oxmol

Release 0.1.0

May 30, 2020

Contents:

`oxmol` is a Python wrapper, written using PyO3, for the minimal molecule implemented in Rust by Rich Apodaca. This follows the 'minimal molecule API' outlined by Apodaca in a blog post.

This package is currently a work in progress, it is missing some of the following key pieces:

- A SMILES parser/writer (this is being worked on)
- Substructure matching
- Coordinate representations and embedding
- Descriptor generation

These will be expanded upon in future versions. At present, molecules can be instantiated and their 'minimal molecule' functionality works.

The project's GitHub repository can be found here. New contributors are welcome. Any bugs or significant frustrations can be reported in the issue tracker.

```
from oxmol import AtomSpec, BondSpec, Molecule

C, H, O = AtomSpec('C'), AtomSpec('H'), AtomSpec('O')
atoms = [C, O, H, H, H, H]

bond_indices = [(0, 1), (1, 2), (0, 3), (0, 4), (0, 5)]
bonds = [BondSpec(sid, tid, 1) for (sid, tid) in bond_indices]

mol = Molecule(atoms, bonds)
print(mol)
# PyMolecule with 6 atoms and 5 bonds.

for (sid, tid) in mol.edges:
    print(mol.element(sid), mol.element(tid))
    print(mol.bond_order(sid, tid))
# PyElement::C Element::O
# PyBondOrder::Single
# PyElement::O Element::H
# PyBondOrder::Single
# PyElement::C Element::H
# PyBondOrder::Single
# PyElement::C Element::H
# PyBondOrder::Single
# PyElement::C Element::H
# PyBondOrder::Single
```

CHAPTER 1

Installation

`oxmol` is currently not listed in any package repos while deployment is worked out, but installation from source is fairly easy. Installation has only been tested on Linux (Ubuntu 18.04) and with Python 3.7, but should be possible for any system compatible with PyO3. This includes Windows, macOS, Linux and BSD systems and both the Python and PyPy interpreters.

1.1 Toolchain

The easiest way to build the Python wheels is to use Maturin, which can be installed via pip.

```
pip install maturin
```

You will also need the nightly Rust compiler. If you have not yet installed the Rust toolchain, see the instructions here. The nightly compiler can then be installed and and enabled by running the following two commands:

```
rustup install nightly
rustup default nightly
```

To set the default compiler back to stable, run the following command:

```
rustup default stable
```

If compiling on macOS, you will need to create a a file at `~/.cargo/config` with the following contents in order to set the linker options:

```
[target.x86_64-apple-darwin]
rustflags = [
"-C", "link-arg=-undefined",
"-C", "link-arg=dynamic_lookup",
]
```

1.2 Dependencies

The Rust component of `oxmol` depends upon `pyo3`, `molecule` and `gamma` (a graph library) but these should be automatically fetched by Cargo when compiling so there are no dependencies to install manually.

1.3 Installation Steps

Clone the repo using git and move into the root folder:

```
git clone https://github.com/thesketh/oxmol
cd oxmol
```

Execute the following command from the root of the repo.

```
maturin build --release
```

Maturin will compile the Rust component of the library and create a Python wheel containing the Rust and Python components, which are placed in `./target/wheels`.

As Maturin will compile a wheel for each interpreter it finds (e.g. the system Python and the interpreter from a Conda environment), there may be multiple versions. You will have to install the version most relevant to your version of Python (e.g. `oxmol-0.1.0-cp36...` for Python 3.6 and `oxmol-0.1.0-cp37...` for Python 3.7). The full name of the file will vary depending on your platform.

cd into `./target/wheels` and use pip to install the most relevant wheel

```
cd ./target/wheels
# Update the following command with the most relevant version.
pip install ./oxmol-0.1.0-cp37-cp37m-manylinux1_x86_64.whl
```

You should now be ready to start using `oxmol`. If you are interested in compiling against a specific version of Python, you can specify the path to the interetreter using the `-i` flag for Maturin:

```
maturin build -i $(which python3) --release
```

For information about other flags, use the Maturin docs.

1.4 Installing against PyPy

The latest version of PyPy changed the ABI string format and Maturin hasn't yet been updated, so you may have to rename the wheel file.

To compile oxmol against PyPy3, use the `-i` flag for Maturin:

```
maturin build -i $(which pypy3) --release
cd target/wheels
```

If you're using PyPy3 version >= 7.3.1 (where `SYSTEM` is e.g. `linux_x86_64`):

```
mv oxmol-0.1.0-pp3pp73-pypy3_pp73-SYSTEM.whl oxmol-0.1.0-pp36-pypy36_pp73-SYSTEM.whl
```

Install using PyPy's pip:

```
pypy3 -m pip install oxmol-0.1.0-pp36-pypy36_pp73-SYSTEM.whl
```

CHAPTER 2

API Reference

`oxmol` is a wrapper around `molcule.rs`, a Rust cheminformatics library written by Richard Apodaca. `oxmol` is implemented in Rust (using PyO3) and Python, and is currently in the alpha stage.

`molecule.rs` is based on a minimal molecule API set out by Apodaca in the following blog posts:

- Initial minimial molecule discussion
- Rust implementation of the minimal molecule

Future development will likely be discussed on the blog. At present, there is a useful molecule representation, but some critical components are currently missing:

- A SMILES parser/writer (this is being worked on)
- Substructure matching
- Coordinate representations and embedding
- Descriptor generation.

These will be expanded upon in future versions.

2.1 Submodules

2.1.1 oxmol.bond_order module

Representation of bond order.

This is a thin wrapper around the PyO3 base class: it calls the base class' `__new__` constructor and return an instance of the base class.

At present, it doesn't seem to be possible to truly subclass PyO3 classes from Python, so this is all we are able to do on the Python end.

class oxmol.bond_order.**BondOrder**
Bases: `oxmol.oxmol.PyBondOrder`

A bond order. This is a Python class representing a Rust enum. The bond orders represented in `molecule.rs` are `Zero`, `Single`, `Double` and `Triple`.

Parameters **order** – an int in range(0, 4) representing the bond order as an `int`.

as_int() → int
Get the bond order as an integer in range(0, 4).

2.1.2 oxmol.element module

Representation of chemical element.

This is a thin wrapper around the PyO3 base class: it calls the base class' `__new__` constructor and return an instance of the base class.

At present, it doesn't seem to be possible to truly subclass PyO3 classes from Python, so this is all we are able to do on the Python end.

class oxmol.element.**Element**
Bases: `oxmol.oxmol.PyElement`

A chemical element. Represented in `molecule.rs` as a Rust enum.

Parameters **atomic_number** – an `int`, the atomic number.

classmethod from_symbol(*symbol: str*) → oxmol.oxmol.PyElement
Create an element from its atomic symbol.

2.1.3 oxmol.molecule module

Representation of a whole molecule.

This is a thin wrapper around the PyO3 base class: it calls the base class' `__new__` constructor and return an instance of the base class.

At present, it doesn't seem to be possible to truly subclass PyO3 classes from Python, so this is all we are able to do on the Python end.

class oxmol.molecule.**Molecule**
Bases: `oxmol.oxmol.PyDefaultMolecule`

A molecular representation. This class is ultimately a representation of `molecule::default_molecule::DefaultMolecule`.

Parameters

- **atoms** – a `list` of `PyAtomSpec`
- **bonds** – a `list` of `PyBondSpec`

Attributes

- `nodes` - a `list` of the atom indices
- `edges` - a `list` of `tuple` of two atom indices, representing the bonds

atom_parity(*atom_id: int*) → Optional[oxmol.oxmol.PyParity]
Given an atom ID, return the atom's tetrahedral chirality.

Parameters **atom_id** – the atom index

Returns the tetrahedral chirality of the atom.

bond_order(*sid: int*, *tid: int*) → oxmol.oxmol.PyBondOrder
Given a start atom ID and target atom ID, return the order of the bond between the two atoms.

Parameters

- **sid** – the atom index of the start of the bond
- **tid** – the atom index of the target of the bond

Returns the order of the bond between the atoms.

bond_parity(*sid: int*, *tid: int*) → Optional[oxmol.oxmol.PyParity]
Given a start atom ID and target atom ID, return the stereochemistry of the bond between the two atoms.

Parameters

- **sid** – the atom index of the start of the bond
- **tid** – the atom index of the target of the bond

Returns the stereochemistry of the bond between the atoms.

charge(*atom_id: int*) → int
Given an atom ID, return the atom's formal charge.

Parameters **atom_id** – the atom index

Returns the atom's formal charge

degree(*atom_id: int*) → int
Given an atom ID, return the number of explicit connections the atom has.

Parameters **atom_id** – the atom index

Returns the number of connections that the atom has.

electrons(*atom_id: int*) → int
Given an atom ID, return the number of nonbonding electrons the atom has in its valence shell.

Parameters **atom_id** – the atom index

Returns the number of nonbonding electrons in the atom's valence shell

element(*atom_id: int*) → oxmol.oxmol.PyElement
Given an atom ID, return the element of that atom.

Parameters **atom_id** – the atom index

Returns the element of the atom

has_edge(*sid: int*, *tid: int*) → bool
Given a start atom ID and target atom ID, return a bool indicating whether a bond exists between the two atoms.

Parameters

- **sid** – the atom index of the start of the bond
- **tid** – the atom index of the target of the bond

Returns whether the bond exists in the molecule

has_node(*atom_id: int*) → bool
Given an atom ID, return a `bool` indicating whether the atom exists in the molecule.

Parameters **atom_id** – the atom index

Returns whether the atom exists in this molecule

hydrogens(*atom_id: int*) → int
Given an atom ID, return the number of virtual hydrogens the atom has.

Parameters **atom_id** – the atom index

Returns the number of virtual hydrogens on the atom

is_empty() → bool
Return whether the molecule contains atoms.

isotope(*atom_id: int*) → Optional[int]
Given an atom ID, return the isotope of that atom if it has been ascribed one. Otherwise, return `None`.

Parameters **atom_id** – the atom index

Returns the isotope, if it has been set, otherwise `None`

neighbors(*atom_id: int*) → List[int]
Given an atom ID, return the indices of the atom's neighbors.

Parameters **atom_id** – the atom index

Returns a `list` of indices of the atom's direct connections

order() → int
Return how many atoms are in the molecule.

size() → int
Return how many bonds are in the molecule.

2.1.4 oxmol.parity module

Representation of stereochemistry.

This is a thin wrapper around the PyO3 base class: it calls the base class' `__new__` constructor and return an instance of the base class.

At present, it doesn't seem to be possible to truly subclass PyO3 classes from Python, so this is all we are able to do on the Python end.

class oxmol.parity.**Parity**
Bases: `oxmol.oxmol.PyParity`

A stereochemical atom or bond parity, represented using a Rust enum. This parity is laid out in the minimal API description.

The enum refers to 'Positive' or 'Negative', which are represented using Python's `True` and `False` respectively.

For tetrahedral chirality: - `True` represents clockwise - `False` represents anticlockwise

For E/Z double bond stereochemistry: - `True` represents syn - `False` represents anti

Parameters **parity** – The atom or bond parity.

2.1.5 oxmol.spec module

Atom and bond specifications, used to construct `Molecule`

These are thin wrappers around the PyO3 classes: they call their base class' `__new__` constructor and return an instance of the base class.

At present, it doesn't seem to be possible to actually subclass PyO3 classes from Python, so this is all we are able to do on the Python end.

In the case of these classes, we tweak the constructors a bit to allow for some limited duck typing (this is harder to do in the PyO3 base classes).

class oxmol.spec.**AtomSpec**

Bases: oxmol.oxmol.PyAtomSpec

An atom specification, used to create a Molecule instance.

Parameters

- **element** – an Element, int (atomic number), or str (element symbol).
- **hydrogens** – an int, the number of implicit hydrogen atoms.
- **ion** – an optional int, the formal charge on the atom.
- **isotope** – an optional int, the isotope of the atom.
- **parity** – an optional Parity or bool, the chirality.

Attributes See Parameters.

class oxmol.spec.**BondSpec**

Bases: oxmol.oxmol.PyBondSpec

A bond specification, used to create a molecule instance.

Parameters

- **sid** – an int, the atom ID of the first atom in the bond.
- **tid** – an int, the atom ID of the last atom in the bond.
- **order** – a BondOrder or an int, the order of the bond.
- **parity** – an optional Parity or bool, the stereochemistry of the bond (only valid for double bonds).

Attributes See Parameters.

Python Module Index

o

Index