The RDKit Book

Original Author: Greg Landrum
Version: Q2 2010

Table of Contents

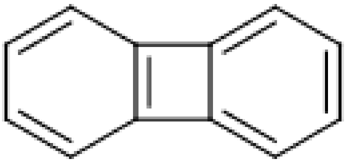
1 Misc Cheminformatics Topics

1.1 Aromaticity

Aromaticity is one of those unpleasant topics that is simultaneously simple and impossibly complicated. Since neither experimental nor theoretical chemists can agree with each other about a definition, it's necessary to pick something arbitrary and stick to it. This is the approach taken in the RDKit.

Instead of using patterns to match known aromatic systems, the aromaticity perception code in the RDKit uses a set of rules. The rules are relatively straightforward.

Aromaticity is a property of atoms and bonds in rings. An aromatic bond must be between aromatic atoms, but a bond between aromatic atoms does not need to be aromatic.
For example the fusing bonds here are not considered to be aromatic by the RDKit:



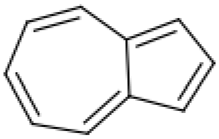
```
>>> from rdkit import Chem
>>> m = Chem.MolFromSmiles('C1=CC2=C(C=C1)C1=CC=CC=C21')
>>> m.GetAtomWithIdx(3).GetIsAromatic()
True
>>> m.GetAtomWithIdx(6).GetIsAromatic()
True
>>> m.GetBondBetweenAtoms(3,6).GetIsAromatic()
False
```

A ring, or fused ring system, is considered to be aromatic if it obeys the 4N+2 rule. Contributions to the electron count are determined by atom type and environment. Some examples:

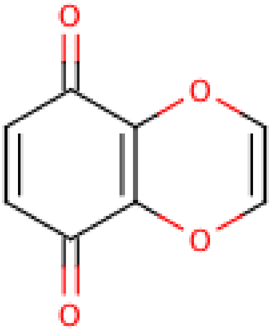
Fragment	Number of pi electrons
c(a)a	1
n(a)a	1
An(a)a	2
o(a)a	2
s(a)a	2
se(a)a	2
te(a)a	2
O=c(a)a	0
N=c(a)a	0
*(a)a	0, 1, or 2

Notation a: any aromatic atom; A: any atom, include H; *: a dummy atom
Notice that exocyclic bonds to electronegative atoms “steal” the valence electron from the ring atom and that dummy atoms contribute whatever count is necessary to make the ring aromatic.

The use of fused rings for aromaticity can lead to situations where individual rings are not aromatic, but the fused system is. An example of this is azulene:



An extreme example, demonstrating both fused rings and the influence of exocyclic double bonds:



```
>>> m=Chem.MolFromSmiles('O=C1C=CC(=O)C2=C1OC=CO2')
>>> m.GetAtomWithIdx(6).GetIsAromatic()
```

```
True
>>> m.GetAtomWithIdx(7).GetIsAromatic()
True
>>> m.GetBondBetweenAtoms(6,7).GetIsAromatic()
False
```

1.2 Ring Finding and SSSR

[Section taken from “Getting Started” document]
As others have ranted about with more energy and eloquence than I intend to, the definition of a molecule's smallest set of smallest rings is not unique. In some high symmetry molecules, a “true” SSSR will give results that are unappealing. For example, the SSSR for cubane only contains 5 rings, even though there are “obviously” 6. This problem can be fixed by implementing a *small* (instead of *smallest*) set of smallest rings algorithm that returns symmetric results. This is the approach that we took with the RDKit.

Because it is sometimes useful to be able to count how many SSSR rings are present in the molecule, there is a GetSSSR function, but this only returns the SSSR count, not the potentially non-unique set of rings.

2 Chemical Reaction Handling

2.1 Reaction SMARTS

Not SMIRKS[1], not reaction SMILES[2], derived from SMARTS[3].

The general grammar for a reaction SMARTS is

```
reaction    :     reactants '>>' products
reactants   :     molecules
products    :     molecules
molecules   :     molecule
            |     molecules '.' molecule
```

and a molecule is a valid SMARTS string (without '.' characters).

Some features

Mapped dummy atoms in the product template are replaced by the corresponding atom in the reactant:

```
>>> from rdkit.Chem import AllChem
>>> rxn = AllChem.ReactionFromSmarts('[C:1]=[O,N:2]>>[C:1][*:2]')
>>> [Chem.MolToSmiles(x,1) for x in
rxn.RunReactants((Chem.MolFromSmiles('CC=O'),))[0]]
['CCO']
>>> [Chem.MolToSmiles(x,1) for x in
rxn.RunReactants((Chem.MolFromSmiles('CC=N'),))[0]]
['CCN']
```

but unmapped dummy atoms are left as dummies:

```
>>> rxn = AllChem.ReactionFromSmarts('[C:1]=[O,N:2]>>[*][C:1][*:2]')
>>> [Chem.MolToSmiles(x,1) for x in
rxn.RunReactants((Chem.MolFromSmiles('CC=O'),))[0]]
```

[1] http://www.daylight.com/dayhtml/doc/theory/theory.smirks.html

[2] http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html

[3] http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html

```
['[*]C(C)O']
```

“Any” bonds in the products are replaced by the corresponding bond in the reactant:

```
>>> rxn = AllChem.ReactionFromSmarts('[C:1]~[O,N:2]>>[*][C:1]~[*:2]')
>>> [Chem.MolToSmiles(x,1) for x in
rxn.RunReactants((Chem.MolFromSmiles('C=O'),))[0]]
['[*]C=O']
>>> [Chem.MolToSmiles(x,1) for x in
rxn.RunReactants((Chem.MolFromSmiles('CO'),))[0]]
['[*]CO']
>>> [Chem.MolToSmiles(x,1) for x in
rxn.RunReactants((Chem.MolFromSmiles('C#N'),))[0]]
['[*]C#N']
```

Rules and caveats

1. Include atom map information at the end of an atom query. So do `[C,N,O:1]` or `[C;R:1]`.
2. Don't forget that unspecified bonds in SMARTS are either single or aromatic. Bond orders in product templates are assigned when the product template itself is constructed and it's not always possible to tell if the bond should be single or aromatic:

```
>>> rxn = AllChem.ReactionFromSmarts('[#6:1][#7,#8:2]>>[#6:1][#6:2]')
>>> [Chem.MolToSmiles(x,1) for x in
rxn.RunReactants((Chem.MolFromSmiles('C1NCCCC1'),))[0]]
['C1CCCCC1']
>>> [Chem.MolToSmiles(x,1) for x in
rxn.RunReactants((Chem.MolFromSmiles('c1ncccc1'),))[0]]
['c1cccc-c1']
```

So if you want to copy the bond order from the reactant, use an “Any” bond:

```
>>> rxn = AllChem.ReactionFromSmarts('[#6:1][#7,#8:2]>>[#6:1]~[#6:2]')
>>> [Chem.MolToSmiles(x,1) for x in
rxn.RunReactants((Chem.MolFromSmiles('c1ncccc1'),))[0]]
['c1ccccc1']
```

3 The Feature Definition File Format

An FDef file contains all the information needed to define a set of chemical features. It contains definitions of feature types that are defined from queries built up using Daylight's SMARTS language. The FDef file can optionally also include definitions of atom types that are used to make feature definitions more readable.

3.1 Chemical Features

Chemical features are defined by a `Feature Type` and a `Feature Family`. The `Feature Family` is a general classification of the feature (such as "Hydrogen-bond Donor" or "Aromatic") while the `Feature Type` provides additional, higher-resolution, information about features. Pharmacophore matching is done using `Feature Family's`.
Each feature type contains the following pieces of information:

The family to which the feature belongs.

- A SMARTS pattern that describes atoms (one or more) matching the feature type.
- Weights used to determine the feature's position based on the positions of its defining atoms.

3.2 Syntax of the FDef file

3.2.1 AtomType definitions

An AtomType definition allows you to assign a shorthand name to be used in place of a SMARTS string defining an atom query. This allows FDef files to be made much more readable. For example, defining a non-polar carbon atom like this:

```
AtomType Carbon_NonPolar [C&!$(C=[O,N,P,S])&!$(C#N)]
```

creates a new name that can be used anywhere else in the FDef file that it would be useful to use this SMARTS. To reference an AtomType, just include its name in curly brackets. For example, this excerpt from an FDef file defines another atom type - `Hphobe` - which references the `Carbon_NonPolar` definition:

```
AtomType Carbon_NonPolar [C&!$(C=[O,N,P,S])&!$(C#N)]
AtomType Hphobe [{Carbon_NonPolar},c,s,S&H0&v2,F,Cl,Br,I]
```

Note that `{Carbon_NonPolar}` is used in the new AtomType definition without any additional decoration (no square brackes or recursive SMARTS markers are required).

Repeating an AtomType results in the two definitions being combined using the SMARTS `","` (or) operator. Here's an example:

```
AtomType d1 [N&!H0]
AtomType d1 [O&!H0]
```

This is equivalent to:

```
AtomType d1 [N&!H0,O&!H0]
```

Which is equivalent to the more efficient:

```
AtomType d1 [N,O;!H0]
```

Note that these examples tend to use SMARTS's high-precendence and operator `"&"` and not the low-precedence and `";"`. This can be important when AtomTypes are combined or when they are repeated. The SMARTS `","` operator is higher precedence than `";"`, so definitions that use `";"` can lead to unexpected results.

It is also possible to define negative AtomType queries:

```
AtomType d1 [N,O,S]
AtomType !d1 [H0]
```

The negative query gets combined with the first to produce a definition identical to this:

```
AtomType d1 [!H0;N,O,S]
```

Note that the negative AtomType is added to the beginning of the query.

3.2.2 Feature definitions

A feature definition is more complex than an AtomType definition and stretches across multiple lines:

```
DefineFeature HDonor1 [N,O;!H0]
   Family HBondDonor
   Weights 1.0
EndFeature
```

The first line of the feature definition includes the feature type and the SMARTS string defining the feature.

The next two lines (order not important) define the feature's family and its atom weights (a comma-delimited list that is the same length as the number of atoms defining the feature). The atom weights are used to calculate the feature's locations based on a weighted average of the positions of the atom

defining the feature. More detail on this is provided below.
The final line of a feature definition must be `EndFeature`. It is perfectly legal to mix AtomType definitions with feature definitions in the FDef file. The one rule is that AtomTypes must be defined before they are referenced.

3.2.3 Additional syntax notes:

- Any line that begins with a # symbol is considered a comment and will be ignored.
- A backslash character, `\`, at the end of a line is a continuation character, it indicates that the data from that line is continued on the next line of the file. Blank space at the beginning of these additional lines is ignored. For example, this AtomType definition:

  ```
  AtomType tButylAtom [$([C;!R](-[CH3])(-[CH3])(-[CH3])),\
                       $([CH3](-[C;!R](-[CH3])(-[CH3])))]
  ```

  is exactly equivalent to this one:

  ```
  AtomType tButylAtom [$([C;!R](-[CH3])(-[CH3])(-[CH3])),$([CH3](-[C;!R]
  (-[CH3])(-[CH3])))]
  ```

  (though the first form is much easier to read!)

Atom weights and feature locations

3.3 Frequently Asked Question(s)

- What happens if a `Feature Type` is repeated in the file? Here's an example:

  ```
  DefineFeature HDonor1 [O&!H0]
      Family HBondDonor
      Weights 1.0
  EndFeature
  DefineFeature HDonor1 [N&!H0]
      Family HBondDonor
      Weights 1.0
  EndFeature
  ```

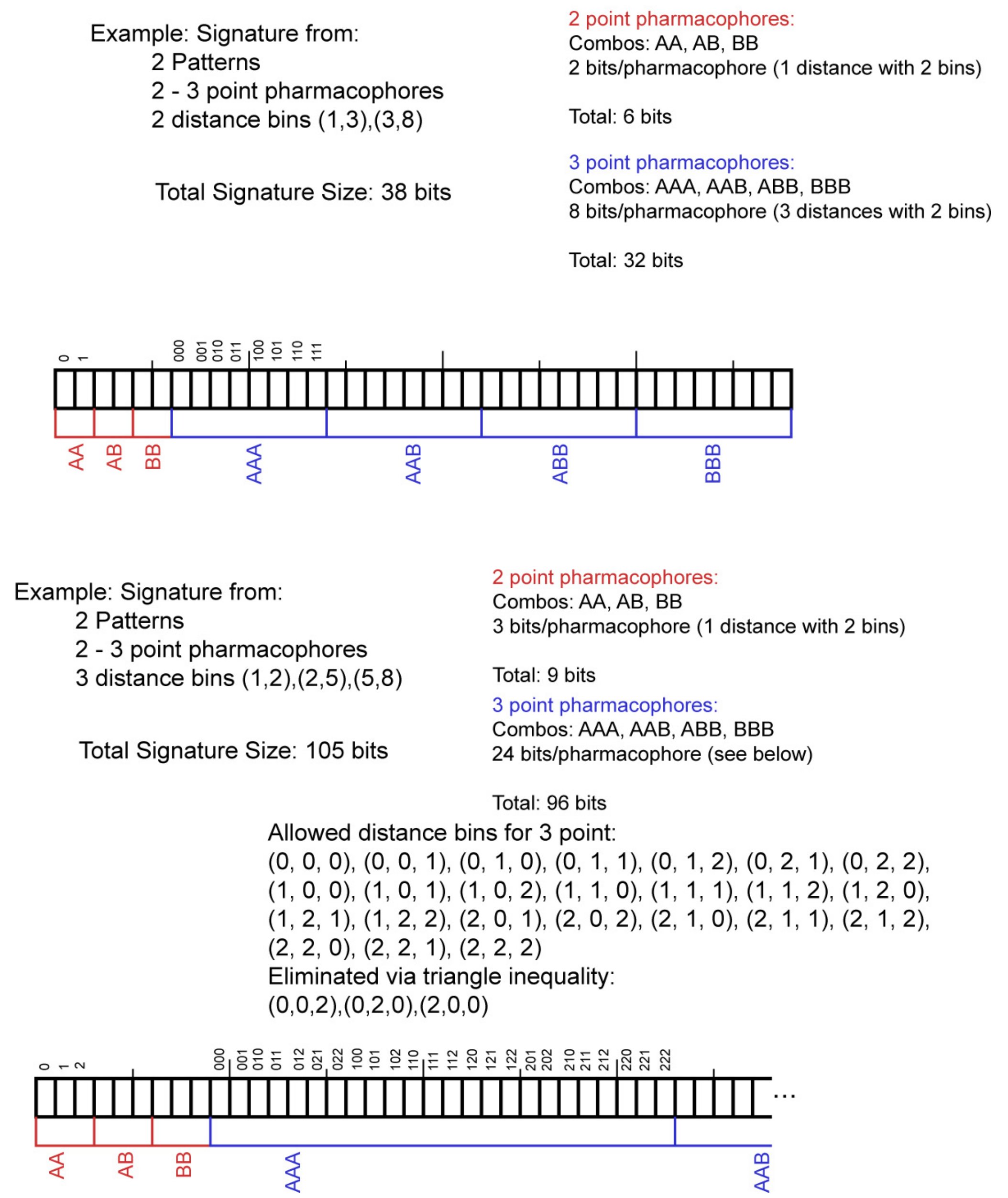
  In this case both definitions of the `HDonor1` feature type will be active. This is functionally identical to:

  ```
  DefineFeature HDonor1 [O,N;!H0]
      Family HBondDonor
      Weights 1.0
  EndFeature
  ```

  However the formulation of this feature definition with a duplicated feature type is considerably less efficient and more confusing than the simpler combined definition.

4 Representation of Pharmacophore Fingerprints

In the RDKit scheme the bit ids in pharmacophore fingerprints are not hashed: each bit corresponds to a particular combination of features and distances. A given bit id can be converted back to the corresponding feature types and distances to allow interpretation. An illustration for 2D pharmacophores is shown in Figure 1.



Example: Signature from:
2 Patterns
2 - 3 point pharmacophores
2 distance bins (1,3),(3,8)

Total Signature Size: 38 bits

2 point pharmacophores:
Combos: AA, AB, BB
2 bits/pharmacophore (1 distance with 2 bins)

Total: 6 bits

3 point pharmacophores:
Combos: AAA, AAB, ABB, BBB
8 bits/pharmacophore (3 distances with 2 bins)

Total: 32 bits

Example: Signature from:
2 Patterns
2 - 3 point pharmacophores
3 distance bins (1,2),(2,5),(5,8)

Total Signature Size: 105 bits

2 point pharmacophores:
Combos: AA, AB, BB
3 bits/pharmacophore (1 distance with 2 bins)

Total: 9 bits

3 point pharmacophores:
Combos: AAA, AAB, ABB, BBB
24 bits/pharmacophore (see below)

Total: 96 bits

Allowed distance bins for 3 point:
(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (0, 1, 2), (0, 2, 1), (0, 2, 2), (1, 0, 0), (1, 0, 1), (1, 0, 2), (1, 1, 0), (1, 1, 1), (1, 1, 2), (1, 2, 0), (1, 2, 1), (1, 2, 2), (2, 0, 1), (2, 0, 2), (2, 1, 0), (2, 1, 1), (2, 1, 2), (2, 2, 0), (2, 2, 1), (2, 2, 2)
Eliminated via triangle inequality:
(0,0,2),(0,2,0),(2,0,0)

Figure 1: Bit Numbering in Pharmacophore Fingerprints

5 License

The intent of this license is similar to that of the RDKit itself. In simple words: “Do whatever you want with it, but please give us some credit.”