class scenario.Rule (rulename, probability, smarts)
    Class to contain a metabolic rule
    Parameters
    • rulename – A string containing a unique name of the rule
    • probability – A probability value between 0 and 1 indicating the empirical success rate of the rule
    • smarts – A reaction smarts describing the chemical transformation of the rule

class scenario.Scenario (scenario)
    Class to read and process metabolic scenario
    Parameters scenario – A list of lists, each representing a metabolic phase as [name_of_file_containing_rules, number_of_cycles_to_apply]

run (parentmol)
    Parameters parentmol – An RDKit molecule
    Returns A sygma.Tree object

class tree.Tree (parentmol=None)
    Class to build and analyse a metabolic tree
    Parameters parentmol – An RDKit molecule

    add_coordinates ()
        Add missing atomic coordinates to all metabolites

    calc_scores ()
        Calculate probability scores for all metabolites

    metabolize_all_nodes (rules, cycles=1)
        Metabolize all nodes according to [rules], for [cycles] number of cycles
        Parameters
- **rules** – List of rules
- **cycles** – Integer indicating the number of subsequent steps to apply the rules

```python
to_list(filter_small_fragments=True, parent_column='parent')
```
Generate a list of metabolites

**Parameters**

- **filter_small_fragments** – Boolean to activate filtering all metabolites with less than 15% of original atoms (of the parent)
- **parent_column** – String containing the name for the column with the parent molecule

**Returns** A list of dictionaries for each metabolite, containing the SyGMa_metabolite (an RDKit Molecule), SyGMa_pathway and SyGMa_score, sorted by decreasing probability.

```python
to_smiles(filter_small_fragments=True)
```
Generate a smiles list of metabolites

**Parameters**

- **filter_small_fragments** – Boolean to activate filtering all metabolites with less then 15% of original atoms (of the parent)

**Returns** A list of metabolites as list [[SyGMa_metabolite as smiles, SyGMa_score]] sorted by decreasing probability score.

```python
write_sdf(file=<open file '<stdout>', mode 'w'), filter_small_fragments=True)
```
Generate an SDFile with metabolites including the SyGMa_pathway and the SyGMa score as properties

**Parameters**

- **file** – The SDF file to write to
- **filter_small_fragments** – Boolean to activate filtering all metabolites with less then 15% of original atoms (of the parent)

```python
class treenode.TreeNode (mol, parents=None, rule=None, score=None, pathway='')
```
Class containing a node of the SyGMa tree

**Key**
- **mol** RDKit Molecule
- **parents** Dictionary {inchikey_of_parent: rulename_transforming_parent_to_self}
- **children** List of inchikeys of the child nodes
- **score** Value between 0 and 1
- **pathway** String describing the pathway from parent to self
- **n_original_atoms** Integer, number of atoms originating from parent or None if not yet determined

```python
gen_coords ()
```
Calculate 2D positions for atoms in self.mol without coordinates


SyGMA: Systematically Generating potential Metabolites

```
         [-l {debug,info,warn, error}]
         parentmol
```

2.1 Positional Arguments

```
parentmol   Smiles string of parent molecule structure
```

2.2 Named Arguments

```
--version      show program’s version number and exit
-o, --outputtype Molecule output type (default: sdf)
                   Default: sdf
-1, --phase1    Number of phase 1 cycles (default: 1)
                   Default: 1
-2, --phase2    Number of phase 2 cycles (default: 1)
                   Default: 1
-l, --loglevel  Possible choices: debug, info, warn, error
                   Set logging level (default: “info”)
                   Default: “info”
```
CHAPTER 3

Introduction


3.1 Requirements

SyGMA requires RDKit with INCHI support

3.2 Installation

- Install with Anaconda: `conda install -c 3d-e-Chem -c rdkit sygma`
  OR
- Install RDKit following the instructions in http://www.rdkit.org/docs/Install.html
  AND
- `pip install sygma` OR, after downloading sygma, `python setup.py install`

3.3 Example

```python
import sygma
from rdkit import Chem

def test_predict_phenol_metabolites():
    """Test prediction of phenol metabolites by sygma module""
```

(continues on next page)
# Each step in a scenario lists the ruleset and the number of reaction cycles to be applied
```python
scenario = sygma.Scenario(
    [sygma.ruleset['phase1'], 1],
    [sygma.ruleset['phase2'], 1])
```

# An rdkit molecule, optionally with 2D coordinates, is required as parent
```python
parent = Chem.MolFromSmiles("c1ccccc1O")
metabolic_tree = scenario.run(parent)
metabolic_tree.calc_scores()
metabolite_list = metabolic_tree.to_list()
```

```python
assert len(metabolite_list) == 12
assert metabolite_list[0]['SyGMa_score'] == 1
assert metabolite_list[1]['SyGMa_pathway'] == 'O-glucuronidation_(aromatic_hydroxyl);
```

## 3.4 Docker

SyGMA can be executed in a Docker container as follows:
```bash
docker run 3dechem/sygma c1ccccc1O
```

## 3.5 Rulesets

SyGMA comes currently with two rulesets:

**phase1**  Phase 1 metabolism rules include mainly different types of oxidation, hydrolysis, reduction and condensation reactions

**phase2**  Phase 2 metabolism rules include severaly conjugation reaction, i.e. with glucuronyl, sulfate, methyl and acetyl
CHAPTER 4

Indices and tables

- genindex
- modindex
- search
r
ruleset, 8

s
scenario, 3

t
tree, 3
treenode, 4
<table>
<thead>
<tr>
<th>Letter</th>
<th>Method/Class</th>
<th>Module/Group</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>add_coordinates()</td>
<td>tree.Tree method</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>calc_scores()</td>
<td>tree.Tree method</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>gen_coords()</td>
<td>treenode.TreeNode method</td>
<td>4</td>
</tr>
<tr>
<td>M</td>
<td>metabolize_all_nodes()</td>
<td>tree.Tree method</td>
<td>3</td>
</tr>
<tr>
<td>R</td>
<td>Rule (class in scenario)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ruleset (module)</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>run() (scenario.Scenario method)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Scenario (class in scenario)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>scenario (module)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>to_list()</td>
<td>tree.Tree method</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>to_smiles()</td>
<td>tree.Tree method</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Tree (class in tree)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tree (module)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TreeNode (class in treenode)</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>treenode (module)</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>write_sdf()</td>
<td>tree.Tree method</td>
<td>4</td>
</tr>
</tbody>
</table>