1.2 cli.utils module

Utilities for the tt command-line interface.

tt.cli.utils.**print_err**(**args*, ***kwargs*)
A thin wrapper around `print` to print to stderr.

tt.cli.utils.**print_info**(**args*, ***kwargs*)
A thin wrapper around `print`.

5.2 definitions

Definitions for tt's expression grammar, operands, and operators.

5.2.1 definitions.grammar module

Definitions related to expression grammar.

5.2.2 definitions.operands module

Definitions related to operands.

5.2.3 definitions.operators module

Definitions for tt's built-in Boolean operators.

class tt.definitions.operators.**BooleanOperator**(*precedence*, *eval_func*, *name*)

Bases: object

A wrapper around a Boolean operator.

precedence

int

The precendence of this operator, relative to other operators.

eval_func

function

The function representing the operation to be performed by this operator.

5.3 errors

tt error types.

5.3.1 errors.base module

The base tt exception type.

exception tt.errors.base.**TtError**(*message*, **args*)

Bases: Exception

Base exception type for tt errors.

message

str

An additional helpful message that could be displayed to the user to better explain the error.

> **Warning:** This exception type should be sub-classed and is not meant to be raised explicitly.

5.3.2 errors.evaluation module

Exception type definitions related to expression evaluation.

exception tt.errors.evaluation.**DuplicateSymbolError**(*message*, **args*)
Bases: *tt.errors.evaluation.EvaluationError*

An exception type for user-specified duplicate symbols.

exception tt.errors.evaluation.**EvaluationError**(*message*, **args*)
Bases: *tt.errors.base.TtError*

An exception type for errors occurring in expression evaluation.

> **Warning:** This exception type should be sub-classed and is not meant to be raised explicitly.

exception tt.errors.evaluation.**ExtraSymbolError**(*message*, **args*)
Bases: *tt.errors.evaluation.EvaluationError*

An exception for a passed token that is not a parsed symbol.

exception tt.errors.evaluation.**InvalidBooleanValueError**(*message*, **args*)
Bases: *tt.errors.evaluation.EvaluationError*

An exception for an invalid truth value passed in evaluation.

exception tt.errors.evaluation.**MissingSymbolError**(*message*, **args*)
Bases: *tt.errors.evaluation.EvaluationError*

An exception type for a missing token value in evaluation.

exception tt.errors.evaluation.**NoEvaluationVariationError**(*message*, **args*)
Bases: *tt.errors.evaluation.EvaluationError*

An exception type for when evaluation of an expression will not vary.

5.3.3 errors.generic module

Generic exception types.

exception tt.errors.generic.**InvalidArgumentTypeError**(*message*, **args*)
Bases: *tt.errors.base.TtError*

An exception type for invalid argument types.

5.3.4 errors.grammar module

Exception type definitions related to expression grammar and parsing.

exception tt.errors.grammar.**BadParenPositionError**(*message*, *expr_str=None*, *error_pos=None*, **args*)
Bases: *tt.errors.grammar.GrammarError*

An exception type for unexpected parentheses.

exception tt.errors.grammar.**EmptyExpressionError**(*message*, *expr_str=None*, *error_pos=None*, **args*)
Bases: *tt.errors.grammar.GrammarError*

An exception type for when an empty expression is received.

exception tt.errors.grammar.**ExpressionOrderError**(*message*, *expr_str=None*, *error_pos=None*, **args*)

Bases: *tt.errors.grammar.GrammarError*

An exception type for unexpected operands or operators.

exception tt.errors.grammar.**GrammarError**(*message*, *expr_str=None*, *error_pos=None*, **args*)

Bases: *tt.errors.base.TtError*

Base type for errors that occur in the handling of expression.

message
str

An additional helpful message that could be displayed to the user to better explain the error.

expr_str
str, optional

The expression in which the grammar error occured; can be left as `None` if the error position will not be specified.

error_pos
int, optional

The index indicating at which position int `expr_str` the troublesome spot began; can be left as `None` for errors without a specific troublesome position.

> **Warning:** This exception type should be sub-classed and is not meant to be raised explicitly.

exception tt.errors.grammar.**UnbalancedParenError**(*message*, *expr_str=None*, *error_pos=None*, **args*)

Bases: *tt.errors.grammar.GrammarError*

An exception type for unbalanced parentheses.

5.4 expressions

Tools for working with Boolean expressions.

5.4.1 expressions.bexpr module

Tools for interacting with Boolean expressions.

class tt.expressions.bexpr.**BooleanExpression**(*raw_expr*)

Bases: `object`

A class for parsing and holding information about a Boolean expression.

raw_expr
str

The raw string expression, to be parsed upon initialization.

symbols
List[str]

The list of unique symbols present in this expression, in the order of their first appearance in the expression.

tokens
List[str]

A list of strings, each element indicating a different token of the parsed expression.

postfix_tokens
List[str]

A list of strings, representing the `tokens` list converted to postfix form.

tree
tt.trees.BooleanExpressionTree

The expression tree representing the expression wrapped in this class, derived from the tokens parsed by this class.

evaluate(***kwargs*)
Evaluate the Boolean expression for the passed keyword arguments.

This is a checked wrapper around the `evaluate_unchecked` function.

Parameters **kwargs** – Keys are names of symbols in this expression; the specified value for each of these keys will be substituted into the expression for evaluation.

Returns The Boolean result of evaluating the expression.

Return type bool

Raises

- `ExtraSymbolError`
- `InvalidBooleanValueError`
- `MissingSymbolError`

Note: See `tt.utils.assertions.assert_all_valid_keys` and `tt.utils.assertions.assert_iterable_contains_all_expr_symbols` for more information about the exceptions raised by this method.

---

evaluate_unchecked(***kwargs*)
Evaluate the Boolean expression without checking the input.

Parameters **kwargs** – Keys are names of symbols in this expression; the specified value for each of these keys will be substituted into the expression for evaluation.

Returns The Boolean result of evaluating the expression.

Return type bool

5.5 tables

Tools for working with truth tables.

5.5.1 tables.truth_table module

Implementation of a truth table.

class tt.tables.truth_table.**TruthTable**(*expr*, *fill_all=True*, *ordering=None*)
Bases: object

A class representing a truth table.

expr
The BooleanExpression object or a string from which to create the BooleanExpression object represented by this truth table.

ordering
List[str], optional

An optional list of symbol names in the passed expression, specifying the order in which they should appear in the truth table (from left to right). If omitted, the ordering of the symbols will be consistent with the symbol's first occurrence in the passed expression.

results
List[bool]

A list of int s representing the resultant evaluations for each combination of possible inputs.

Raises

- DuplicateSymbolError
- ExtraSymbolError
- MissingSymbolError

fill(***kwargs*)
Fill the table with results, based on values specified by kwargs.

Parameters ****kwargs** – Filter which entries in the table are filled by specifying symbol values through the keyword args.

Raises

- ExtraSymbolError
- InvalidBooleanValueError
- MissingSymbolError

5.6 trees

Tools for working with Boolean expression trees.

5.6.1 tt.expr_tree module

An expression tree implementation for Boolean expressions.

class tt.trees.expr_tree.**BooleanExpressionTree**(*postfix_tokens*)
Bases: object

An expression tree for Boolean expressions.

Note: This class expects any input it receives to be well-formed; any tokenized lists you pass it directly (instead of through the BooleanExpression class) will not be checked.

postfix_tokens
List[str]

The tokens in the expression from which to build the tree.

root
tt.trees.ExpressionTreeNode

The root of the tree; `None` for an empty tree.

evaluate(*input_dict*)
Evaluate the expression held in this tree for specified inputs.

Parameters **input_dict** (*Dict*) – A dict mapping string variable names to the values for which they should be evaluated.

Returns The truthy result of the expression tree evaluation.

Note:

This function does not check to ensure the validity of `input_dict` in any way.

5.6.2 tt.tree_node module

A node, and related classes, for use in expression trees.

class tt.trees.tree_node.**BinaryOperatorExpressionTreeNode**(*operator_str*, *l_child*, *r_child*)
Bases: *tt.trees.tree_node.ExpressionTreeNode*

An expression tree node for binary operators.

operator
tt.operators.BooleanOperator

The actual operator wrapped in this node.

class tt.trees.tree_node.**ExpressionTreeNode**(*symbol_name*, *l_child=None*, *r_child=None*)
Bases: `object`

A base class for expression tree nodes.

symbol_name
str

The string operator/operand name of wrapped in this node.

l_child
tt.trees.ExpressionTreeNode, optional

This node's left child.

r_child
tt.trees.ExpressionTreeNode, optional

This node's right child.

evaluate(*input_dict*)
Recursively evaluate this node.

Parameters **input_dict** (*Dict*) – A dictionary mapping expression symbols to the value for which they should be subsituted in expression evaluation.

Note: Node evaluation does no checking of the validity of inputs; they should be check before being passed here.

Returns A truthy value.

class tt.trees.tree_node.**OperandExpressionTreeNode**(*operand_str*)
Bases: *tt.trees.tree_node.ExpressionTreeNode*

An expression tree node for operands.

class tt.trees.tree_node.**UnaryOperatorExpressionTreeNode**(*operator_str*, *l_child*)
Bases: *tt.trees.tree_node.ExpressionTreeNode*

An expression tree node for unary operators.

5.7 utils

Utilities for use under the hood.

5.7.1 utils.assertions module

Utilities for asserting inputs and states.

tt.utils.assertions.**assert_all_valid_keys**(*symbol_input_dict*, *symbol_set*)
Assert that all keys in the passed input dict are valid.

Valid keys are considered those that are present in the passed set of symbols and that map to valid Boolean values. Dictionaries cannot have duplicate keys, so no duplicate checking is necessary.

Parameters

- **symbol_input_dict** (*Dict*) – A dict containing symbol names mapping to what should be Boolean values.
- **symbol_set** (*Set[str]*) – A set of the symbol names expected to

Raises

- ExtraSymbolError – If any keys in the passed input dict are not present in the passed set of symbols.
- InvalidBooleanValueError – If any values in the passed input dict are not valid Boolean values (1, 0, True, or False).

tt.utils.assertions.**assert_iterable_contains_all_expr_symbols**(*iterable*, *reference_set*)
Assert a one-to-one presence of all symbols in the passed iterable.

Parameters

- **iterable** – An iterable of strings to assert.
- **reference_set** (*Set[str]*) – A set of strings, each of which will be asserted to be present in the passed iterable.

Note: This function will consume the passed iterable.

Raises

- `DuplicateSymbolError` – If the passed iterable contains more than one of the same symbol.
- `ExtraSymbolError` – If the passed iterable contains symbols not present in the reference set.
- `MissingSymbolError` – If the passed iterable is missing symbols present in the reference set.

t

S

T

U