MolVS Documentation

Release 0.1.1

Matt Swain

Mar 24, 2019

Contents

MolVS is a molecule validation and standardization tool, written in Python using the RDKit chemistry framework (http://www.rdkit.org).

Building a collection of chemical structures from different sources can be difficult due to differing representations, drawing conventions and mistakes. MolVS can standardize chemical structures to improve data quality, help with de-duplication and identify relationships between molecules.

There are sensible defaults that make it easy to get started:

```
>>> from molvs import standardize_smiles
>>> standardize_smiles('[Na]OC(=O)c1ccc(C[S+2]([O-])([O-]))cc1')
'[Na+].O=C([O-])c1ccc(CS(=O)=O)cc1'
```

Each standardization module is also available separately, allowing the development of custom standardization processes.

CHAPTER 1

Features

- Normalization of functional groups to a consistent format.
- Recombination of separated charges.
- Breaking of bonds to metal atoms.
- Competitive reionization to ensure strongest acids ionize first in partially ionize molecules.
- Tautomer enumeration and canonicalization.
- Neutralization of charges.
- Standardization or removal of stereochemistry information.
- Filtering of salt and solvent fragments.
- Generation of fragment, isotope, charge, tautomer or stereochemistry insensitive parent structures.
- Validations to identify molecules with unusual and potentially troublesome characteristics.

CHAPTER 2

User guide

A step-by-step guide to getting started with MolVS.

2.1 Introduction

Building a collection of chemical structures from various different sources is difficult. There are differing file formats, molecular representations, drawing conventions, and things that are just plain wrong.

A lot of this arises due to our chemical models being an imperfect description of reality, but even within the idealized models there is often no single correct answer to whether two differently represented molecules are actually "the same". Whether tautomers or isomers of the same molecule should be considered equivalent or distinct entities can depend entirely on the specific application.

MolVS tries to address this problem through customizable validation and standardization processes, combined with the concept of "parent" molecule relationships to allow multiple simultaneous degrees of standardization.

This guide provides a quick tour through MolVS concepts and functionality.

2.1.1 MolVS license

MolVS is released under the MIT License. This is a short, permissive software license that allows commercial use, modifications, distribution, sublicensing and private use. Basically, you can do whatever you want with MolVS as long as you include the original copyright and license in any copies or derivative projects.

See the LICENSE file (https://github.com/mcs07/MolVS/blob/master/LICENSE) for the full text of the license.

2.2 Installation

MolVS supports Python versions 2.7 and 3.5+.

There are a variety of ways to download and install MolVS.

2.2.1 Option 1: Use conda (recommended)

The easiest and recommended way to install is using conda. Anaconda Python (https://www.continuum.io/anaconda-overview) is a self-contained Python environment that is particularly useful for scientific applications. If you don't already have it, start by installing Miniconda (http://conda.pydata.org/miniconda.html), which includes a complete Python distribution and the conda package manager. Choose the Python 3 version, unless you have a particular reason why you must use Python 2.

To install MolVS, at the command line, run:

```
conda config --add channels conda-forge
conda install molvs
```

This will add the conda-forge (https://conda-forge.org/) channel to your conda config, then install MolVS and all its dependencies into your conda environment.

2.2.2 Option 2: Use pip

An alternative method is to install using pip:

```
pip install molvs
```

This will download the latest version of MolVS, and place it in your *site-packages* folder so it is automatically available to all your python scripts.

Note: MolVS requires RDKit, which cannot be installed using pip. On the Mac, you can use Homebrew:

```
brew tap mcs07/cheminformatics
brew install rdkit
```

The official RDKit documentation has installation instructions for a variety of platforms (http://www.rdkit.org/docs/Install.html).

2.2.3 Option 3: Download the latest release

Alternatively, download the latest release (https://github.com/mcs07/MolVS/releases/) manually and install yourself:

```
tar -xzvf MolVS-0.1.1.tar.gz
cd MolVS-0.1.1
python setup.py install
```

The setup.py command will install MolVS in your *site-packages* folder so it is automatically available to all your python scripts.

2.2.4 Option 4: Clone the repository

The latest development version of MolVS is always available on GitHub (https://github.com/mcs07/MolVS). This version is not guaranteed to be stable, but may include new features that have not yet been released. Simply clone the repository and install as usual:

```
git clone https://github.com/mcs07/MolVS.git
cd MolVS
python setup.py install
```

2.3 Getting started

This page gives a introduction on how to get started with MolVS. This assumes you already have MolVS *installed* (page 6).

TODO...

2.4 Validation

The MolVS `Validator` provides a way to identify and log unusual and potentially troublesome characteristics of a molecule.

The validation process makes no actual changes to a molecule – that is left to the standardization process, which fixes many of the issues identified through validation. There is no real requirement to validate a molecule before or after standardizing it - the process simply provides additional information about potential problems.

2.4.1 Validating a molecule

The *`validate_smiles()`* (page 32) function is a convenient way to quickly validate a single SMILES string:

```
>>> from molvs import validate_smiles
>>> validate_smiles('O=C([O-])c1ccccc1')
['INFO: [NeutralValidation] Not an overall neutral system (-1)']
```

It returns a list of log messages as strings.

The `Validator` class provides more flexibility when working with multiple molecules or when a custom `Validation` list is required:

```
>>> fmt = '%(asctime)s - %(levelname)s - %(validation)s -
↪%(message)s'
>>> validator = Validator(log_format=fmt)
>>> mol = Chem.MolFromSmiles('[2H]C(Cl)(Cl)Cl')
>>> validator.validate(mol)
['2014-08-05 16:04:23,682 - INFO - IsotopeValidation - Molecule␣
↪contains isotope 2H']
```

2.4.2 Available validations

The *API documentation* (page 32) contains a full list of the individual validations that are available.

2.5 Standardization

This page gives details on the standardization process.

2.5.1 Standardizing a molecule

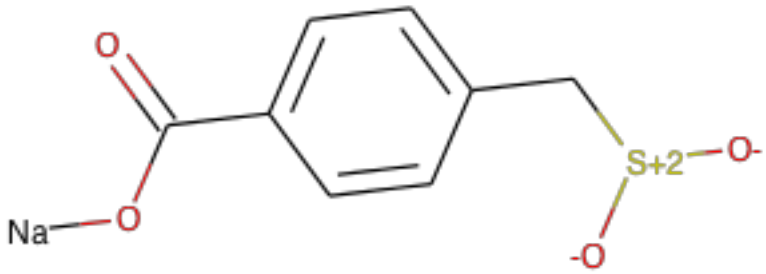
The `standardize_smiles` function provides a quick and easy way to get the standardized version of a given SMILES string:

```
>>> from molvs import standardize_smiles
>>> standardize_smiles('C[n+]1c([N-](C))cccc1')
'CN=c1ccccn1C'
```

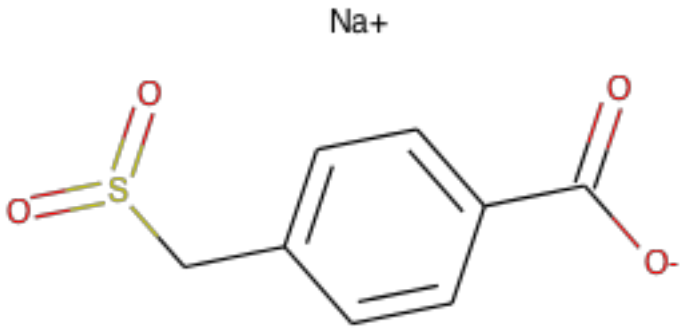
While this is convenient for one-off cases, it's inefficient when dealing with multiple molecules and doesn't allow any customization of the standardization process.

The `Standardizer` class provides flexibility to specify custom standardization stages and efficiently standardize multiple molecules:

```
>>> from rdkit import Chem
>>> mol = Chem.MolFromSmiles('[Na]OC(=O)c1ccc(C[S+2]([O-])([O-]))cc1
↪')
```




```
>>> from molvs import Standardizer
>>> s = Standardizer()
>>> smol = s.standardize(mol)
```



2.5.2 The standardization process

TODO: Explain this properly...

RDKit Sanitize

- Nitro N=O: `CN(=O)=O >> C[N+](=O)[O-]` and `C1=CC=CN(=O)=C1 >> C1=CC=C[N+]([O-])=C1`
- Nitro N#O: `C-N=N#N >> C-N=[N+]=[N-]`
- Perchlorate: `Cl(=O)(=O)(=O)[O-] >> [Cl+3]([O-])([O-])([O-])[O-]`
- Calculate explicit and implicit valence of all atoms. Fails when atoms have illegal valence.
- Calculate symmetrized SSSR. Slowest step, fails in rare cases.
- Kekulize. Fails if a Kekule form cannot be found or non-ring bonds are marked as aromatic.
- Assign radicals if hydrogens set and bonds+hydrogens+charge < valence.

- Set aromaticity, if none set in input. Go round rings, Huckel rule to set atoms+bonds as aromatic.
- Set conjugated property on bonds where applicable.
- Set hybridisation property on atoms.
- Remove chirality markers from sp and sp2 hybridised centers.

RDKit RemoveHs

- RDKit implementation detail - this is the preferred way to store the molecule.
- Remove explicit H count from atoms, instead infer it on the fly from valence model.

Disconnect metals

- Break covalent bonds between metals and organic atoms under certain conditions.
- First, disconnect N, O, F from any metal. Then disconnect other non-metals from transition metals (with exceptions).
- For every bond broken, adjust the charges of the begin and end atoms accordingly.
- In future, we might attempt to replace with zero-order bonds.

Apply normalization rules

- A series of transformations to correct common drawing errors and standardize functional groups. Includes:
- Uncharge-separate sulfones
- Charge-separate nitro groups
- Charge-separate pyridine oxide
- Charge-separate azide
- Charge-separate diazo and azo groups
- Charge-separate sulfoxides
- Hydrazine-diazonium system

Reionize acids

If molecule with multiple acid groups is partially ionized, ensure strongest acids ionize first.

The algorithm works as follows:

- Use SMARTS to find the strongest protonated acid and the weakest ionized acid.
- If the ionized acid is weaker than the protonated acid, swap proton and repeat.

Recalculate stereochemistry

- Use built-in RDKit functionality to force a clean recalculation of stereochemistry

2.6 Tautomers

This page gives details on tautomer enumeration and canonicalization.

2.6.1 Background

Tautomers are sets of molecules that readily interconvert with each other through the movement of a hydrogen atom. Tautomers have the same molecular formula and net charge, but they differ in terms of the positions of hydrogens and the associated changes in adjacent double and single bonds.

Because they rapidly interconvert, for many applications tautomers are considered to be the same chemical compound. And even in situations where it is important to treat tautomers as distinct compounds, it is still useful to be aware of the tautomerism relationships between molecules in a collection.

Varying tautomeric forms of the same molecule can have significantly different fingerprints and descriptors, which can negatively impact models for things like property prediction if they are used inconsistently.

There are two main tautomerism tasks that MolVS carries out:

- Tautomer enumeration: Finding the set of all the different possible tautomeric forms of a molecule.
- Tautomer canonicalization: Consistently picking one of the tautomers to be the canonical tautomer for the set.

2.6.2 Tautomer enumeration

- All possible tautomers are generated using a series of transform rules.
- Remove stereochemistry from double bonds that are single in at least 1 tautomer.

2.6.3 Tautomer canonicalization

- Enumerate all possible tautomers using transform rules.
- Use scoring system to determine canonical tautomer.
- Canonical tautomer should be “reasonable” from a chemist’s point of view, but isn’t guaranteed to be the most energetically favourable.

2.7 Fragments

This page gives details on dealing with fragments.

The term fragment refers to covalently bonded units. A molecule can contain multiple fragments.

2.7.1 Getting the largest fragment

- LargestFragmentChooser

2.7.2 Filtering out fragments

- FragmentRemover

2.8 Charges

This page gives details on dealing with charges in molecules.

2.8.1 Acid reionization

- Ensure the strongest acid groups ionize first in partially ionized molecules.

2.8.2 Neutralization

- Attempt to neutralize charges by adding and/or removing hydrogens where possible.
- Not always possible to produce a neutral molecule.

2.9 Command Line Tool

MolVS comes with a simple command line tool that allows standardization and validation by typing `molvs` at the command line.

2.9.1 Standardization

See standardization help by typing `molvs standardize -h`:

```
usage: molvs standardize [infile] [-i {smi,mol,sdf}] [-O <outfile>]
                         [-o {smi,mol,sdf}] [-: <smiles>]

positional arguments:
  infile                input filename

optional arguments:
  -i {smi,mol,sdf}, --intype {smi,mol,sdf}
                        input filetype
  -: <smiles>, --smiles <smiles>
                        input SMILES instead of file
  -O <outfile>, --outfile <outfile>
                        output filename
  -o {smi,mol,sdf}, --outtype {smi,mol,sdf}
                        output filetype
```

2.9.2 Validation

See validation help by typing `molvs validate -h`:

```
usage: molvs validate [infile] [-i {smi,mol,sdf}] [-O <outfile>]
                      [-: <smiles>]

positional arguments:
  infile                input filename

optional arguments:
  -i {smi,mol,sdf}, --intype {smi,mol,sdf}
                        input filetype
  -: <smiles>, --smiles <smiles>
                        input SMILES instead of file
  -O <outfile>, --outfile <outfile>
                        output filename
```

2.9.3 Examples

SMILES standardization:

```
$ molvs standardize -:"C[n+]1c([N-](C))cccc1"
CN=c1ccccn1C
```

Specifying an output format:

```
$ molvs standardize -:"[N](=O)(=O)O" -o mol

     RDKit
```

(continues on next page)

(continued from previous page)

```
  4  3  0  0  0  0  0  0  0  0999 V2000
    0.0000    0.0000    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
↪0
    0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
↪0
    0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
↪0
    0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
↪0
  1  2  1  0
  1  3  2  0
  1  4  1  0
M  CHG  2   1   1   2  -1
M  END
```

Using stdin:

```
$ echo "C[n+]1c([N-](C))cccc1" | molvs standardize
CN=c1ccccn1C
```

Specifying an input file:

```
$ molvs standardize example.mol
CN=c1ccccn1C
```

Specifying an output file:

```
$ molvs standardize example.mol -O output.smi
$ molvs standardize example.mol -O output.mol
$ molvs standardize example.mol -O output -o mol
```

Logging validations to stdout:

```
$ molvs validate -:"O=C([O-])c1ccccc1"
INFO: [NeutralValidation] Not an overall neutral system (-1)
```

Logging validations to a file:

```
$ molvs validate -:"O=C([O-])c1ccccc1" -O logs.txt
```

2.10 Contributing

Contributions of any kind are greatly appreciated!

2.10.1 Feedback

The Issue Tracker (https://github.com/mcs07/MolVS/issues) is the best place to post any feature ideas, requests and bug reports.

The following are especially welcome:

- General feedback on whether any standardization stages should work differently.
- Specific molecules that don't validate or standardize as expected.
- Ideas for new validation and standardization stages.

2.10.2 Contributing

If you are able to contribute changes yourself, just fork the source code (https://github.com/mcs07/MolVS) on GitHub, make changes and file a pull request. All contributions are welcome, no matter how big or small.

The following are especially welcome:

- New validation or standardization stages.
- Alternative tautomer transforms and scores.
- Lists of salts and solvents to strip out.
- New or improved documentation of existing features.

Quick guide to contributing

1. Fork the MolVS repository on GitHub (https://github.com/mcs07/MolVS/fork), then clone your fork to your local machine:

```
git clone https://github.com/<username>/MolVS.git
cd molvs
```

2. Install the development requirements into a conda environment (https://conda.io/docs/):

```
conda env create -n molvs -f environment.yml
source activate molvs
```

3. Create a new branch for your changes:

```
git checkout -b <name-for-changes>
```

4. Make your changes or additions. Ideally add some tests and ensure they pass by running:

```
pytest
```

5. Commit your changes and push to your fork on GitHub:

```
git add .
git commit -m "<description-of-changes>"
git push origin <name-for-changes>
```

4. Submit a pull request (https://github.com/mcs07/MolVS/compare/).

Tips

- Follow the PEP8 (https://www.python.org/dev/peps/pep-0008) style guide.
- Include docstrings as described in PEP257 (https://www.python.org/dev/peps/pep-0257).
- Try and include tests that cover your changes.
- Try to write good commit messages (http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html).
- Consider squashing your commits (http://gitready.com/advanced/2009/02/10/squashing-commits-with-rebase.html) with rebase.
- Read the GitHub help page on Using pull requests (https://help.github.com/articles/using-pull-requests).

CHAPTER 3

API documentation

Comprehensive API documentation with information on every function, class and method. This is automatically generated from the MolVS source code and comments.

3.1 API documentation

This part of the documentation is automatically generated from the MolVS source code and comments.

The MolVS package is made up of the following modules:

3.1.1 molvs.standardize

This module contains the main *Standardizer* (page 18) class that can be used to perform all standardization tasks, as well as convenience functions like *standardize_smiles()* (page 21) for common standardization tasks.

class molvs.standardize.**Standardizer**(*normalizations=NORMALIZATIONS*, *acid_base_pairs=ACID_BASE_PAIRS*, *tautomer_transforms=TAUTOMER_TRANSFORMS*, *tautomer_scores=TAUTOMER_SCORES*, *max_restarts=MAX_RESTARTS*, *max_tautomers=MAX_TAUTOMERS*, *prefer_organic=PREFER_ORGANIC*)

The main class for performing standardization of molecules and deriving parent molecules.

The primary usage is via the *standardize()* (page 19) method:

```
s = Standardizer()
mol1 = Chem.MolFromSmiles('C1=CC=CC=C1')
mol2 = s.standardize(mol1)
```

There are separate methods to derive fragment, charge, tautomer, isotope and stereo parent molecules.

Initialize a Standardizer with optional custom parameters.

Parameters

- **normalizations** – A list of Normalizations to apply (default: *NORMALIZATIONS* (page 22)).
- **acid_base_pairs** – A list of AcidBasePairs for competitive reionization (default: *ACID_BASE_PAIRS* (page 28)).
- **charge_corrections** – A list of ChargeCorrections to apply (default: CHARGE_CORRECTIONS).
- **tautomer_transforms** – A list of TautomerTransforms to apply (default: *TAUTOMER_TRANSFORMS* (page 24)).
- **tautomer_scores** – A list of TautomerScores used to determine canonical tautomer (default: *TAUTOMER_SCORES* (page 24)).
- **max_restarts** – The maximum number of times to attempt to apply the series of normalizations (default 200).
- **max_tautomers** – The maximum number of tautomers to enumerate (default 1000).
- **prefer_organic** – Whether to prioritize organic fragments when choosing fragment parent (default False).

__call__(*mol*)
Calling a Standardizer instance like a function is the same as calling its *standardize()* (page 19) method.

standardize(*mol*)
Return a standardized version the given molecule.

The standardization process consists of the following stages: RDKit RemoveHs() (http://www.rdkit.org/docs/source/rdkit.Chem.rdmolops.html#rdkit.Chem.rdmolops.RemoveHs), RDKit SanitizeMol() (http://www.rdkit.org/docs/source/rdkit.Chem.rdmolops.html#rdkit.Ch *MetalDisconnector* (page 23), *Normalizer* (page 22), *Reionizer* (page 28), RDKit AssignStereochemistry() (http://www.rdkit.org/docs/source/rdkit.Chem.rdmolops.html#rdkit.Chem.rdmolops.AssignStereo

Parameters **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The molecule to standardize.

Returns The standardized molecule.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

tautomer_parent(*mol*, *skip_standardize=False*)
Return the tautomer parent of a given molecule.

Parameters

- **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The input molecule.
- **skip_standardize** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Set to True if mol has already been standardized.

Returns The tautomer parent molecule.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

fragment_parent(*mol*, *skip_standardize=False*)
Return the fragment parent of a given molecule.

The fragment parent is the largest organic covalent unit in the molecule.

Parameters

- **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The input molecule.
- **skip_standardize** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Set to True if mol has already been standardized.

Returns The fragment parent molecule.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

stereo_parent(*mol*, *skip_standardize=False*)
Return the stereo parent of a given molecule.

The stereo parent has all stereochemistry information removed from tetrahedral centers and double bonds.

Parameters

- **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The input molecule.
- **skip_standardize** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Set to True if mol has already been standardized.

Returns The stereo parent molecule.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

isotope_parent(*mol*, *skip_standardize=False*)
Return the isotope parent of a given molecule.

The isotope parent has all atoms replaced with the most abundant isotope for that element.

Parameters

- **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The input molecule.
- **skip_standardize** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Set to True if mol has already been standardized.

Returns The isotope parent molecule.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

charge_parent(*mol*, *skip_standardize=False*)
Return the charge parent of a given molecule.

The charge parent is the uncharged version of the fragment parent.

Parameters

- **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The input molecule.
- **skip_standardize** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Set to True if mol has already been standardized.

Returns The charge parent molecule.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

super_parent(*mol*, *skip_standardize=False*)
Return the super parent of a given molecule.

THe super parent is fragment, charge, isotope, stereochemistry and tautomer insensitive. From the input molecule, the largest fragment is taken. This is uncharged and then isotope and stereochemistry information is discarded. Finally, the canonical tautomer is determined and returned.

Parameters

- **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The input molecule.
- **skip_standardize** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Set to True if mol has already been standardized.

Returns The super parent molecule.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

disconnect_metals

Returns A callable *MetalDisconnector* (page 23) instance.

normalize

Returns A callable *Normalizer* (page 22) instance.

reionize

Returns A callable *Reionizer* (page 28) instance.

uncharge

Returns A callable *Uncharger* (page 29) instance.

remove_fragments

Returns A callable *FragmentRemover* (page 26) instance.

largest_fragment

Returns A callable *LargestFragmentChooser* (page 27) instance.

enumerate_tautomers

Returns A callable *TautomerEnumerator* (page 25) instance.

canonicalize_tautomer

Returns A callable *TautomerCanonicalizer* (page 25) instance.

molvs.standardize.**standardize_smiles**(*smiles*)

Return a standardized canonical SMILES string given a SMILES string.

Note: This is a convenience function for quickly standardizing a single SMILES string. It is more efficient to use the *Standardizer* (page 18) class directly when working with many molecules or when custom options are needed.

Parameters **smiles** (*string*) – The SMILES for the molecule.

Returns The SMILES for the standardized molecule.

Return type string.

molvs.standardize.**enumerate_tautomers_smiles**(*smiles*)
Return a set of tautomers as SMILES strings, given a SMILES string.

Parameters **smiles** – A SMILES string.

Returns A set containing SMILES strings for every possible tautomer.

Return type set of strings.

molvs.standardize.**canonicalize_tautomer_smiles**(*smiles*)
Return a standardized canonical tautomer SMILES string given a SMILES string.

Note: This is a convenience function for quickly standardizing and finding the canonical tautomer for a single SMILES string. It is more efficient to use the *Standardizer* (page 18) class directly when working with many molecules or when custom options are needed.

Parameters **smiles** (*string*) – The SMILES for the molecule.

Returns The SMILES for the standardize canonical tautomer.

Return type string.

3.1.2 molvs.normalize

This module contains tools for normalizing molecules using reaction SMARTS patterns.

molvs.normalize.**NORMALIZATIONS**
The default list of Normalization transforms.

molvs.normalize.**MAX_RESTARTS = 200**
The default value for the maximum number of times to attempt to apply the series of normalizations.

class molvs.normalize.**Normalization**(*name*, *transform*)
A normalization transform defined by reaction SMARTS.

Parameters

- **name** (*string*) – A name for this Normalization
- **transform** (*string*) – Reaction SMARTS to define the transformation.

class molvs.normalize.**Normalizer**(*normalizations=NORMALIZATIONS*, *max_restarts=MAX_RESTARTS*)
A class for applying Normalization transforms.

This class is typically used to apply a series of Normalization transforms to correct functional groups and recombine charges. Each transform is repeatedly applied until no further changes occur.

Initialize a Normalizer with an optional custom list of *Normalization* (page 22) transforms.

Parameters

- **normalizations** – A list of *Normalization* (page 22) transforms to apply.
- **max_restarts** (*int* (https://docs.python.org/3/library/functions.html#int)) – The maximum number of times to attempt to apply the series of normalizations (default 200).

__call__(*mol*)

Calling a Normalizer instance like a function is the same as calling its normalize(mol) method.

normalize(*mol*)

Apply a series of Normalization transforms to correct functional groups and recombine charges.

A series of transforms are applied to the molecule. For each Normalization, the transform is applied repeatedly until no further changes occur. If any changes occurred, we go back and start from the first Normalization again, in case the changes mean an earlier transform is now applicable. The molecule is returned once the entire series of Normalizations cause no further changes or if max_restarts (default 200) is reached.

Parameters **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The molecule to normalize.

Returns The normalized fragment.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

3.1.3 molvs.metal

This module contains tools for disconnecting metal atoms that are defined as covalently bonded to non-metals.

class molvs.metal.**MetalDisconnector**

Class for breaking covalent bonds between metals and organic atoms under certain conditions.

__call__(*mol*)

Calling a MetalDisconnector instance like a function is the same as calling its disconnect(mol) method.

disconnect(*mol*)

Break covalent bonds between metals and organic atoms under certain conditions.

The algorithm works as follows:

- Disconnect N, O, F from any metal.
- Disconnect other non-metals from transition metals + Al (but not Hg, Ga, Ge, In, Sn, As, Tl, Pb, Bi, Po).

- For every bond broken, adjust the charges of the begin and end atoms accordingly.

Parameters **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The input molecule.

Returns The molecule with metals disconnected.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

3.1.4 molvs.tautomer

This module contains tools for enumerating tautomers and determining a canonical tautomer.

molvs.tautomer.**TAUTOMER_TRANSFORMS**
The default list of TautomerTransforms.

molvs.tautomer.**TAUTOMER_SCORES**
The default list of TautomerScores.

molvs.tautomer.**MAX_TAUTOMERS = 1000**
The default value for the maximum number of tautomers to enumerate, a limit to prevent combinatorial explosion.

class molvs.tautomer.**TautomerTransform**(*name*, *smarts*, *bonds=()*, *charges=()*, *radicals=()*)
Rules to transform one tautomer to another.

Each TautomerTransform is defined by a SMARTS pattern where the transform involves moving a hydrogen from the first atom in the pattern to the last atom in the pattern. By default, alternating single and double bonds along the pattern are swapped accordingly to account for the hydrogen movement. If necessary, the transform can instead define custom resulting bond orders and also resulting atom charges.

Initialize a TautomerTransform with a name, SMARTS pattern and optional bonds and charges.

The SMARTS pattern match is applied to a Kekule form of the molecule, so use explicit single and double bonds rather than aromatic.

Specify custom bonds as a string of `-`, `=`, `#`, `:` for single, double, triple and aromatic bonds respectively. Specify custom charges as `+`, `0`, `-` for +1, 0 and -1 charges respectively.

Parameters

- **name** (*string*) – A name for this TautomerTransform.
- **smarts** (*string*) – SMARTS pattern to match for the transform.
- **bonds** (*string*) – Optional specification for the resulting bonds.

- **charges** (*string*) – Optional specification for the resulting charges on the atoms.

class molvs.tautomer.**TautomerScore**(*name*, *smarts*, *score*)

A substructure defined by SMARTS and its score contribution to determine the canonical tautomer.

Initialize a TautomerScore with a name, SMARTS pattern and score.

Parameters

- **name** – A name for this TautomerScore.
- **smarts** – SMARTS pattern to match a substructure.
- **score** – The score to assign for this substructure.

class molvs.tautomer.**TautomerCanonicalizer**(*transforms=TAUTOMER_TRANSFORMS*, *scores=TAUTOMER_SCORES*, *max_tautomers=MAX_TAUTOMERS*)

Parameters

- **transforms** – A list of TautomerTransforms to use to enumerate tautomers.
- **scores** – A list of TautomerScores to use to choose the canonical tautomer.
- **max_tautomers** – The maximum number of tautomers to enumerate, a limit to prevent combinatorial explosion.

__call__(*mol*)

Calling a TautomerCanonicalizer instance like a function is the same as calling its canonicalize(mol) method.

canonicalize(*mol*)

Return a canonical tautomer by enumerating and scoring all possible tautomers.

Parameters **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The input molecule.

Returns The canonical tautomer.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

class molvs.tautomer.**TautomerEnumerator**(*transforms=TAUTOMER_TRANSFORMS*, *max_tautomers=MAX_TAUTOMERS*)

Parameters

- **transforms** – A list of TautomerTransforms to use to enumerate tautomers.
- **max_tautomers** – The maximum number of tautomers to enumerate (limit to prevent combinatorial explosion).

__call__(*mol*)
Calling a TautomerEnumerator instance like a function is the same as calling its enumerate(mol) method.

enumerate(*mol*)
Enumerate all possible tautomers and return them as a list.

Parameters **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The input molecule.

Returns A list of all possible tautomers of the molecule.

Return type list of rdkit.Chem.rdchem.Mol

3.1.5 molvs.fragment

This module contains tools for dealing with molecules with more than one covalently bonded unit. The main classes are *LargestFragmentChooser* (page 27), which returns the largest covalent unit in a molecule, and *FragmentRemover* (page 26), which filters out fragments from a molecule using SMARTS patterns.

molvs.fragment.**REMOVE_FRAGMENTS**
The default list of *FragmentPatterns* (page 26) to be used by *FragmentRemover* (page 26).

molvs.fragment.**LEAVE_LAST = True**
The default value for whether to ensure at least one fragment is left after FragmentRemover is applied.

molvs.fragment.**PREFER_ORGANIC = False**
The default value for whether LargestFragmentChooser sees organic fragments as "larger" than inorganic fragments.

class molvs.fragment.**FragmentPattern**(*name*, *smarts*)
A fragment defined by a SMARTS pattern.

Initialize a FragmentPattern with a name and a SMARTS pattern.

Parameters

- **name** – A name for this FragmentPattern.
- **smarts** – A SMARTS pattern.

molvs.fragment.**is_organic**(*fragment*)
Return true if fragment contains at least one carbon atom.

Parameters **fragment** – The fragment as an RDKit Mol object.

class molvs.fragment.**FragmentRemover**(*fragments=REMOVE_FRAGMENTS*, *leave_last=LEAVE_LAST*)
A class for filtering out fragments using SMARTS patterns.

Initialize a FragmentRemover with an optional custom list of *FragmentPattern* (page 26).

Setting leave_last to True will ensure at least one fragment is left in the molecule, even if it is matched by a *FragmentPattern* (page 26). Fragments are removed in the order specified in the list, so place those you would prefer to be left towards the end of the list. If all the remaining fragments match the same *FragmentPattern* (page 26), they will all be left.

Parameters

- **fragments** – A list of *FragmentPattern* (page 26) to remove.
- **leave_last** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Whether to ensure at least one fragment is left.

__call__(*mol*)

Calling a FragmentRemover instance like a function is the same as calling its remove(mol) method.

remove(*mol*)

Return the molecule with specified fragments removed.

Parameters **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The molecule to remove fragments from.

Returns The molecule with fragments removed.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

class molvs.fragment.**LargestFragmentChooser**(*prefer_organic=PREFER_ORGANIC*)

A class for selecting the largest covalent unit in a molecule with multiple fragments.

If prefer_organic is set to True, any organic fragment will be considered larger than any inorganic fragment. A fragment is considered organic if it contains a carbon atom.

Parameters **prefer_organic** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Whether to prioritize organic fragments above all others.

__call__(*mol*)

Calling a LargestFragmentChooser instance like a function is the same as calling its choose(mol) method.

choose(*mol*)

Return the largest covalent unit.

The largest fragment is determined by number of atoms (including hydrogens). Ties are broken by taking the fragment with the higher molecular weight, and then by taking the first alphabetically by SMILES if needed.

Parameters **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The molecule to choose the largest fragment from.

Returns The largest fragment.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

3.1.6 molvs.charge

This module implements tools for manipulating charges on molecules. In particular, *Reionizer* (page 28), which competitively reionizes acids such that the strongest acids ionize first, and *Uncharger* (page 29), which attempts to neutralize ionized acids and bases on a molecule.

molvs.charge.**ACID_BASE_PAIRS**
The default list of AcidBasePairs, sorted from strongest to weakest. This list is derived from the Food and Drug Administration Substance Registration System Standard Operating Procedure guide.

class molvs.charge.**AcidBasePair**(*name*, *acid*, *base*)
An acid and its conjugate base, defined by SMARTS.

A strength-ordered list of AcidBasePairs can be used to ensure the strongest acids in a molecule ionize first.

Initialize an AcidBasePair with the following parameters:

Parameters

- **name** (*string*) – A name for this AcidBasePair.
- **acid** (*string*) – SMARTS pattern for the protonated acid.
- **base** (*string*) – SMARTS pattern for the conjugate ionized base.

class molvs.charge.**Reionizer**(*acid_base_pairs=ACID_BASE_PAIRS*)
A class to fix charges and reionize a molecule such that the strongest acids ionize first.

Initialize a Reionizer with the following parameter:

Parameters

- **acid_base_pairs** – A list of *AcidBasePairs* (page 28) to reionize, sorted from strongest to weakest.
- **charge_corrections** – A list of ChargeCorrections.

__call__(*mol*)
Calling a Reionizer instance like a function is the same as calling its reionize(mol) method.

reionize(*mol*)
Enforce charges on certain atoms, then perform competitive reionization.

First, charge corrections are applied to ensure, for example, that free metals are correctly ionized. Then, if a molecule with multiple acid groups is partially ionized, ensure the strongest acids ionize first.

The algorithm works as follows:

- Use SMARTS to find the strongest protonated acid and the weakest ionized acid.
- If the ionized acid is weaker than the protonated acid, swap proton and repeat.

Parameters **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The molecule to reionize.

Returns The reionized molecule.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

class molvs.charge.**Uncharger**(*acid_base_pairs=(AcidBasePair('-OSO3H', 'OS(=O)(=O)[OH]', 'OS(=O)(=O)[O-]'), AcidBasePair('–SO3H', '[!O]S(=O)(=O)[OH]', '[!O]S(=O)(=O)[O-]'), AcidBasePair('-OSO2H', 'O[SD3](=O)[OH]', 'O[SD3](=O)[O-]'), AcidBasePair('-SO2H', '[!O][SD3](=O)[OH]', '[!O][SD3](=O)[O-]'), AcidBasePair('-OPO3H2', 'OP(=O)([OH])[OH]', 'OP(=O)([OH])[O-]'), AcidBasePair('-PO3H2', '[!O]P(=O)([OH])[OH]', '[!O]P(=O)([OH])[O-]'), AcidBasePair('-CO2H', 'C(=O)[OH]', 'C(=O)[O-]'), AcidBasePair('thiophenol', 'c[SH]', 'c[S-]'), AcidBasePair('(-OPO3H)-', 'OP(=O)([O-])[OH]', 'OP(=O)([O-])[O-]'), AcidBasePair('(-PO3H)-', '[!O]P(=O)([O-])[OH]', '[!O]P(=O)([O-])[O-]'), AcidBasePair('phthalimide', 'O=C2c1ccccc1C(=O)[NH]2', 'O=C2c1ccccc1C(=O)[N-]2'), AcidBasePair('CO3H (peracetyl)', 'C(=O)O[OH]', 'C(=O)O[O-]'), AcidBasePair('alpha-carbon-hydrogen-nitro group', 'O=N(O)[CH]', 'O=N(O)[C-]'), AcidBasePair('-SO2NH2', 'S(=O)(=O)[NH2]', 'S(=O)(=O)[NH-]'), AcidBasePair('-OBO2H2', 'OB([OH])[OH]', 'OB([OH])[O-]'), AcidBasePair('-BO2H2', '[!O]B([OH])[OH]', '[!O]B([OH])[O-]'), AcidBasePair('phenol', 'c[OH]', 'c[O-]'), AcidBasePair('SH (aliphatic)', 'C[SH]', 'C[S-]'), AcidBasePair('(-OBO2H)-', 'OB([O-])[OH]', 'OB([O-])[O-]'), AcidBasePair('(-BO2H)-', '[!O]B([O-])[OH]', '[!O]B([O-])[O-]'), AcidBasePair('cyclopentadiene', 'C1=CC=C[CH2]1', 'c1ccc[cH-]1'), AcidBasePair('-CONH2', 'C(=O)[NH2]', 'C(=O)[NH-]'), AcidBasePair('imidazole', 'c1cnc[nH]1', 'c1cnc[n-]1'), AcidBasePair('-OH (aliphatic alcohol)', '[CX4][OH]', '[CX4][O-]'), AcidBasePair('alpha-carbon-hydrogen-keto group', 'O=C([!O])[C!H0+0]', 'O=C([!O])[C-]'), AcidBasePair('alpha-carbon-hydrogen-acetyl ester group', 'OC(=O)[C!H0+0]', 'OC(=O)[C-]'), AcidBasePair('sp carbon hydrogen', 'C#[CH]', 'C#[C-]'), AcidBasePair('alpha-carbon-hydrogen-sulfone group',*

Class for neutralizing charges in a molecule.

This class uncharges molecules by adding and/or removing hydrogens. In cases where there is a positive charge that is not neutralizable, any corresponding negative charge is also preserved.

__call__(*mol*)
Calling an Uncharger instance like a function is the same as calling its uncharge(mol) method.

uncharge(*mol*)
Neutralize molecule by adding/removing hydrogens.

Parameters **mol** (*rdkit.Chem.rdchem.Mol* (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.html#rdkit.Chem.rdchem.Mol)) – The molecule to uncharge.

Returns The uncharged molecule.

Return type rdkit.Chem.rdchem.Mol (http://www.rdkit.org/docs/source/rdkit.Chem.rdchem.

3.1.7 molvs.validate

This module contains the main *Validator* (page 31) class that can be used to perform all *Validations* (page 32), as well as the *validate_smiles()* (page 32) convenience function.

molvs.validate.**SIMPLE_FORMAT = '%(levelname)s: [%(validation)s] %(messag**
The default format for log messages.

molvs.validate.**LONG_FORMAT = '%(asctime)s - %(levelname)s - %(validation)**
A more detailed format for log messages. Specify when initializing a Validator.

class molvs.validate.**Validator**(*validations=VALIDATIONS*, *log_format=SIMPLE_FORMAT*, *level=logging.INFO*, *stdout=False*, *raw=False*)
The main class for running *Validations* (page 32) on molecules.

Initialize a Validator with the following parameters:

Parameters

- **validations** – A list of Validations to apply (default: *VALIDATIONS* (page 32)).
- **log_format** (*string*) – A string format (default: *SIMPLE_FORMAT* (page 31)).
- **level** – The minimum logging level to output.
- **stdout** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Whether to send log messages to standard output.

- **raw** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Whether to return raw `LogRecord` (https://docs.python.org/3/library/logging.html#logging.LogRecord) objects instead of formatted log strings.

__call__(*mol*)
Calling a Validator instance like a function is the same as calling its `validate()` method.

`molvs.validate.`**validate_smiles**(*smiles*)
Return log messages for a given SMILES string using the default validations.

Note: This is a convenience function for quickly validating a single SMILES string. It is more efficient to use the *Validator* (page 31) class directly when working with many molecules or when custom options are needed.

Parameters **smiles** (*string*) – The SMILES for the molecule.

Returns A list of log messages.

Return type list of strings.

3.1.8 molvs.validations

This module contains all the built-in *Validations* (page 32).

`molvs.validations.`**VALIDATIONS**
The default list of *Validations* (page 32) used by *Validator* (page 31).

class `molvs.validations.`**Validation**(*log*)
The base class that all *Validation* (page 32) subclasses must inherit from.

class `molvs.validations.`**SmartsValidation**(*log*)
Abstract superclass for *Validations* (page 32) that log a message if a SMARTS pattern matches the molecule.

Subclasses can override the following attributes:

level = 20
The logging level of the message.

message = 'Molecule matched %(smarts)s'
The message to log if the SMARTS pattern matches the molecule.

entire_fragment = False
Whether the SMARTS pattern should match an entire covalent unit.

smarts
The SMARTS pattern as a string. Subclasses must implement this.

class `molvs.validations.`**IsNoneValidation**(*log*)
Logs an error if `None` is passed to the Validator.

This can happen if RDKit failed to parse an input format. If the molecule is `None`, no subsequent validations will run.

class molvs.validations.**NoAtomValidation**(*log*)
Logs an error if the molecule has zero atoms.

If the molecule has no atoms, no subsequent validations will run.

class molvs.validations.**DichloroethaneValidation**(*log*)
Logs if 1,2-dichloroethane is present.

This is provided as an example of how to subclass *SmartsValidation* (page 32) to check for the presence of a substructure.

class molvs.validations.**FragmentValidation**(*log*)
Logs if certain fragments are present.

Subclass and override the `fragments` class attribute to customize the list of *FragmentPatterns* (page 26).

class molvs.validations.**NeutralValidation**(*log*)
Logs if not an overall neutral system.

class molvs.validations.**IsotopeValidation**(*log*)
Logs if molecule contains isotopes.

3.1.9 molvs.cli

This module contains a command line interface for standardization.

3.1.10 molvs.errors

This module contains exceptions that are raised by MolVS.

exception molvs.errors.**MolVSError**

exception molvs.errors.**StandardizeError**

exception molvs.errors.**ValidateError**

exception molvs.errors.**StopValidateError**
Called by Validations to stop any further validations from being performed.

CHAPTER 4

Useful links

- MolVS on GitHub (https://github.com/mcs07/MolVS)
- MolVS on PyPI (https://pypi.python.org/pypi/MolVS)
- Issue tracker (https://github.com/mcs07/MolVS/issues)
- Release history (https://github.com/mcs07/MolVS/releases)
- MolVS Travis CI (https://travis-ci.org/mcs07/MolVS)

Index

Symbols

A

C

D

E

F

I

L

M

N

P

R

S

T

U

V