gavel

Release 0.0.0

Dec 02, 2019

Contents

CHAPTER 1

Overview

docs	
tests	
package	

A toolset for prover independent premise selection. Template generated with cookiecutter-pylibrary.

1.1 Installation

```
pip install gavel
```

1.2 Documentation

https://gavel.readthedocs.io/

1.3 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 2

Installation

At the command line:

```
pip install gavel
```

CHAPTER 3

Usage

To use gavel in a project:

```
import gavel
```

CHAPTER 4

Reference

4.1 gavel

4.2 Logic

class gavel.logic.logic.**BinaryConnective**
An enumeration.

class gavel.logic.logic.**BinaryFormula**(*left: gavel.logic.logic.LogicElement*, *operator*, *right: gavel.logic.logic.LogicElement*)

Variables

- **oparator** – A binary operator
- **left** – The formula on the left side
- **right** – The formula on the right side

class gavel.logic.logic.**DefinedConstant**
An enumeration.

class gavel.logic.logic.**DefinedPredicate**
An enumeration.

class gavel.logic.logic.**QuantifiedFormula**(*quantifier*, *variables*, *formula*)

Variables

- **quantifier** – A quantier (existential or universal)
- **variables** – A list of variables bound by the quantifier
- **formula** – A logical formula

class gavel.logic.logic.**Quantifier**
An enumeration.

class gavel.logic.logic.**UnaryConnective**
An enumeration.

class gavel.logic.logic.**UnaryFormula**(*connective*, *formula: gavel.logic.logic.LogicElement*)

Variables

- **connective** – A unary connective
- **formula** – A formula

4.3 Problem

class gavel.logic.problem.**FormulaRole**
An enumeration.

class gavel.logic.problem.**Problem**(*premises: Iterable[gavel.logic.problem.Sentence], conjecture: Iterable[gavel.logic.problem.Sentence], imports=None*)
This class stores the important information that are needed for a proof

Variables

- **premises** (gavel.logic.logic.Sentence) – The premises available for this problem
- **conjecture** (gavel.logic.logic.Sentence) – The conjecture that should be proven‘

4.4 Proof

class gavel.logic.proof.**Axiom**(*formula: gavel.logic.logic.LogicElement = None, name: str = None, **kwargs*)

class gavel.logic.proof.**Inference**(*antecedents: Iterable[gavel.logic.proof.ProofStep] = None, conclusion: gavel.logic.logic.LogicElement = None, **kwargs*)

class gavel.logic.proof.**Introduction**(*introduction_type: gavel.logic.proof.IntroductionType = None, **kwargs*)

class gavel.logic.proof.**IntroductionType**
An enumeration.

class gavel.logic.proof.**ProofStep**(*formula: gavel.logic.logic.LogicElement = None, name: str = None, **kwargs*)

4.5 Dialects

4.5.1 Parser

Gavel comes with a collection of parsers for different logical frameworks. You can use these parsers to parse a single expression from a string:

```
from gavel.dialects.tptp.parser import TPTPParser

parser = TPTPParser()
string = "cnf(name, axiom, a | b)."

structure = parser.parse_single_from_string(string)
print(structure.formula)
```

```
(a) | (b)
```

In most cases the exact structrue of the input is not known, e.g. it might contain several expressions. To extract all lines from a string use the *parse_single_from_string* which returns a generator of found expressions

```
string = "cnf(name, axiom, a | b).cnf(name, axiom, d | e)."

for line in parser.stream_formula_lines(string):
    structure = parser.parse_single_from_string(line)
    print(structure.formula)
```

```
(a) | (b)
(d) | (e)
```

If you want to parse a complete problem from a file, use the specific problem parser. As some reasoners (e.g. SPASS) do not accept problems that cotain multiple conjectures, `ProblemParser.parse` returns a generator of problems, containing one conjecture each.

```
from gavel.dialects.tptp.parser import TPTPProblemParser

problem_parser = TPTPProblemParser()
string = "cnf(a1, axiom, a | b).cnf(a1, axiom, ~a).cnf(a2, negated_conjecture, b)."

for problem in problem_parser.parse(string):
    print("Axioms:")
    for a in problem.premises:
        print(a.formula)
    print("Conjecture:")
    print(problem.conjecture.formula)
```

```
Axioms:
(a) | (b)
~(a)
Conjecture:
b
```

class gavel.dialects.base.parser.**LogicParser**

class gavel.dialects.base.parser.**Parser**

parse(*structure: Parseable*, **args*, ***kwargs*) → Target

Transforms the input structure into metadata as used by the OpenEnergyPlatform

Parameters **inp** (*str*) – The input string that should be parsed into OEP metadata

Returns *OEPMetadata* – OEP metadata represented by *inp*

exception gavel.dialects.base.parser.**ParserException**

class gavel.dialects.base.parser.**ProblemParser**(**args*, ***kwargs*)

logic_parser_cls
alias of *LogicParser*

parse(*inp*, **args*, ***kwargs*)
Transforms the input structure into metadata as used by the OpenEnergyPlatform

Parameters **inp** (*str*) – The input string that should be parsed into OEP metadata

Returns *OEPMetadata* – OEP metadata represented by *inp*

class gavel.dialects.base.parser.**ProofParser**

class gavel.dialects.base.parser.**StringBasedParser**

static _unpack_file(**args*, ***kwargs*) → Iterable[str]

is_valid(*inp: str*) → bool
Verify if *inp* is a sting representation that is parsable by this parser :Parameters: **inp** (*str*) – String to verify

Returns *bool* – Indicated whether this object is parsable or not

load_single_from_string(*string: str*, **args*, ***kwargs*) → Parseable
Load a string into the structure represented by the dialect

Parameters **string**

parse_single_from_string(*string: str*, *load_args=None*, *parse_args=None*, *load_kwargs=None*, *parse_kwargs=None*) → Target
Parse a string into OEPMetadata

Parameters **string**

4.5.2 Compiler

4.6 Provers

Gavel is designed to support a number of provers. The communication to those is handled by so called prover interfaces (see *BaseProverInterface*).

class gavel.prover.base.interface.**BaseProverInterface**(**args*, ***kwargs*)
Base class for prover support

prove(*problem: gavel.logic.problem.Problem*, **args*, ***kwargs*) → gavel.logic.proof.Proof
Takes a instance of a gavel proof problem. submits it to the prover in a supported format and parses the result into a gavel proof object. Both translations are handled by the prover dialect in `_prover_dialect_cls`

Parameters **problem** (*gavel.logic.problem.Problem*) – The problem to prove

Returns *A proof if the proof was successful*

4.6.1 Custom Provers

The following example defines a rudimentary parser that checks whether the conjecture is part of the premises - without any semantic insight.

```
def simple_prover(problem):
    for premise in problem.premises:
        if premise.formula == problem.conjecture.formula:
            # Create a proof structure
            p = Proof(steps=[ProofStep(formula=premise.formula)])
            return p
    return None
```

We can use this prover to prove really simple problems:

```
problem = prob.Problem(
    premises=[prob.AnnotatedFormula(logic="fof", name="axiom1", role=prob.FormulaRole.
↪AXIOM, formula=logic.DefinedConstant.VERUM)],
    conjecture=prob.AnnotatedFormula(logic="fof", name="conjecture", role=prob.
↪FormulaRole.CONJECTURE, formula=logic.DefinedConstant.VERUM)
)
proof = simple_prover(problem)
for step in proof.steps:
    print(step.formula)
```

```
$true
```

If you want to use your own prover in gavel, you need to implement in a prover interface. Simply implement a subclass of *BaseProverInterface*.

```
class YourProverInterface(BaseProverInterface):
    def _submit_problem(self, problem_instance, *args, **kwargs):
        # Call your prov:qer here
        result = simple_prover(problem_instance)
        return result
```

```
pi = YourProverInterface()
proof = pi.prove(problem)
for step in proof.steps:
    print(step.formula)
```

```
$true
```

Note that *simple_prover* is accepting and returning the structures used by gavel. If your parser requires a different format, you may want to implement a dialect and use it in `YourProverInterface._prover_dialect_cls`.

4.7 Using existing Provers

In order to use the TPTP-library, you have to "download it":http://www.tptp.org/TPTP/Distribution/TPTP-v7.3.0.tgz and set the *TPTP_ROOT* environment variable to the root folder (i.e. the folder containing the 'Axioms' and 'Problems' folders). This has to be done because TPTP uses relative imports in it's problem files

This is a list of all prover interface that ship with gavel:

- *EProver*
- *Hets*

If you want to use your own tools to alter the structure of the problem or solution, you may call the prover interfaces listed above on your own with a `Problem`-instance:

```
proof = prover.prove(problem)
```

4.8 Proving with EProver

Install EProver according to "it's documentation":https://github.com/eprover/eprover and set the following environment variables:

- *EPROVER*: Path to your EProver binary

Now you can run gavel with the following command:

eprover <problem_path>

4.8.1 Python interface

The EProver interface does not take any Parameters:

```
from gavel.prover.eprover.interface import EProverInterface
prover = EProverInterface()
```

class gavel.prover.eprover.interface.**EProverInterface**(**args*, ***kwargs*)

4.9 Proving with Hets

4.9.1 Online

In order to use the online version run *gavel* with the following environment variables:

- *HETS_HOST*: rest.hets.eu
- *HETS_PORT*: 80

following options parameters:

eprover –hets <problem_path>

4.9.2 Offline

Follow the instructions in the "hets documentation":https://github.com/spechub/Hets and start *hets –server* **or** run "the docker container":https://github.com/spechub/Hets/wiki/How-to-use-the-Hets-Docker-Container (the mount option is not necessary as gavel uses the rest interface instead of shared files).

- *HETS_HOST*: localhost
- *HETS_PORT*: The port your docker container is forwarding

following options parameters:

eprover –hets <problem_path>

4.9.3 Python interface

Hets is just a layer around a number of provers. The latest version of gavel only supports EProver. In order to use any Proverinterface with hets, you have to pass it to the constructor:

```
from gavel.prover.hets.interface import HetsEngine, HetsSession, HetsProve
from gavel.prover.eprover.interface import EProverInterface
internal_prover = EProverInterface()
# Or any  other subclass or anything that quacks like BaseProverInterface
hets_engine = HetsEngine()
hets_session = HetsSession(hets_engine)
prover = HetsProve(internal_prover, hets_session)
```

class gavel.prover.hets.interface.**HetsEngine**(*url=None*, *port=None*)

class gavel.prover.hets.interface.**HetsSession**(*engine*, **args*, ***kwargs*)

class gavel.prover.hets.interface.**HetsProve**(*prover_interface: gavel.prover.base.interface.BaseProverInterface*, *session: gavel.prover.hets.interface.HetsSession*, **args*, ***kwargs*)

4.10 Select Premises

As theorem proving is very computationally expensive, you may want to select premises that are beneficial to proof the given conjecture. Note that an exhaustive proof but unsuccessful attempt does not yield any information about the whole problem.

```
from gavel.selection.selector import Selector
selector = Selector()
small_problem = selector.select(problem)
prover.prove(small_problem)
```

class gavel.selection.selector.**Selector**

Base class for gavel selectors.

select(*problem: gavel.logic.problem.Problem*, ***kwargs*) → gavel.logic.problem.Problem

Takes a problem instance and returns a (smaller) problem instance :param problem: :return:

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

gavel could always use more documentation, whether as part of the official gavel docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/MGlauer/python-gavel/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *python-gavel* for local development:

1. Fork python-gavel (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/python-gavel.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Martin Glauer - http://theo.cs.ovgu.de/Staff/Martin+Glauer.html

CHAPTER 7

Changelog

7.1 0.0.0 (2019-02-19)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

Symbols

A

B

D

E

F

G

H

I

L

P

Q

S

U