OpenEye
Scientific Software

OMEGA

Release 2.5.1.4

OpenEye Scientific Software, Inc.

May 15, 2013

CONTENTS

CHAPTER

ONE

FRONT MATTER

CHAPTER

TWO

INSTALLATION AND PLATFORM NOTES

2.1 Licenses

To run OMEGA you will need to obtain a license file for OMEGA from OpenEye Scientific Software (business@eyesopen.com). The license file should be pointed to by the environment variable **OE_LICENSE**.

If you intend on running multi-processor OMEGA via Open MPI, only the master machine needs access to the license file.

On Windows, the environment variables can be set under the system Control Panel.

2.2 General Installation

2.2.1 Linux

Linux distributions are provided as a gzipped tarball of the distribution tree described below. Installation is performed by simply untarring the file in the desired location. The top-level directory in the tarball is named `openeye`. Distributions for different Linux variants can be installed into the same location, allowing multiple Linux versions to be run from a single shared directory.

To ensure that the installed applications can be called from the command line, be sure to add the full path of the `openeye/bin` subdirectory to the **PATH** environment variable. For instance, if the distribution was installed into `/usr/local/openeye`, the **PATH** environment variable should contain: `/usr/local/openeye/bin`.

Under the top-level `openeye` directory are the following subdirectories:

arch This directory contains the collection of platform specific subdirectories. Each subdirectory contains the actual installed executables and support libraries for the associated platform. In the platform specific subdirectory there will be a subdirectory for each application. Within that will be another subdirectory for each version of that application.

bin This directory contains a startup script for each application that has been installed. This script determines, at run-time, what the current platform is and then calls the appropriate executable in the `arch`. This script enables the easy co-existence of multiple platforms and versions of any OpenEye application in the same distribution tree.

data This directory contains all of the associated data for the installed applications. There will be a subdirectory for each installed application and within that subdirectory there will be another subdirectory for each specific version of that application.

docs This directory contains all of the documentation associated with the installed applications. There will be a subdirectory for each installed application and within that subdirectory there will be another subdirectory for each specific version of that application.

examples This directory contains all of the examples associated with the installed applications. There will be a subdirectory for each installed application and within that subdirectory there will be another subdirectory for each specific version of that application.

The startup script discussed in the section on the `bin` directory above will have the same name as the installed executable with which it is associated. When the script is called, it will attempt to determine the current platform and run the appropriate executable if installed. If an appropriate executable cannot be found, the script will report that information, as well as a list of the currently installed platforms. The auto-detection can be overridden by setting one of two environment variables:

- **OE_ARCH** can be used to specify a colon separated list of compatible distributions for the current platform such as:

  ```
  redhat-RHEL5-x64:redhat-RHEL4-x64
  ```

  Specification of this environment variable overrides the auto-detection process, if it is present. If none of the compatible distributions listed are found, the script will fall back to the auto-detection process.

- **APPNAME_OE_ARCH** can be used to specify a colon separated list of compatible distributions for a specific application (as specified by changing the **APPNAME** text in the environment variable name) just like **OE_ARCH** as detailed above.

  Specification of this environment variable overrides the **OE_ARCH** environment variable as well as the auto-detection process. If none of the compatible distributions listed are found, the script will fall back to the **OE_ARCH** list first and then to the auto-detection process.

  Specifying this variable provides a simple way to customize the behavior for individual applications on non-standard platforms.

The startup script also supports a few commandline arguments including:

-path	Specifying this argument will output the full path of the executable to be run. The executable will not be started if this argument is present.
-print_arch	Specifying this argument will output the details of the current platform as detected by the script as well as which platform-version of the executable is being run. The executable will be started if this argument is present.
-use_version	Specifying this argument followed by a specific version number allows the user to control which released version of the executable to run.

2.2.2 Windows

Windows distributions are provided as a standard EXE installer. For installation double click the executable and follow the installation instructions. By default, OpenEye applications will install into the `C:\Program Files (x86)` directory (for 32-bit applications) or `C:\Program Files` (for 64-bit applications).

Under the application directory (`C:\Program Files\Application Name`) there are subdirectories for:

bin This directory contains the application executable.

data This directory contains all of the associated data for the installed applications.

docs This directory contains all of the documentation associated with the installed applications.

examples This directory contains all of the examples associated with the installed applications.

An OpenEye group with an application specific subgroup will be added to the *Start* menu. The application specific subgroup will contain links to the documentation, the uninstaller, and, for some applications, a Windows command shell with **PATH** settings already defined to allow the user to simply type the executable name at the prompt without concern for where the executable is actually installed. Links are also included to add and remove the installed location to the user's default path.

For graphical applications, a link to the application will be created on the desktop as well as in the application specific subgroup of the *Start* menu.

2.2.3 Mac OS X

Mac OS X distributions are provided as a *dmg* disk image. For installation, double click the .dmg file to open it, and drag the application to the Applications folder.

A folder containing documentation and example data is included in the disk image (Right click, `Show Package Contents`). Under the top level `Contents` folder there are subdirectories for:

data This directory contains all of the associated data for the installed applications.

docs This directory contains all of the documentation associated with the installed applications.

MacOS/bin This directory contains the application executable.

The documentation and example data can be copied to any convenient location. Graphical applications have built-in documentation, which is available from the application's Help menu.

For command-line only tools, an application named "Install Command Line Support" can be run from the .dmg file, and this will allow the user to add the application's location to the user's **PATH** environment variable. Command-line applications can also be run from the Applications folder, in which case they will open a terminal window with a properly configured environment.

2.3 MPI

2.3.1 Open MPI

This application uses the Open MPI implementation of MPI, found at http://www.open-mpi.org. This version includes a full Open MPI install, so no additional software is needed.

There are two requirements to run under Open MPI.

- Every machine in the cluster must have the same version of the application installed.
- The path to the application's `bin` directory must be in the **PATH** environment variable on all machines, and before any other locations that may contain MPI executables (orted, mpirun, etc).

2.3.2 Using Open MPI

To run under Open MPI on a single machine:

```
prompt> [application executable] -mpi_np [number of processes] [other application options]
```

To run Open MPI on multiple machines, begin by generating a text file that will include the MPI hosts you plan to use and the number of processes on each (for this example, we'll call this file `hosts`). The file should contain a line for each machine with the name of the machine, a space, then `slots=N`, where N is the number of processors for your run. For this example, you would want the file to look like:

```
c1 slots=4
c2 slots=4
c3 slots=4
c4 slots=4
c5 slots=4
```

Now the following command-line will start a job with 1 master and 19 slaves.

```
prompt> [application executable] -mpi_hostfile hosts [other application options]
```

The master will be on c1, where the job is being started. All the i/o for the run will be from the master machine, and all results and logging information will be combined.

2.3.3 Open MPI License

2.4 Uninstallation

2.4.1 Linux

To uninstall a single distribution of a product the relevant subdirectories for that product and version simply need to be deleted from within the following directories:

arch In the `openeye/arch` directory is a platform specific subdirectory. Within this are directories for each installed product and within those are subdirectories for each version of the product. Delete the subdirectory for the version which is to be uninstalled. For example, to delete or uninstall v1.0.0 of a product, delete the folder "<product_name>/1.0.0".

data In the `openeye/data` directory is a subdirectory for each installed product and within those are subdirectories for each version of the product. Delete the subdirectory for the version which is to be uninstalled.

docs In the `openeye/docs` directory is a subdirectory for each installed product and within those are subdirectories for each version of the product. Delete the subdirectory for the version which is to be uninstalled.

examples In the `openeye/examples` directory is a subdirectory for each installed product and within those are subdirectories for each version of the product. Delete the subdirectory for the version which is to be uninstalled.

2.4.2 Windows

Installation of an OpenEye product on Windows causes an OpenEye group with an application specific subgroup to be added to the *Start* menu. One of the items in the application specific subgroup is a link to the uninstaller. Clicking on the uninstaller initiates a wizard which guides the user through uninstallation.

For graphical applications, uninstallation also removes the application's link from the desktop and *Start* menu.

2.4.3 Mac OS X

To uninstall a single distribution of an application simply drag the application from the Application folder to the Trash.

Some applications may, at the user's request, install symbolic links in the /usr/local/bin directory, and modify the PATH variable in the .openeyerc.* file(s) in the user's home directory. The symbolic links may be safely deleted after uninstallation, and the .openeyerc.* file(s) can be edited, if desired, to remove obsolete entries.

CHAPTER

THREE

OMEGA THEORY

OMEGA is a conformation generator for molecules. OMEGA is composed of two main components; model building of molecular fragments and torsion driving. Model generation may be bypassed by importing fragment structures from external sources.

OMEGA builds initial models of structures by assembling fragment templates along sigma bonds. Input molecules' graphs are fragmented at exocyclic sigma, and carbon to heteroatom acyclic (but not exocyclic) sigma bonds. Conformations for the fragments are either retrieved from pregenerated libraries built with *makefraglib*, or constructed on-the-fly using the same distance constraints followed by geometry optimization protocol that *makefraglib* uses. Molecule assembly is accomplished by simple vector alignment since all inter-fragment joints are along sigma bonds.

Once an initial model of a structure is constructed, or given as input, OMEGA generates additional models by enumerating ring conformations and invertible nitrogen atoms. Ring conformations are taken from the same fragment library used to build an initial model. OMEGA detaches all exocyclic substituents from a ring system, aligns and attaches them relative to the new ring conformation. OMEGA attempts to generate every possible combination of ring conformations possible for a given structure.

The next step in model generation is to detect and enumerate invertible nitrogens. Nitrogens that have pyramidal geometry, no stereochemistry specified, no more than one hydrogen, are three valent, and have no more than three ring bonds are considered by OMEGA to be invertible. Invertible in this context simply means that at room temperature a pyramidal nitrogen is likely to be able to rapidly (on an NMR timescale) interconvert between two puckered forms. All multiconformer ring models are further expanded by enumerating all possible nitrogen puckers. The resulting model set is the starting point for conformer search by torsion driving.

OMEGA begins the torsion search process by examining the molecular graph and determining the bonds that may freely rotate. By default, OMEGA selects acyclic sigma bonds that have at least one non-hydrogen atom attached to each end of the bond. By default, hydrogen rotors (e.g. hydroxyl groups) are not altered during the torsion search; however, this may be enabled as of version 2.5.1. The final ensemble selection is based on RMS distance of heavy atoms and sampled hydrogen atoms; unsampled hydrogen atoms do not affect this RMS distance. A list of possible dihedral angles are then assigned to each rotatable bond. The current mechanism for assignment is based on SMARTS matching, although alternate strategies for assigning angles based on experimental (i.e. X-ray) or theoretical (i.e. fragment optimization studies) are possible. The molecular graph is then subjected to pattern and geometric symmetry detection. Common patterns such as para-disubstituted benzene are used to reduce the number of symmetry equivalent dihedral angles that need to be searched. All torsions are altered by 120 and 180 degrees, and an RMS calculation is performed taking into account symmetry equivalent atoms in order to detect two and three fold symmetries. Exhaustive depth first torsion search is performed on each of the fragments, and the resulting conformers are placed into a list sorted by energy. Entire structures are assembled by combining the lowest energy set of fragments, and then the next lowest set, until the search is terminated. The search will terminate when the limit on the total number of conformers that may be generated is exceeded, the fragment list is exhausted, or the sum of the fragment energies exceeds the energy window of the global minimum structure. The best conformers identified in the torsion search are rank ordered by energy. A final ensemble is selected by sequentially testing the conformers using the RMS distance cutoff. To be accepted in to the final ensemble, a conformer must have an RMS distance to every other member of the ensemble that

exceeds the user defined cutoff value. The final ensemble is populated up to the user defined maximum ensemble size limit, or until the list of low energy conformers is exhausted.

3.1 Filtering

Filtering based on graph (and possibly physical) properties should always be carried out prior to generating multi-conformer databases using OMEGA. Eliminating undesirable compounds prior to generating conformers will save execution time of both OMEGA and down stream applications, and space on disk. Large polypeptides or proteins, very flexible molecules, or simply molecules that would never be considered useful for the ultimate modeling application are best eliminated from a data set as early as possible.

The most important graph filter to apply is rotatable bond count. Although OMEGA may be able to generate conformers for molecules with more than 20 rotatable bonds, the results of such an exercise would be dubious at best for any conformer generation method. Number of rings, especially flexible rings, should also be used to exclude irrelevant molecules. OMEGA will handle molecules that have many thousands of possible ring combinations, although the time expenditure may be prohibitive and the results equally questionable to molecules with an unreasonably large number of rotatable bonds. Simple element filters may be useful as well, although OMEGA will discard compounds for which no force field parameters exist. Using element filters beforehand may simply aid in tracking the rationale for discarding compounds instead of searching through OMEGA log files for failure modes.

The *filter* program from OpenEye Scientific Software provides all of the functionality outlined above, and additional physical property filters. It is highly recommended that *filter* or a program similar in functionality be used for input file preparation of large datasets.

3.2 Stereochemistry Enumeration

Compounds that contain unspecified or ambiguous definitions of stereochemistry may be preprocessed before generating conformers to add explicitly specified stereochemistry. Input molecules that have three dimensional coordinates inherently have stereochemistry specified, but SMILES or two dimensional SD files may have atoms (R/S) or bonds (E/Z) for which the stereochemistry is unknown or unspecified. OMEGA will generate structures for compounds of unknown configuration, however, in virtual screening exercises it may be beneficial to simply enumerate possible stereoisomers and treat each stereoisomer as a separate compound.

OMEGA distributions contain a utility called *flipper* that enumerates unspecified stereochemistry within user defined limits. For an explanation of how to use *flipper*, consult the *flipper: Enumerating Stereocenters* chapter in the OMEGA application manual. Stereochemistry enumeration is an exponential task. For every atom or bond (N) in a molecule that has two possible stereochemical 'states', there are 2^N possible stereoisomers for the molecule. Enumerating all possible stereoisomers for molecules may be unreasonable in terms of CPU time or storage space. *flipper* has user defined limits to regulate the enumeration process and keep it practical for routine applications.

Enumerating stereoisomers provides an information gain that may aid in operations downstream from conformer generation. *flipper*, or a similar stereochemistry enumeration tool should be used prior to generating conformers with OMEGA.

3.3 Fragment Library Generation

Prior to conformer generation, OMEGA builds a set of three dimensional models of a compound that contain the bond lengths, angles, and ring conformations that will be held fixed during the torsion search. OMEGA is capable of generating fragment templates on-the-fly, however, using pregenerated templates is far more efficient and will speed execution of conformer generation. OMEGA distributions include a program called *makefraglib* which can be used to create libraries of molecular fragments and ring conformers that can be used by OMEGA to build three dimensional

models of molecules. For an explanation of how to use *makefraglib*, consult the Makefraglib chapter in the OMEGA application manual. In practice, extensive fragment libraries need only to be constructed a single time and rarely need to be altered. Corporate and vendor databases can be used to construct an extensive fragment library. Once built, the fragment libraries will rarely need to be updated, and will provide a significant enhancement in performance.

CHAPTER

FOUR

OMEGA APPLICATION USAGE

4.1 Command Line Interface

A description of the command line interface can be obtained by executing OMEGA with the *--help* option.

```
prompt> omega2 --help
```

will generate the following output:

```
Help functions:
  omega2 --help simple      : Get a list of simple parameters
  omega2 --help all         : Get a complete list of parameters
  omega2 --help <parameter> : Get detailed help on a parameter
  omega2 --help html        : Create an html help file for this program
```

4.1.1 Required Parameters

-in
: File containing one or more molecular connection tables to be processed by OMEGA. Multiple input files of the same format can be merged together into a single input file using the *cat* command. A pipe can be used to avoid writing the merged input file to disk.

```
cat filename1.ism filename2.ism | omega2 -in .ism -out output.oeb.gz
```

-out
: File to write conformers generated by OMEGA. Gzipped OEBinary is the recommended output format.

4.1.2 Optional Parameters

Execute Options

-param
: The argument for this flag is the name of a file containing control parameters. The control parameter file acts to either replace or augment the command line interface. All parameters necessary for program execution may be provided in the control parameter file, although any command given explicitly on the command line will supersede options found in the parameter file. The application generates a new parameter file containing the full set of execution parameters upon every execution. The name of the parameter file is created by combining the prefix base name with the *'.parm'* extension.

-mpi_np <n>
Specifies the number of processors `n` when the application is run in MPI mode.

-mpi_hostfile <filename>
Specifies the name of the file containing processors configuration. For every host this file should contain a line `host_name slots=n` where `n` is the number of processors on the host.

File Options

-commentEnergy
This flag causes the conformer energy in kcal/mol to be written in the comment field for each conformer. This is particularly useful when writing SD or MOL2 files that will be passed to software that anticipates strain energies written in the comment field. [default = false]

-includeInput
When true this boolean flag will include the input conformer in the output file. This requires that the input file format is a 3D format (eg: SDF, MOL2, OEB, etc) [default = false]

-log
The argument for this flag specifies the name of the log file. The level of detail for logfile information can be altered using the -verbose flag. Output can be directed to the terminal as well with the *-progress log* option. Generation of an output log may be disabled by providing 'nul' or 'null' as an argument. [default = *prefix*.log]

-pendingFile
Filename used for pending molecules file.

-prefix
The argument for this flag defines the prefix to be used for various information and data files generated by OMEGA. Most important among these is the '*omega2.parm*' file which includes a copy of all the parameters used in the OMEGA run. The prefix is also used to generate a default log file name if not explicitly specified with the *-log* flag. [default = omega2].

-progress
Show progress on screen. Options are 'none', 'dots', 'log' and 'percent'. The 'dots' options will displays dots on screen to show molecules completed. The 'log' option will duplicate the log file on screen. The 'percent' option will track progress through the input file. [default = none]

-rotorOffsetCompress
This flag controls the behavior of writing the rotor offset compressed flavor of the OEBinary format. Rotor offset compressed files contain the same information as standard Cartesian OEBinary files, but require a small fraction of the storage space. Optimal compression of OMEGA output can be obtained by writing GZip compressed OEBinary files with the *-rotorOffsetCompress* flag turned on. [default = true]

-sdEnergy
This flag controls the behavior of writing strain energies as SD tags. This flag may be used with either OEBinary or MDL SD files. [default = false]

-status
Filename used for status report file

-verbose
This is a boolean flag that controls the level of detail written to the log file. By default OMEGA will only write minimal information to the log file. Molecule titles and warning messages constitute the bulk of logging at the default level. Verbose logging will cause more information to be written to the log file in order to follow behavior during program execution. In order to have the log shown on the screen, use *-progress log*. [default = false]

-warts
This boolean flag is used to generate unique titles for conformers that reflect their position in an ensemble of

conformations produced by OMEGA. The title given to each conformer will begin with the molecule title taken from the input file, and appended with an underscore and the integer corresponding to the rank order number of the conformer in the final ensemble. [default = false]

3D Construction Parameters

-addfraglib
The argument that follows this flag is one or more fragment files, normally generated by *makefraglib*. These fragments will supplement the built-in fragment library

-buildff
This flag sets the force field used for constructing fragments that are assembled to build an initial model of the input structure. Consult the description of Force Fields (see *Force Fields*) for an explanation of appropriate arguments for this flag. [default = mmff94s_NoEstat]

-canonOrder
This flag can be used to disable the automatic reordering of input molecules to a canonical atom and bond order. In order obtain consistent results from different file formats and different connection table orders, OMEGA reorders the input connection table. This behavior can be turned off using the *-canonOrder* flag, however, the resultant ensembles will likely be inconsistent in their composition. [default = true]

-deleteFixHydrogens
When a fixfile fragment is specified, all possible mappings (matches) of the fragment and the input molecule are considered for positioning the input molecule. This process begins with a substructure search of the fragment in the input molecule. If hydrogens are included on the fixfile fragment, the number of possible matches will grow exponentially with the number of equivalences. Each geminal hydrogen pair on the fixfile produces a non-productive match multiplied by all other non-productive matches. This flag can be used to prevent the explosion of non-productive matches by deleting hydrogens from the fixfile fragment prior to the substructure search. [default = true]

-dielectric
The argument to this flag allows the user to specify the dielectric applied to the Coulomb term of the force field. This flag may only be used with a version of the search force field that includes the Coulomb term, such as the MMFF94 and MMFF94s variants. [default = 1.0]

-exponent
The argument to this flag allows the user to specify the exponent applied to the inverse distance calculation of the Coulomb term of the force field (i.e. $(1/r)^X$ where X is the argument to the *-exponent* flag). This flag may only be used with a version of the search force field that includes the Coulomb term, such as the MMFF94 and MMFF94s variants. The legal values are 1 or 2. [default = 1]

-fixfile
The argument that follows this flag is a molecule file used to specify the coordinates for a substructure of the input molecules. An initial structure is generated for every input molecule, and then a substructure search is performed using the molecule or fragment provided in the fixfile as a query molecule. Every instance of the fixed substructure found in the input molecule up to a predetermined limit (see *maxmatch* and *umatch*) is used to replace the coordinates of the atoms that match the substructure. The input molecule coordinates are aligned relative to the substructure prior to fragment replacement, and then the coordinates are taken from the fixed fragment and assigned to the corresponding atoms of the input molecule. A separate alignment, replacement, and then conformer search is carried out for every matching substructure in the input molecule. Molecules that do not match the fixfile will be sent to the fail file.

-fixrms
Fixfile fragments taken from crystallographic sources may differ in their geometry relative to optimal MMFF geometries. OMEGA attempts to superimpose built structures onto fixfile fragments and, if the geometry differs too greatly, OMEGA considers the superposition a poor match and will fail to build a structure using the fixfile.

This flag can be used to loosen the default RMS superposition criteria to allow suboptimal superpositions to succeed in spite of the poor geometric complementarity. [default = 0.15]

-fixsmarts
Another way to specify a fixed portion of a molecule. The SMARTS pattern is used to fix a portion of the *-fixfile* primarily, or fix or portion of the input molecule if *-fromCT* is set to `false` secondarily. Molecules that do not match will be sent to the fail file.

-fromCT
This boolean flag determines whether OMEGA should generate an initial set of 3D coordinates using only the connection table of the input molecule. Initial model generation is always necessary for molecule file formats devoid of coordinates (i.e SMILES). Bond lengths and angles taken from molecule files containing coordinates may be retained by setting this flag to false. *-enumNitrogen false* must also be set for identical stereochemistry to be guaranteed in the output. [default = true]

-maxmatch
This flag is used to limit the number of fixfile substructure matches in the input molecule. Each match will result in replacement of the matching substructure with coordinates taken from the fixfile fragment. The number of matches may need to be limited using this parameter for a substructure where many matches are possible. [default = 1]

-setfraglib
The argument that follows this flag is an OEBinary molecule file containing the coordinates of pre-built acyclic fragments, and multiple conformations of cyclic systems. Starting with OMEGA2.2, this flag is no longer required, as a default fragment library has been built into the program. Although OMEGA can generate fragment 3D geometries on the fly, building them in advance speeds execution significantly. Multiple fragment files may be provided as a comma delimited list of filenames. Only the first matching fragment will be used even if the fragment occurs multiple times. The fragment file(s) chosen should be constructed using a force field corresponding to the one being used for the torsion search. For example, the MMFF94s variant of the Merck Molecular Force Field should be used both for fragment construction and torsion search. Normally this file is built using the *makefraglib* auxiliary program.

-strictfrags
This flag sets how OMEGA generates fragments that are not in the fraglib. The default is a faster and less rigorous fragment generation than the makefraglib program, whereas a 'true' value will require OMEGA to be identical to the makefraglib program. The most noticeable difference is the time limit for fragment generation, which is increased from 30 seconds to 300 seconds with *-strictfrags* set to `true`. [default = false]

-strictatomtyping
Use strict atom typing for MMFF94 or allow 'close enough' atom typing. A *true* setting will fail any molecule that contains an atom type that does not have specified parameters. A *false* setting will allow parameters from a similar atom type to be used. [default = true]

-umatch
The *-umatch* boolean flag determines whether only the unique substructure matches of the fixfile are used for coordinate replacement. A unique substructure match is defined as a match that does not cover the identical set of target atoms as any other substructure match in a set. For example, a benzene substructure will match a benzene ring 12 times. Only one substructure match constitutes a unique match, while the other 11 matches are duplicates. If the flag is set to false then all possible substructure matches may be used for coordinate replacement. This behavior is usually unnecessary as non-unique matches will frequently lead to duplication. [default = true]

Structure Enumeration

-enumNitrogen
The *-enumNitrogen* string flag controls the behavior of OMEGA with respect to enumeration of non-planar nitrogens. Any nitrogen with pyramidal geometry in the initial model of the input molecule, and having no more

than two ring bonds is considered by OMEGA to be 'invertible'. OMEGA will enumerate all possible puckers if the *-enumNitrogens* flag is set to `true`. OMEGA will only enumerate unspecified invertible nitrogens if the flag is set to *unspecified*. [default = true]

-enumRing
The *-enumRing* boolean flag controls the behavior of OMEGA with respect to ring conformations. If this flag is set to true, OMEGA will generate all possible combinations of all ring conformations in a molecule. Ring systems with only a single conformation will be replaced with a conformation taken from a fragment file, or generated on the fly by OMEGA. If this flag is set to false then no ring conformer enumeration or replacement will occur. Initial geometries provided in by an input file (see *-fromCT*) may therefore be preserved by setting *-enumRing* to false as well. [default = true]

-sampleHydrogens
Sets whether hydrogens will be sampled. This option enables sampling of hydrogen locations for -OH, -SH, and amines. [default = false]

Torsion Driving Parameters

-addtorlib
Takes a filename as a parameter. Torsion rules in the file are placed above any previous torsion rules and therefore are matched first. (see *Torsion Library Format*)

-erange
The *-erange* flag sets the energy cutoff used as an accept or reject criteria for conformers depending on the number of rotatable bonds in the structure. Any conformer that has a calculated strain energy less than the sum of the energy window and the energy of the global minimum conformer will be accepted. Conformers with strain energies above this threshold are rejected. The energy range is given as a comma separated list of values that correspond to the *-rangeIncrement* parameter. For example, *-erange "5.0, 10.0, 15.0, 20.0"* used with *-rangeIncrement 3* sets the energy window to 5.0 Kcal/mol for structures with zero to two rotatable bonds, 10.0 Kcal/mol for structures with three to five rotatable bonds, and so on. The energy window for structures with more rotors than the highest *-erange* value specified will be taken as the highest specified value.

-ewindow
The *-ewindow* flag sets the energy window in kcal/mol used as an accept or reject criteria for conformers. Any conformer that has a calculated strain energy less than the sum of the energy window and the energy of the global minimum conformer will be accepted. Conformers with strain energies above this threshold are rejected. [default = 10.0]

-maxConfRange
This string argument to this flag allows the user the specify the maximum number of conformers to be output for a structure based on the number of rotatable bonds in the structure. For example, *-maxConfRange "100,200"* used with *-rangeIncrement 5* will cause OMEGA to output 100 conformers for structures with zero to 4 rotors, and 200 conformers for all structures with more than 4 rotors.

-maxconfs
The *-maxconfs* flag sets the maximum number of conformations to be generated. Conformers are assembled in energy sorted order. As a special case, setting *-maxconfs 0* will result in OMEGA skipping the duplicate removal step and it will write all generated conformers to the output file. Note that this implies *-rms 0* is also used. [default = 200]

-maxrot
The *-maxrot* flag sets the maximum number of rotatable bonds cutoff. Molecules that have equal to or fewer rotors than the *-maxrot* cutoff will be processed by OMEGA. OMEGA will not search for conformers of molecules that have more rotors than the *-maxrot* cutoff. By default, OMEGA does not apply a number of rotatable bonds cutoff. Instead, a desired cutoff must be supplied by the user. [default = -1]

-maxtime
This flag limits the amount of time (in seconds) spent generating conformers for each molecule. [default = 120.0]

-rangeIncrement
The *-rangeIncrement* is used to control the number of rotatable bonds range used with the *-maxConfRange*, *-rmsrange*, and *-erange* flags. The preceding flags are used to control the maximum number of conformers, RMS cutoff, and energy windows used that are dependent on the number of rotors in a given structure. [default = 5]

-rms
The *-rms* flag sets the minimum Root Mean Square (RMS) Cartesian distance below which two conformers are duplicates. The RMS calculation is performed after superposition such that the true minimum distance between conformers is calculated. Lowering the *-rms* value may cause OMEGA to generate ensembles that contain more representative conformers of a similar shape. Higher *-rms* values may result in smaller, yet possibly more shape diverse ensembles. [default = 0.5]

-rmsrange
This string argument to this flag allows the user the specify the RMS cutoff used for duplicate conformer removal based on the number of rotatable bonds in the structure. For example, *-rmsrange "0.8,1.0"* used with *-rangeIncrement 5* will cause OMEGA to use an RMS cutoff value of 0.8 for structures with zero to 4 rotors, and an RMS cutoff value of 1.0 for all structures with more than 4 rotatable bonds.

-searchff
This flag sets the force field used to calculate strain energies of conformers generated during a torsion search. Consult the description of Force Fields (see *Force Fields*) for an explanation of appropriate arguments for this flag. [default = mmff94s_NoEstat]

-settorlib
The *-settorlib* flag is used to specify the file name of the file containing rules used in resolution control of the torsion driving part of conformer generation. Refer to the section describing the format of the torsion library (see *Torsion Library Format*). If no torsion library is provided then OMEGA will use an internally stored copy of the default torsion library.

Stereo Parameters

-flipper
Creates an ensemble or stereoisomers before passing the molecules for conformer generation. A value of '*true*' will flip all unspecified stereo centers and a value of '*force*' will flip all stereo centers. [default = false]

-flipper_maxcenters
Sets the maximum number of stereocenters than can be flipped exhaustively. [default = 12]

-flipper_warts
Add wart to title of each stereoisomer. [default = false]

-strictstereo
Requires all stereo centers to be specified. Molecules sent to conformer generation with unspecified stereo will fail. [default = true]

General

-strict
Convenience flag for setting *-strictstereo*, *-strictatomtyping*, and *-strictfrags* at once. This setting will override any individual settings for these flags.

4.1.3 Example Executions

This section has a series of example OMEGA command-line executions. Each example is followed by a brief description of its behavior. Sample data files can be found in the data directory.

Basic Commands

```
prompt> omega2 drugs.smi drugs.oeb.gz
prompt> omega2 -in drugs.smi -out drugs.oeb.gz
```

These two commands will yield identical results. These execute OMEGA with the default parameters. The file *drugs.smi* is opened in SMILES format for input, and the output is written to the file *drugs.oeb.gz* in gzipped OEBinary format.

Using Parameters and Param Files

```
prompt> omega2 -in drugs.smi -out drugs.oeb.gz -maxconfs 1
```

The parameter *-maxconfs* can be used to specify the maximum number of conformers for each output molecule. This command will generate a single low energy conformer for every molecule in drugs.smi.

```
prompt> omega2 -in drugs.smi -out drugs.oeb.gz -param myparameters
```

This command is the same as the Basic Commands above except for the *-param* flag. It executes OMEGA with the parameters found in the *myparameters* file. The file *drugs.smi* is opened in SMILES format for input, and the output is written to the file *drugs.oeb.gz* in gzipped OEBinary format.

```
prompt> omega2 -param myparameters drugs.smi drugs.oeb.gz
prompt> omega2 drugs.smi drugs.oeb.gz -param myparameters
```

The first of these two commands will yield exactly the same results as the example above. *drugs.smi* will be mapped to the *-in* flag and *drugs.oeb.gz* will be mapped to the *-out* flag begin the second to last and last command-line arguments respectively. Unfortunately, the second of these two commands, will fail to parse because the implicit input and output arguments are not the final two arguments in the list.

```
prompt> omega2 -in drugs.smi -out drugs_maxconfs600.oeb.gz -param myparameters -maxconfs 600
```

Again, this is a very similar command. It executes OMEGA using the parameters in the *myparameters* file, except the *-maxconfs* parameter is over-ridden with the 600 from the command line. The command-line *-maxconfs* parameter would take precedence over the value in the parameter file independent of the order of flags on the command line.

Advanced Commands

```
prompt> DBQuery "barbiturate" | omega2 -in .sdf -out .oeb.gz | vida2 .oeb.gz
```

This execution assumes that a process called "DBQuery" can be called with the parameter "barbiturate" and return a stream of molecules in MDL's .sdf format. This output is piped into OMEGA, which interprets the format correctly and generates multiconformer molecules using the default parameters. OMEGA writes the output to *std::cout* in gzipped OEBinary format, which is read by OpenEye's VIDA molecular viewer.

```
prompt> omega2 -in drugs.mol2 -out drugs.oeb.gz -fromCT true
```

The *-fromCT* true flag will cause OMEGA to ignore the input conformations in the *drugs.mol2* file. An initial conformation will be generated by a distance-bounds algorithm from the connection-table of the molecules in the input file. Be aware that the default value of *-fromCT* is true, however it is listed explicitly here for emphasis.

```
prompt> omega2 -in drugs.smi -out drugs.oeb.gz -log null
```

The *-log* flag normally allows specification of the log file's name. However, *nul* and *null* are reserved names which indicate that no log file will be written (this includes failure modes).

4.2 Further specifications

4.2.1 Force Fields

OMEGA provides the facility for users to select one of a number of pre-defined force fields. The force field selected may be different for model construction and torsion search. The ability to select a force field provides a mechanism for task specificity. Some force fields may be more appropriate for solution phase ensemble generation, whilst others may excel for bioactive conformer reproduction. The following pre-defined force fields may be used as arguments to the *-buildff* and *-searchff* flags.

- *mmff* Exact reproduction of the published Merck Molecular Force Field (MMFF) with addition atom types parameterized using the same algorithm.
- *mmff_NoEstat* This force field variant includes all MMFF terms except Coulomb interactions.
- *mmff_Trunc* This force field variant excludes both Coulomb interactions and the attractive part of Van der Waals interactions. All other components of the MMFF force field are calculated according to literature specifications.
- *mmff94s* Exact reproduction of the 94s variant of the Merck Molecular force Field (MMFF94s) with addition atom types parameterized using the same algorithm.
- *mmff94s_NoEstat* This force field variant includes all MMFF94s terms except Coulomb interactions.
- *mmff94s_Trunc* This force field variant excludes both Coulomb interactions and the attractive part of Van der Waals interactions. All other components of the MMFF94s force field are calculated according to literature specifications.

4.2.2 Torsion Library Format

A file of alternate torsion rules may be specified with the *-settorlib* command. OMEGA will match only the first rule found for a torsion angle, and discontinue matching alternate possible rules. Thus, a correctly ordered torsion file will be arranged with the most specific patterns appearing at the top of the file, and more general patterns appearing toward the end. Simple torsion rules are composed of a single SMARTS pattern containing at least four atom expressions, followed by a listing of the torsion angles that OMEGA will sample. Each reference atom in the SMARTS pattern that is used to define the torsion angle being sampled must have a map index (numbered 1 through 4) specified that indicates the ordering of the atoms in the torsion angle. The pattern must appear all on a single line, with a carriage return separating one rule from the next. Comments in the file must be preceded with a # character. The following is an example of a simple torsion rule.

```
#methyl ester pattern
[O:1]=[C:2]-[O:3][CH3:4] 0
```

More advanced rules may be included that alter the energy calculation for particular torsion angles. In these types of torsion rules, a SMARTS pattern with associated map indices is still used to define the molecular environment in which the rule is to be applied, but the sampled values appear on subsequent lines with one torsion angle per line. The first number per line indicates the torsion angle, in degrees, that OMEGA must sample. If a second number follows a torsion angle on the same line, the value is added to the total energy computed for that conformer. The following is an example of an advanced torsion rule.

```
#experimental structure test
O=[C:1][NX3H:2][c:3]([cH,nH0])[nH:4]
0
180 10.0
<end>
```

CHAPTER

FIVE

MAKEFRAGLIB: CREATING A FRAGMENT LIBRARY

makefraglib identifies, extracts, and generates conformers of molecular fragments. The model builder in OMEGA fragments input molecules, retrieves corresponding fragment conformers, and assembles them into a three dimensional structure. Pregenerating fragment conformers using *makefraglib* accelerates the model building procedure.

5.1 *makefraglib* Theory

The model builder in OMEGA attempts to capture all of the relevant bond length and angles, and ring conformations for an input molecule. Fragment assembly allows for a 'divide and conquer' approach to model building. Much of the relevant information for building a molecular models is contained in carefully chosen fragments. *makefraglib* uses distance constraints and geometry optimization to generate fragment conformations. Only a single conformation is stored for acyclic fragments, while all possible unique ring conformers are retained within user defined limits. Libraries created with *makefraglib* can then be provided to OMEGA using the *-setfraglib* or *-addfraglib* flags.

5.2 *makefraglib* Usage

makefraglib can be used from the command-line to generate ring templates for use with OMEGA starting from only a collection of molecules. The output file from *makefraglib* can then be passed directly into OMEGA using the *-setfraglib* or *-addfraglib* flag. Although fragment libraries may be concatenated using the Unix 'cat' command, this is not strictly necessary as multiple fragment libraries may be specified as one argument to the flag.

5.2.1 Required Parameters

`-in`
: File containing one or more molecules from which fragments will be generated

`-out`
: File to write fragments generated by *makefraglib*. OEBinary format is required and Gzipped OEBinary is recommended.

5.2.2 Optional Parameters

Execute Options

-param
The argument for this flag is the name of a file containing control parameters. The control parameter file acts to either replace or augment the command line interface. All parameters necessary for program execution may be provided in the control parameter file, although any command given explicitly on the command line will supersede options found in the parameter file. The application generates a new parameter file containing the full set of execution parameters upon every execution. The name of the parameter file is created by combining the prefix base name with the *'.parm'* extension.

-mpi_np <n>
Specifies the number of processors `n` when the application is run in MPI mode.

-mpi_hostfile <filename>
Specifies the name of the file containing processors configuration. For every host this file should contain a line `host_name slots=n` where `n` is the number of processors on the host.

File Options

-log
The argument for this flag specifies the name of the log file. The level of detail for logfile information can be altered using the *-verbose* flag. Output can be directed to the terminal instead of a file by giving a hyphen '-' as the argument to the flag instead of a filename. Generation of an output log may be disabled by providing 'nul' or 'null' as an argument. [default = *prefix*.log]

-prefix
The argument for this flag defines the prefix to be used for various information and data files generated by OMEGA. Most important among these is the *'makefraglib.parm'* file which includes a copy of all the parameters used in the makefraglib run. The prefix is also used to generate a default log file name if not explicitly specified with the *-log* flag. [default = makefraglib].

-progress
Show progress on screen. Options are 'none', 'dots', 'log' and 'percent'. The 'dots' options will displays dots on screen to show molecules completed. The 'log' option will duplicate the log file on screen. The 'percent' option will track progress through the input file. [default = none]

-skip
Existing fragment library to avoid duplicating

-verbose
This is a boolean flag that controls the level of detail written to the log file. [default = false]

3D Construction Parameters

-buildff
This flag sets the force field used for constructing fragments that are assembled to build an initial model of the input structure. Consult the description of Force Fields (see *Force Fields*) for an explanation of appropriate arguments for this flag. [default = mmff94s_NoEstat]

-ewindow
The *-ewindow* flag sets the energy window used as an accept or reject criteria for fragments. Any fragment that has a calculated strain energy less than the sum of the energy window and the energy of the global minimum fragment will be accepted. Conformers with strain energies above this threshold are rejected. [default = 4.0]

-fromCT
This boolean flag determines whether *makefraglib* should generate an initial set of 3D coordinates using only the connection table of the input molecule. Initial model generation is always necessary for molecule file formats devoid of coordinates (i.e SMILES). Bond lengths and angles taken from molecule files containing coordinates may be retained by setting this flag to false. [default = true]

-rms
The *-rms* flag sets the minimum Root Mean Square (RMS) Cartesian distance below which two fragments are duplicates. The RMS calculation is performed after superposition such that the true minimum distance between fragments is calculated. Lowering the *-rms* value may cause *makefraglib* to generate ensembles that contain more fragment conformers of a similar shape. Higher *-rms* values may result in smaller, yet possibly more shape diverse ensembles. [default = 0.1]

-startfact
Factor for determining number of random starting geometries when generating fragments.

5.2.3 Example Executions

The following section shows several common command-line executions of *makefraglib*. Each example is followed by an explanation of what the program will do. Sample data files can be found in the data directory.

```
prompt> makefraglib -in drugs.smi -out fraglib.oeb
```

All of the molecules in *drugs.smi* will be processed. The set of all unique fragments (according to OMEGA fragmentation rules) from all of the molecules in *drugs.smi* will be collected.

```
prompt> makefraglib -in vendor.smi -out new_fraglib.oeb -skip fraglib.oeb -ewindow 5.0
```

Conformers will be generated for all of the unique fragments found in the user supplied file *vendor.smi* that are not present in *fraglib.oeb*. An energy window of 5 Kcal above the global minimum conformer will be used to accept or reject conformations of flexible ring systems.

CHAPTER

SIX

FLIPPER: ENUMERATING STEREOCENTERS

6.1 *flipper* Theory

The OMEGA distribution includes *flipper*, a utility program for enumerating stereocenters in a molecule. We have taken the philosophy that conformer generation and stereochemistry enumeration are related but distinct problems. Thus we provide *flipper* as part of the OMEGA product, but not as part of the same executable.

The current version of *flipper* can handle enumeration of R/S stereochemistry (including N and S stereochemistry) as well as *cis/trans* stereochemistry. *flipper* uses graph algorithms to determine which atoms are stereocenters. If a stereocenter has a designated stereochemistry, by default flipper does not change the stereochemistry. However, if a stereocenter does not have a specified stereochemistry, *flipper* will enumerate both stereochemistry states of the stereocenter.

Many common file formats are limited in their ability to fully express stereochemistry without resorting to Cartesian coordinates. This can lead to ambiguity in whether a particular conformer is supposed to represent a racemic mixture or an isolated stereoisomer. SD file format is better than most formats. It allows designation of R, S, or racemic on each tetrahedral stereocenter independent of the particular conformation. However, SD format is not as robust for *cis/trans* stereochemistry. Isomeric SMILES is the only common file format which allows complete specification of R/S and *cis/trans* stereochemistry. In fact, although *flipper* does not currently support higher orders of stereochemistry, the isomeric SMILES format is capable of handling them. For this reason, *flipper* output is limited to SD format (.sdf or .mol) and isomeric SMILES format (.ism). OpenEye OEBinary format, like isomeric SMILES, is capable of fully specifying stereochemistry, however, isomeric SMILES is more efficient for compounds without any data beyond the title and connection table.

6.1.1 Input dimensionality

flipper is designed for use prior to 3D coordinate generation. So 2D or 1D (SMILES) input is normally expected. Input files containing 3D coordinates are handled, but the 3D coordinates are ignored and not transferred to the output.

6.2 *flipper* Usage

flipper is a simple command-line utility program. The *flipper* process can also be done with OMEGA using the *-flipper* flag.

6.2.1 Required Parameters

-in
File containing one or more molecules for which stereoisomers will be generated

-out
This required parameter indicates the output molecule file. The output file format is limited to OEB, isomeric SMILES (*.ism* NOT *.smi*) or SD format since these are the only formats which fully specify stereocenters without the need for three-dimensional coordinates.

6.2.2 Other options

-enumNitrogen
The *-enumNitrogen* string flag controls the behavior of FLIPPER with respect to enumeration of non-planar nitrogens. Any nitrogen with pyramidal geometry in the initial model of the input molecule, and having no more than two ring bonds is considered by OMEGA to be 'invertible'. FLIPPER will enumerate all possible puckers if the *-enumNitrogens* flag is set to `true`. [default = false]

-forceflip
This parameter forces *flipper* to modify all of the stereocenters in a molecule. If *-forceflip* is false (the default), *flipper* only enumerates stereocenters which do not already have a specified stereochemistry. [default = false]

-maxcenters
Obviously, the number of molecules generated by enumerating the stereocenters is 2^N, where *N* is the number of stereocenters. In some instances, this may be larger than is desired. The *-maxcenters* parameter indicates the maximum number of stereocenters which will be fully enumerated. If a molecule has more than *-maxcenters* stereocenters, *flipper* will randomly enumerate *2^(maxcenters)* instances from the full set of potential isomers. [default = 12]

-warts
Add wart to title of each stereoisomer. [default = false]

6.2.3 Example Executions

Below are three example *flipper* executions. Each example is followed by a brief description of the parameter's effects. Sample data files can be found in the data directory.

```
prompt> flipper -in drugs.smi -out enumerated.ism
```

This execution will examine each stereocenter in *drugs.smi*. If the stereocenter does not have a specified stereochemistry, *flipper* will generate molecules with each of the stereochemistry states at each stereocenter.

```
prompt> flipper -in drugs.smi -out enumerated_forceflip.ism -forceflip true
```

This execution will differ from the first in that **all** stereocenters will be enumerated, regardless of whether the stereochemistry is specified in the input file.

```
prompt> flipper -in drugs.smi -out enumerated_maxcenters6.ism -maxcenters 6
```

In this execution, *flipper* will exhaustively enumerate all unspecified stereocenters in molecules with up to 6 stereocenters. For molecules with more than 6 stereocenters, a random set of 64 isomers will be generated from the larger set of potential stereoisomers.

CHAPTER

SEVEN

OEB2SDCONF

7.1 *oeb2sdconf* Usage

oeb2sdconf is a utility that allows OMEGA generated conformers to be used with the MOE program. Please contact Chemical Computing Group support if you require assistance with this utility.

CHAPTER

EIGHT

FILTERING THEORY

8.1 Introduction

Filtering attempts to eliminate inappropriate or undesirable compounds from a large set before beginning to use them in modelling studies. The goal is to remove all of the compounds that should not be suggested to a medicinal chemist as a potential hit. This exercise is obviously case dependent, depending on ease of the assay, intended target, personal bias of the modeller & medicinal chemist, strengths of the company, etc.

To match this need, FILTER's default filter encapsulates many of the standard filtering principles, such as removal of unstable, reactive, and toxic moieties. In addition, FILTER allows the customization of the filtering criteria to fit specific needs.

The criteria for passing or failing a given molecule fall into three categories.

- Physical properties
  - Molecular weight
  - Topological polar surface area (TPSA)
  - logP
  - Bioavailability
- Atomic and functional group content
  - Absolute and relative content of heteroatoms
  - Limits on a very wide variety of functional groups
- Molecular graph topology
  - Number and size of ring systems
  - Flexibility of the molecule
  - Size and shape of non-ring chains

All of the data FILTER generates in filtering molecules can be written to a tab-separated file for easy import into a spreadsheet. This function allows for combining the values dynamically for a variety of purposes, including, but not limited to, determining which filter values best fit each project's needs.

8.1.1 History

When OpenEye's work on filtering technology began in 2000, it was designed simply to remove compounds with reactive or otherwise undesirable functional groups. Over the years, the understanding of lead-like and drug-like

compound selection has advanced. In addition, with the publication of Lipinski's "Rule of 5" [Lipinski-1997], more and more pharmacokinetic properties have been pushed earlier into the virtual screening process.

In addition to providing basic functional group selection, the technology is a one-stop database preparation tool aimed at generating databases suitable for high-throughput virtual screening.

- Cheminformatics quality-control
  - Valence-state validation
  - Aromaticity perception
  - Implicit hydrogen perception
  - Bond-order perception
- Database preparation
  - Setting pKa states
  - Applying normalizations (tautomers & dative or hypervalent states)
- Compound selection
  - Physical properties (see *above*)
  - Assay counter-indicators (aggregators and dyes)
  - PK ([Martin-2005], [Veber-2002], [Egan-2000], [Lipinski-1997])

Finally, it should be pointed out that in the virtual screening world, time is of the essence. Algorithms for preliminary database preparation should not take large amounts of time. Because of this, all the calculations included in FILTER are 2D or graph-based algorithms. While this does occasionally limit the technology, it allows for the delivery of a product that is appropriate for the task of virtual-screening database preparation.

8.1.2 The Rant!

Nearly every computational tool used in early drug discovery yields statistically predictive, rather than absolutely definitive results. In nearly every case, prudence demands that one consider the causes of false-positives and false-negatives and make an attempt to optimize the area under the receiver-operator curve (ROC) for the computational tool. However, there are well known methods for improving statistical predictions of this nature that are independent of the absolute false-positive and false-negative rates. These methods include filtering the population to which a test will be applied. By applying a test to smaller populations that only contain molecules appropriate for the specific application at hand, the negative impact of the false-positive rate on the predictive results can be dramatically improved.

A familiar example from the medical world will serve to illustrate this principle. Assume we have a test for the presence of the new *foo virus* which has an exceptional ROC curve with false-positive and false-negative values (1/1,000 and 1/1,000 respectively). Let us assume that the *foo-syndrome*, caused by the *foo virus*, effects 1 person in 20,000. If we gave this test to 100,000 people from the general population, we would expect 5 to actually have the *foo syndrome*. With this test, there is only a 0.05% percent chance that any of them would not be detected (i.e. be a false-negative). However, we would expect there to be 100 false positive test results. Thus of the 105 total positive test results, only 4.8% would actually have the *foo syndrome* (positive predictive value = 4.8%).

Table 8.1: Confusion table for the unfiltered foo virus test (prevalence 1 in 20,000)

	Actual Positive	**Actual Negative**	
Predicted Positive	True Positives = 5	False Positives = 100	Positive Predictive Value = 4.8%
Predicted Negative	False Negatives = 0	True Negatives = 99,895	

Alternatively, we could start by using very simple screening before applying the test. We first eliminate people who do not have any risk factors for contracting the *foo virus*. Next we may eliminate people whose blood is incompatible with

the test for the *foo virus*. Further, we may want to eliminate people who acknowledge that they will refuse treatment for the *foo virus* even if we determine that they do have it. By these admittedly simple screens, we apply the test for the *foo virus* to a much smaller group with a decidedly higher prevalence of the virus. For instance, after the filtering, we may be left with a group of only 1,000 people who have a 1 in 200 chance of having the syndrome. Now, we still have the same 5 people who actually have the disease, but we only expect 1 false positive test. Suddenly, there are 6 total positive tests, and 83% of them actually have the syndrome! This is reflected in a much more reasonable (83%) positive predictive value.

Table 8.2: Confusion table for the filtered foo virus test (prevalence 1 in 200)

	Actual Positive	**Actual Negative**	
Predicted Positive	True Positives = 5	False Positives = 1	Positive Predictive Value = 83%
Predicted Negative	False Negatives = 0	True Negatives = 994	

Bringing the discussion back to drug design. If we have a ligand-based design tool such as ROCS, we can imagine that the receiver-operator curve may have a false positive rate as low as 1 in 10,000. For this exercise, lets assume no false negatives. When using that to identify 50 inhibitors from a database of 2.5 million available compounds, we'd identify 300 potential inhibitors, and 5 out of every 6 of these would be a false positive (positive predictive value of 17%)! If we first run filter and eliminate 65% of the 2.5 million compounds, this leaves us with 875,000 compounds to push through ROCS. There will be about 88 false positives to go with the 50 true positives and the positive predictive value will increase over two-fold with relatively little work.

Table 8.3: Confusion table for the unfiltered ROCS virtual screen

	Actual Positive	**Actual Negative**	
Predicted Positive	True Positives = 50	False Positives = 250	Positive Predictive Value = 17%
Predicted Negative	False Negatives = 0	True Negatives = 2,499,700	

Table 8.4: Confusion table for the filtered ROCS virtual screen

	Actual Positive	**Actual Negative**	
Predicted Positive	True Positives = 50	False Positives = 88	Positive Predictive Value = 36%
Predicted Negative	False Negatives = 0	True Negatives = 874,862	

8.1.3 Filtering Principles

The same principle of increasing positive predictive value by removing obvious true negatives applies to screening for lead candidates, regardless of whether it is virtual screening or high-throughput screening. While both are reasonable screens, each can be plagued by very low positive-predictive values (despite low false-positive rates), particularly when applied to all available compounds, or large virtual libraries. Simple filtering techniques focus the set of compounds passed on to more computationally intensive screening methods.

The first approach to consider is filtering based on *functional groups*. Generally speaking, there are toxic and reactive functional groups that you simply do not want to consider (alkyl-bromides, metals etc). There are also functional groups that are not strictly forbidden, but are not desired in large quantities. For instance, parafluoro-benzene, or trifluoromethyl have specific purposes, but heavily fluorinated molecules can be eliminated.

Beyond simple functional group filtering, you can consider both simple and complex physical properties which can be used to characterize the kinds of compounds you would like to keep and those you would like to eliminate. These properties attempt to consider "drug-likeness", such as bio-availability, solubility, toxicity, and synthetic accessibility even before the primary high-throughput or virtual screening, which primarily are geared toward detecting potency alone. The best known of the physical property filters is *Lipinski's "rule-of-five"*, which focuses on bioavailability [Lipinski-1997]. However, many other physical properties, such as *solubility*, *atomic content*, *ring structures*, and *surface area ratios* can also be considered. FILTER provides algorithms for calculating many of these properties, and applying them with filters based on literature studies.

Finally, you should eliminate the types of compounds that can be troublesome at later stages. For instance, Shoichet's *aggregating compounds* often produce false positives that can waste enormous resources if they were identified by virtual or high-throughput screening [McGovern-2003] [Seidler-2003] . Similarly, *dyes* can appear to be inhibitors by interfering with colorimetric or fluorometric assays or binding non-specifically to the target protein.

8.1.4 Variations of Filters

Different types of filters are appropriate under different circumstances. Very early in a project, when little or no SAR is available, very strict drug-like filters can be applied. This prevents a project team from spending chemistry resources pursuing difficult compounds that may not be modifiable to introduce appropriate properties. However, when considering compounds for purchase for HTS, different filters can be applied. Oprea, et al, pointed out that the best molecules for initial HTS are smaller and less functionalised than drugs, but with some activity [Oprea-2000]. Therefore, strict lead-like filters can be applied to ensure that hits identified from HTS have sufficient "room" for elaboration into (usually larger and more highly functionalised) leads. However, when SAR suggests that particular compounds or series may yield valuable information, filtering criteria can be loosened, because the secondary screens (QSAR models, similarity to known actives) that are being applied are effective in detecting useful compounds. Reflecting back on the medical analogy, this is the case where an improved primary screen with a dramatically improved false-positive rate (say 1 in 100,000) can be safely applied to a larger population without terrible effects on the positive-predictive value.

FILTER provides the following "light" (lenient) and "heavy" (restrictive) filters:

- BlockBuster
- Lead

The BlockBuster filter is based on 141 best-selling, non-antibiotic, prescription drugs. We designed the physical property portion of the filter so that it passes all of the compounds. The physical property values in this filter are quite good. However, the functional group filters in this filter are probably too restrictive because 141 compounds is not sufficient to span all acceptable functionality.

Note: The original [Oprea-2000] Drug filter is provided as well. However, experience has shown it to be too restrictive. The BlockBuster filter was developed in response to complaints about the Drug filter being too restrictive.

Hint: We recommend the BlockBuster filter, the default, for most purposes. If your project is unusual, or you are unsatisfied with the results we recommend you review the *filter file* for your specific filtering needs.

If you decide to modify the *filter file* the depictions found in the *Functional Group Rules* section can be particularly helpful in determining what functional groups are indicated by each name in the file.

8.1.5 Accumulation of Rules

FILTER contains numerous rules that judge the quality of molecules on many different facets. When examined individually, each of these rules seems quite reasonable and even profitable. However, when each molecule is tested against hundreds of individual filters, the fraction of molecules that pass all the filters can be surprisingly small. Sometimes less than 50% of vendor databases pass the filters. If this is unacceptable we recommend you examine the predicted aggregator, solubility, and Veber filters. In our experience, these are the most common failures. The best method of investigating failures is looking at the filter log.

The BlockBuster filter was adjusted to demonstrate this in a tangible way. For each value, rather than spanning the entire range, its properties were set to cover from the 2.5th percentile to the 97.5th percentile. The differences between the original BlockBuster filter and the adjusted filter are both in reasonable ranges. For instance, the full range of molecular weight for the BlockBuster filter spans 130 to 781, while the 2.5th percentile is 145 and the 97.5th percentile is 570. The remarkable result is that when the reduced filter is used, only 75 of the 141 original molecules pass the filter! This demonstrates how slight changes to many filters can lead to a significant reduction in the number of compounds that pass all of the filters.

Taking the opposite approach of allowing everything to pass can be equally futile. To demonstrate this a filter was designed around the "small molecule drug" file available from the DrugBank website. In order to pass every one of these molecules, including some that would not be acceptable for modern project work, many of the individual filters must be set to unreasonable values. For instance, the molecular weight range is 30 to 1500 and the hetero-atom count range is 1 to 60!

Hint: OpenEye can not magically divine the needs of every project. You should personally inspect the *filter file*. The depictions found in the *Functional Group Rules* section can be particularly helpful in discerning what functional groups are desirable and undesirable.

CHAPTER

NINE

FILTER APPLICATION USAGE

9.1 Command Line Interface

A description of the command line interface can be obtained by executing FILTER with the *--help* option.

```
prompt> filter --help
```

will generate the following output:

```
Help functions:
  filter --help simple      : Get a list of simple parameters
  filter --help all         : Get a complete list of parameters
  filter --help <parameter> : Get detailed help on a parameter
  filter --help html        : Create an html help file for this program
```

9.1.1 Required Parameters

-in
: File containing one or more molecular connection tables from which you would like to remove the non-medicinal compounds.

-out
: File to fill with the molecules that pass all of the specified filters.

Execute Options

-param
: The argument for this flag is the name of a file containing control parameters. The control parameter file acts to either replace or augment the command line interface. All parameters necessary for program execution may be provided in the control parameter file, although any command given explicitly on the command line will supersede options found in the parameter file. FILTER generates a new parameter file containing the full set of execution parameters upon every execution. The name of the parameter file written by FILTER is created by combining the prefix base name with the *'.param'* extension.

Optional Parameters

-filter
: This optional parameter specifies a filter file to be used in place of the default filter. If only simple additions to

the default filters are desired, please see the *-newrule* parameter. The file format for this file is described in the *Filter Files* chapter. There are two special reserved strings for this parameter. If the *-filter* parameter is "lead", then the default lead-based filter will be used. If the *-filter* parameter is "drug", then the default drug-based filter will be used. [default = blockbuster]

-fail
This specifies an optional molecular output file where the molecules that fail to pass the filter will be written. If this parameter is specified, then every molecule from *-in* will either be written to *-out* or to *-fail*. [default = null]

-prefix
For an execution of the FILTER program, three general purpose files are written in addition to the output file specified on the command-line. These file are the "info", "log" and "param" files. Normally, they all begin with the prefix "filter". However, this can be overridden with the *-prefix* parameter. This is particularly useful if you want to run multiple FILTER jobs in the same directory without overwriting files.

-info
Normally, FILTER writes an "info" file during the progress of any execution. At regular intervals during the execution, the info file is updated to reflect the most recent progress. If you are interested in seeing the progress of a run, it is best to either use the *-dots* parameter or look at the info file. Normally, the info file is saved as *filename.info* where "filename" is the prefix specified by the *-prefix* flag. However, one may use the *-info* flag to specify an info file with a separate name. [default = null]

-newrule
This optional parameter can specify a file that contains filter rules to supplement the default filter or the filter specified with the *-filter* parameter. This parameter can be used to extend the functional group list used to filter. New filter rules can also be added directly to the filter file specified with *-filter*. [default = null]

-typecheck
This boolean flag controls whether the valence states of atoms will be checked. This check identifies molecules that are poorly specified, or represent nonsensical chemical states. For example, an oxygen with eight hydrogens attached or a carbon with a +9 formal charge would be rejected. [default = true]

-select
This parameter is a SMARTS string that allows a user to require a specific functional group or substructure be present in all molecules that pass the filter. This feature is particularly useful for identification of reagents for library design. Selection items can also be added directly to the filter file specified with *-filter*. The command-line argument only allows specification of the SMARTS pattern, and exactly one copy of that functional group is required. If a user wants to specify a selection SMARTS with minimum and maximum number of occurrences other than 1, then they can use a SELECT statement inside the filter file. [default = null]

-log
This flag determines where the logging information is written. The logging information includes a listing of the filter used, followed by a one line comment about why each molecule failed, or if it passed, an assessment of the probability that the compound lies in drug-like space.

-table
This flag specifies a file for a tab-separated format table that includes all of the filter data. These data files are ready for import into a spreadsheet program for easy examination. Each column of the table includes one of the filter categories (such as "Molecular Weight") and each row of the table corresponds to a single molecule. The table contains complete entries for all of the molecules in the input file regardless of whether they pass or fail. NOTE: Setting this flag will cause the program to slow down. [default = null]

-tableFlag
This flag specifies that, if a table is being written, any values in the table that would cause a molecule to fail a filter will be flagged with an asterisk. This provides a means of seeing all the filters a molecule might fail, as the log file typically only provides the first failure. [default = false]

-interval
This is the interval at which data is written to the *filter.info* file. The *filter.info* file contains running totals that

are relevant to a FILTER run. Examining the *filter.info* file is the best means of checking on the progress of a FILTER execution. If this flag is 50, then the filter.info file is re-written every 50 molecules. [default = 5000]

-pkanorm
This boolean flag determines whether compounds will be modified to reflect a pH=7.4 model. Notice, this will modify the molecules permanently. [default = true]

-normalize
This flag indicates an optional SMIRKS file. This file should contain the set of reactions you wish to use to normalize the connection table of your molecules. Please note: These reactions are applied before the filtering process and can significantly slow the filtering process. [default = null]

-salt
This flag specifies a molecule file that you consider to be salts. If any molecular entries contain multiple disconnected fragments, then any fragment contained in the "salt" file will be removed. If no file is specified, or if there are multiple disconnected fragments in a molecule record that are not in the salt file, then the first largest remaining fragment will be retained and all others discarded. [default = null]

-sdtag
This boolean flag indicates whether you want the molecular properties used for the filtering run (see *-filter*) to be attached to output molecules as SD tag data. This parameter will only work for *.sdf* or *.oeb* formats. [default = false]

-dots
Boolean flag that determines whether FILTER writes a single dot (.) to the terminal (stdout) for every 500 compounds that are processed.

-unique
This flag accepts the name of a file that contains molecules for FILTER to skip. Only unique molecules that do not appear in this file will be sent to the output. Molecules are checked for uniqueness after FILTER has processed them, therefore a parameter such as -pkanorm could change an input structure from a duplicate to a unique molecule.

9.1.2 Example Executions

This section has a series of example FILTER command-line executions. Each example is followed by a brief description of its behavior.

```
prompt> filter drugs.smi drugs.oeb.gz
prompt> filter -in drugs.smi -out drugs.oeb.gz
```

These two commands will yield identical results. These execute FILTER with the default parameters. The file *drugs.smi* is opened in SMILES format for input, and the output is written to the file *drugs.oeb.gz* in gzipped OEBinary version 2 format.

```
prompt> filter -in drugs.smi -out drugs.sdf -filter myfilter
```

This command is the same as the one above except for the *-filter* flag. It executes FILTER with the parameters found in the *myfilter* file. The file *drugs.smi* is opened in SMILES format for input, and the output is written to the file *drugs.oeb.gz* in gzipped OEBinary version 2 format.

```
prompt> filter -param myparameters drugs.smi drugs.oeb.gz
prompt> filter drugs.smi drugs.oeb.gz -param myparameters
```

The first of these two commands will yield exactly the same results as the example above. The file *drugs.smi* will be mapped to the *-in* flag and *drugs.oeb.gz* will be mapped to the *-out* flag being the second to last and last command-line

arguments respectively. Unfortunately, the second of these two commands, will fail to parse because the implicit input and output arguments are not the final two arguments in the list.

```
prompt> filter -in drugs.smi -out drugs.oeb.gz -table drugs.table
```

This executes FILTER on the file *drugs.smi* and writes molecules that pass the filter to the file *drugs.oeb.gz*. It also writes the all of the filter data to the tab-separated value file *drugs.table*.

```
prompt> cat maybridge.05-1.sdf |filter -in .sdf -out .ism|omega .ism m.oeb.gz
```

This command presumes that you have an SD format file called *maybridge.05-1.sdf*. That file is piped to the FILTER program. The *-in .sdf* flag indicates that FILTER should read .sdf format from std::in. Since no *-filter* flag is specified, the default filter will be used. The *-out .ism* flag indicates FILTER will write isomeric smiles format to std::out. The output would then be piped into OMEGA.

```
prompt> filter -in drugs.smi -out drugs.oeb.gz -select "[N;$(*-a)]"
```

This command will filter the compounds in *drugs.smi* with the default filter and write the output to *drugs.oeb.gz*. It also requires that molecules contain exactly one instance of the aniline substructure defined by the SMARTS pattern *"[N;$(*-a)]"*.

CHAPTER

TEN

FILTER PREPROCESSING

Before the applying any of the *molecular property filters* a preprocessing step occurs that can alter the molecule significantly to fit the criteria needed for most modeling applications. This filtering preprocessing step is a precisely defined series of stages that occur on the molecule in the following order:

1. *Metal Removal*
2. *Salt Removal*
3. *Canonicalization*
4. *pKa Normalization*
5. *Normalization*
6. *Reagent Selection*
7. *Type Checking*

10.1 Metal Removal

Metal removal is the first stage of elemental based filtering. This stage will remove specified metal complexes from the molecule. It will not reject a molecule for having a metal complex. This allows the filter to treat atoms in the counter-ion portion of a molecule separately from the atoms in the primary portion of the molecular record.

For instance, this allows organic molecules that are complexed with silver to be eliminated based on their metal chelate even though they themselves are acceptable while at the same time eliminating a sulphate counter-ion from another molecule before it leads to elimination of the acceptable cationic molecule.

See Also:

Elemental Filters

10.2 Salt Removal

This step deletes all atoms that are not part of the largest connected component of a compound. This effectively eliminates all non-covalently bound portions of the compound.

10.3 Canonicalization

This step canonicalizes the atom and bond order of the parts of the molecule that are left after the previous removal steps. This is necessary to avoid different atom orderings producing slightly different normalizations in the following normalization steps.

See Also:

10.4 pKa Normalization

FILTER has a rule-based system to set the ionization state of input molecules. If pKa normalization is turned on, the molecule is set to its most energetically favorable ionization state for `pH=7.4`. The rule-based nature of this calculation allows it to be very fast. Further, despite being rule-based, this approach takes into account many secondary charge interactions.

While more advanced levels of theory can be found for predicting ionization states, this method is very well suited to virtual-screening database preparation. However, FILTER may not be appropriate for hit-to-lead or lead optimization.

See Also:

10.5 Normalization

In addition to *pKa normalization*, FILTER allows any number of additional molecular normalizations. Since normalizations are usually specific to a particular company or site, FILTER provides the ability for users to input normalizations, such as the nitro tautomer state, but does not provide default implementations.

See Also:

10.6 Reagent Selection

Reagent selection for small linear library synthesis or large combinatorial library synthesis is still a necessary task at many pharmaceutical companies. For a user hoping to identify a set of acyl-halide reagents, they can specify a selection parameter to require that each compound have exactly one acyl-halide. In addition they might want to modify the filter to exclude functional groups (such as primary amines) that may be acceptable for typical lead-like molecules, but are not acceptable for the specific reagent the user has in mind.

Therefore, the selection parameter is the reverse of a filtering parameter. The molecule must *include* the given substructure in order to pass the filter.

See Also:

The *select parameter* in filter files.

10.7 Type Checking

This checks the valence state and formal charge of the entire molecule. The check identifies molecules that are poorly specified, or represent nonsensical chemical states, often from corrupt input data. For example, an oxygen with eight hydrogens attached or a carbon with a +9 formal charge would be rejected.

See Also:

CHAPTER

ELEVEN

MOLECULAR PROPERTIES AND PREDICTORS

FILTER provides a range of properties to predictors to be used as molecular filters. Molecular properties are distinct physical properties that can be measured such as the following:

- *Structural and Chemical Features*
- *Functional Groups*

There have been several attempts at developing fast-approximate QSAR models for bioavailability that have been published. The first of these was *Lipinski's work* ([Lipinski-1997]) and it has been followed by work at Pharmacopia [Egan-2000], Abbott [Martin-2005], and GSK [Veber-2002]. The simplest and probably most trusted is the work of *LogP* and *PSA* used by Egan. The most recent work by Martin, the Abbott Bioavailability Score (ABS) appears to be a refinement of the first generation models and is designed specifically to categorize a molecule's probability of having a bioavailability>10% in rats.

See Also:

The *Pharmacokinetic Predictors* section in the Filter Files chapter.

11.1 Structural and Chemical Features

There are a number of important structural and chemical features of molecules that it is desirable to limit for virtual screening. The following simple measures are provided as filterable properties:

- Molecular weight
- Ring count
- Ring-system size
- Size of non-ring structures
- Length of unbranched chains
- Hetero-atom fraction
- Halide fraction
- Formal charges
- Rotatable bonds

It also includes slightly more complex algorithms for hydrogen-bond donors and acceptors as well as chiral centers.

See Also:

The *Basic Properties* section in the Filter Files chapter.

11.2 Functional Groups

Functional group removal remains at the heart of the filter algorithm. Functional groups fall into several categories including:

- Reactive or labile groups
- Undesirable groups
- Generally acceptable groups
- Protecting groups
- User-derived groups

While reasonable defaults are provided for each functional group in all of the above categories, it is unusual to find an experienced computational or medicinal chemist who agrees with all of the default values. Examining the filter file at least once is strongly encouraged. Developing a custom filter to fit a desired design is also strongly encouraged. The filter files were designed as basic guides.

See Also:

The filter files contain only functional group names. If there is any confusion regarding the definitions, please refer to the *Functional Group Rules* section for complete descriptions.

11.2.1 Dyes

Years ago, many high-throughput assays could be interfered with by colored molecules. While assay technology has continued to advance and often this is not a problem, it is recognized that most molecules that are dyes are not the type of molecules that are commonly carried forward in lead-development projects. Therefore, a pattern-based filter for dye molecules is included. While these patterns occasionally identify molecules that are considered acceptable by some users, in general, they identify molecules that the majority of chemists would rather not see at the top of their virtual-screening hitlists.

11.3 LogP

The XLogP algorithm ([Wang-1997]) is provided because its atom-type contribution allows calculation of the XLogP contribution of any fragment in a molecule and allows minimal corrections in a simple additive form to calculate the LogP of any molecule made from combinations of fragments. Further, although the method contains many many free parameters, its simple linear form allows for ready interpretation of the model and most of the parameters in the model make rational sense.

Unfortunately, the original algorithm is difficult to implement as published. First, the internal-hydrogen bond term was calculated using a single 3D conformation. It was found that this was both arbitrary and unnecessary. This arbitrary 3D calculation has been replaced with a 2D approach to recognize common internal-hydrogen bonds. In tests, this 2D method worked comparably to the published 3D algorithm. Next, the training set had a few subtle atom-type inconsistencies.

Both of these problems were corrected and refit to the original XLogP training data. This implementation gives results that are quite similar to the original XLogP algorithm, so it is called OEXLogP to distinguish it from the original method.

See Also:

- The *XLogP parameter* in the Filter Files chapter.

11.4 LogS

The work of [Yalkowsky-1980] at Arizona has resulted in what is now called the "generalized solvation equation." It states that the solubility of a compound can be broken into two steps, first the melting for the pure solid to pure liquid and second, the phase transfer from pure liquid into water. For many small organic molecules, this second step is somewhat related to LogP.

Because of this relation we choose to explore the use of the XLogP atom-types in solubility prediction. The expectation was that this might provide an approximate though robust and fast method for calculating solubility. We fit the XLogP atom-types to a training set of nearly 1000 public solubilities. From this we derived a linear model for solubility. The model is extremely fast and is useful for classifying compounds as insoluble, poorly soluble, slightly soluble, moderately, soluble or very soluble. The model is notable for the difficulty it has predicting solubilities for compounds with ionizable groups. Further, it is not suitable for the PK predictions that come late in a project. However, it is useful for eliminating compounds with severe solubility problems early in the virtual-screening process.

See Also:

The *solubility parameter* in the Filter Files chapter.

11.5 Polar Surface Area

Topological polar-surface area (TPSA) is based on the algorithm developed by Ertl *et al* [Ertl-2000]. In Ertl's publication, use of TPSA both with and without accounting for phosphorus and sulfur surface-area is reported. However, evidence shows that in most PK applications one is better off not counting the contributions of phosphorus and sulfur atoms toward the total TPSA for a molecule. This implementation of TPSA allows either inclusion or exclusion of phosphorus and sulfur surface area with the default being to not include it.

See Also:

Including phosphorus and sulfur surface area:

- The `PSA_USE_SandP` parameter in the *filter file*

Warning: TPSA values are mildly sensitive to the protonation state of a molecule. If the pKaNorm parameter is false, the TPSA value is calculated using the input structure and if pKaNorm is true, the TPSA is calculated using the pKa normalized molecular structure.
See Also:
- The *pKa normalization* section of the Filter Preprocessing chapter.
- function

11.6 Lipinski and Hydrogen-bonds

The work of Lipinski ([Lipinski-1997]) introduced the application of simple filter-like rules to roughly predict late-stage PK properties, in particular oral bioavailability. Unfortunately, Lipinski's "Rule-of-Five" has come into the common vernacular to such a large degree that some of the specific details are often lost in the commotion. There are two critical examples of this. First, Lipinski used "violation of 2 rules" to categorize compounds. In the subsequent analysis, a significant difference in the two populations of molecules was detected, however, little analysis of the importance of a single violation was done. However, many now consider a single violation to be bad and two violations

to be worse. At this writing, there is no known evidence to support this. Second, Lipinski used well-codified and well-understood yet imprecise definitions of "hydrogen-bond donors" and "hydrogen-bond acceptors" in his classification model. While this makes the algorithm quite understandable and easy to implement, it sometimes causes confusion with those who prefer more refined definitions of hydrogen-bond donors and acceptors.

To address the first problem, users are allowed to set the number of Lipinski failures required to reject a molecule. In keeping with the original publication, the default value is 2. To address the second problem, two kinds of hydrogen-bond donors and acceptors are calculated. In the Lipinski calculation, the published definitions are used (donor count is the number nitrogen or oxygen atoms with at least one hydrogen attached and acceptors are the number of nitrogens and oxygens). For calculation of the number of donors and acceptors in the molecule for the sake of chemical properties, a more complex algorithmic approach is taken. This approach identifies the donors and acceptors outlined in the work of Mills and Dean ([MillsDean-1996]) and also in the book by Jeffrey ([Jeffrey-1997]).

See Also:

The *Lipinski violations* and *hydrogen-bond acceptors* sections in the Filter Files chapter.

11.7 Aggregators

The Shoichet lab ([McGovern-2003], [Seidler-2003]) has demonstrated the importance of small-molecule aggregation in medium and high-throughput assays. Because these small-molecule aggregates can sequester some proteins, they give the appearance of being active inhibitors. There are now several hundred published aggregators in addition to a published QSAR model for predicting aggregation propensity. The user has the ability to eliminate any of the known aggregators as well as the ability to eliminate compounds that are predicted to be aggregators using the QSAR model. In OpenEye's experience, the published QSAR model for predicting aggregators is quite aggressive. It occasionally identifies compounds that are known to be genuine small-molecule inhibitors of a specific protein. While in most cases this model can be useful, until more definitive work is published, we feel you should gain some experience with the model and judge its performance for yourself.

More recent work in industry indicates that in some cases, aggregation properties are specific to the particular experimental conditions being used. Thus we recommended caution in the interpretation of these predictions. Nevertheless, aggregation remains an important issue in HTS hit follow-up and, short of experimental validation, this flag may be the best-available.

See Also:

The *aggregators parameter* in the Filter Files chapter.

CHAPTER

TWELVE

FILTER FILES

There are two parameter files a user can provide if they would like to override or augment the default parameter sets. The first is the "filter file". It provides acceptable limits for all of the physical properties and functional groups in the default filter. The second is the "newrule file". If you have a filter you like, but would like to augment it with a set of additional rules, these can be added with a newrule file.

There are four types of statements that can occur in a filter file:

- physical property limits
- rules
- new rules
- selections

The statements should occur one-per-line in the filter file.

Note: If the appropriate line is not in the filter file, or the value is false, the respective measure will not be used in filtering and its value will not be included in any table-based output.

12.1 Physical Property Limits

There are a large number of physical property limits. They occur as three fields on a line. For example:

```
MIN_HETEROATOMS 2 "Minimum number of heteroatoms"
```

The first field is the property keyword, the second field is the value assigned to that keyword, and the third field is a brief informational message. There are a fixed number of physical property keywords. No additional physical property keywords can be added by the user. The current keywords and brief definitions of each are listed below.

Hint: The values listed below are those found in the default *BlockBuster* filter.

12.1.1 Basic Properties

Molecular Weight

Isotopic molecular weight

```
MIN_MOLWT 130 "Minimum molecular weight"
MAX_MOLWT 781 "Maximum molecular weight"
```

Heavy Atom Count

Number of non-hydrogen atoms

```
MIN_NUM_HVY 9 "Minimum number of heavy atoms"
MAX_NUM_HVY 55 "Maximum number of heavy atoms"
```

Carbon Count

Number of carbons

```
MIN_CARBONS 3 "Minimum number of carbons"
MAX_CARBONS 41 "Maximum number of carbons"
```

Hetero-Count

Number of non-carbon and non-hydrogen atoms

```
MIN_HETEROATOMS 1 "Minimum number of heteroatoms"
MAX_HETEROATOMS 14 "Maximum number of heteroatoms"
```

Hetero-Atom to Carbon Ratio

Hetero-count/carbon-count

```
MIN_Het_C_Ratio 0.04 "Minimum heteroatom to carbon ratio"
MAX_Het_C_Ratio 4.0 "Maximum heteroatom to carbon ratio"
```

Chiral Count

Number of chiral atoms

```
MIN_CHIRAL_CENTERS 0 "Minimum chiral centers"
MAX_CHIRAL_CENTERS 21 "Maximum chiral centers"
```

Hydrogen-bond Acceptors

Number of atoms which match any of the following:

- degree 2, aromatic, non-positive nitrogens
- electron rich or negative, valence less than 4, non-aromatic nitrogens
- negatively charged or not electron withdrawn and neutral oxygens
- degree 1, double bonded, electron rich sulfur

```
MIN_HBOND_ACCEPTORS 0 "Minimum number of hydrogen-bond acceptors"
MAX_HBOND_ACCEPTORS 13 "Maximum number of hydrogen-bond acceptors"
```

Hydrogen-bond Donors

Number of hydrogen atoms on nitrogen, oxygen, or sulfur atoms

```
MIN_HBOND_DONORS 0 "Minimum number of hydrogen-bond donors"
MAX_HBOND_DONORS 9 "Maximum number of hydrogen-bond donors"
```

Lipinski Acceptors

Number of nitrogens or oxygens

```
MIN_LIPINSKI_ACCEPTORS 1 "Minimum number of oxygen & nitrogen atoms"
MAX_LIPINSKI_ACCEPTORS 14 "Maximum number of oxygen & nitrogen atoms"
```

Lipinski Donors

Number of hydrogens attached to nitrogen or oxygen

```
MIN_LIPINSKI_DONORS 0 "Minimum number of hydrogens on O & N atoms"
MAX_LIPINSKI_DONORS 6 "Maximum number of hydrogens on O & N atoms"
```

Halide Fraction

Percent of molecular weight from halides

```
MIN_HALIDE_FRACTION 0.0 "Minimum Halide Fraction"
MAX_HALIDE_FRACTION 0.66 "Maximum Halide Fraction"
```

Formal Count

Number of atoms with a formal charge (excludes dative)

```
MIN_COUNT_FORMAL_CRG 0 "Minimum number formal charges"
MAX_COUNT_FORMAL_CRG 4 "Maximum number of formal charges"
```

Formal Sum

Total formal charge

```
MIN_SUM_FORMAL_CRG -2 "Minimum sum of formal charges"
MAX_SUM_FORMAL_CRG 2 "Maximum sum of formal charges"
```

Connected Non-Ring

Considers sets of contiguous (bonded) non-ring atoms

```
MIN_CON_NON_RING 0 "Minimum number of connected non-ring atoms"
MAX_CON_NON_RING 19 "Maximum number of connected non-ring atoms"
```

Unbranched Chains

The size of unbranched non-ring chains

```
MIN_UNBRANCHED 1 "Minimum number of connected unbranched non-ring atoms"
MAX_UNBRANCHED 13 "Maximum number of connected unbranched non-ring atoms"
```

Total Functional Group Count

Total number of functional groups. Does not count any ring-systems as functional groups. Degree 1 heteroatoms, particularly those with double bonds or dative bonds are considered part of ring systems and do not count as a functional group.

```
MIN_FCNGRP 0 "Minimum number of functional groups"
MAX_FCNGRP 7 "Maximum number of functional groups"
```

Note: This is different then the functional group rules.

Ring Systems

Number of ring systems (contiguous systems of ring atoms and bonds)

```
MIN_RING_SYS 0 "Minimum number of ring systems"
MAX_RING_SYS 5 "Maximum number of ring systems"
```

Ring Size

Maximum size of any single ring system

```
MIN_RING_SIZE 0 "Minimum atoms in any ring system"
MAX_RING_SIZE 20 "Maximum atoms in any ring system"
```

Rotor Count

Number of rotatable bonds. Allows optional adjustment for aliphatic rings following the method of [Oprea-2000].

```
MIN_ROT_BONDS 0 "Minimum number of rotatable bonds"
MAX_ROT_BONDS 16 "Maximum number of rotatable bonds"
ADJUST_ROT_FOR_RING true "BOOLEAN for whether to estimate degrees of freedom in rings"
```

Rigid Count

Number of rigid bonds (non-rotatable bonds)

```
MIN_RIGID_BONDS 4 "Minimum number of rigid bonds"
MAX_RIGID_BONDS 55 "Maximum number of rigid bonds"
```

12.1.2 LogP

The logP calculation is a derivative of the published XLogP algorithm [Wang-1997]., but reparameterized without the dependence on 3D coordinates or the SYBYL/Mol2 aromaticity model.

XLogP

Calculated LogP

```
MIN_XLOGP -3.0 "Minimum XLogP"
MAX_XLOGP 6.85 "Maximum XLogP"
```

12.1.3 Solubility

The solubility predictions are based on using the atom-types from the XLogP algorithm, [Wang-1997], and reparameterizing them based on available solubility data. Rather than a quantitative cutoff, solubility uses categories. The 6 allowable categories are:

1. insoluble
2. poorly
3. moderately
4. soluble
5. very
6. highly

These categories are keywords used in the filter files as follows.

Solubility

Calculated solubility class

```
MIN_SOLUBILITY insoluble "Minimum solubility"
```

12.1.4 Pharmacokinetic Predictors

Several secondary filters that are built upon published combinations of simpler properties are available.

Note: All of these properties are used for filtering in the default filters.

Lipinski Violations

Number of allowable Lipinski violations. A single Lipinski violation is considered acceptable. The published work, [Lipinski-1997], allows compounds to pass with a single violation but not multiple violations.

```
MAX_LIPINSKI 3 "Maximum number of Lipinski violations"
```

See Also:

The *Lipinski theory* section in the Molecular Properties and Predictors chapter.

PSA

Peter Ertl's, [Ertl-2000], topological polar surface area (phosphorus and sulfur area is optional).

```
PSA_USE_SandP false "Count S and P as polar atoms"
MIN_2D_PSA 0.0 "Minimum 2-Dimensional (SMILES) Polar Surface Area"
MAX_2D_PSA 205.0 "Maximum 2-Dimensional (SMILES) Polar Surface Area"
```

See Also:

The *PSA theory* section in the Molecular Properties and Predictors chapter.

GSK/Veber

Veber's measure of bioavailability (PSA > 140 or Rotatable bonds >10). [Veber-2002].

```
GSK_VEBER false "PSA>140 or >10 rot bonds"
```

Abbott/Martin

Yvonne Martin's Abbott Bioavailability Score. This is reported as a probability that `F>10%` in rats. [Martin-2005]

```
MIN_ABS 0.11 "Minimum probability F>10% in rats"
```

Pharmacopia/Egan

`Egan egg` measure of bioavailability (LogP >5.88 or PSA > 131.6). [Egan-2000]

```
PHARMACOPIA false "LogP > 5.88 or PSA > 131.6"
```

12.1.5 Aggregators

Aggregators are small molecules that can interfere with assay results by sequestering protein in an aggregation of small molecules in solution. They appear to have activity in many assays, but in fact are usually not specific inhibitors of the protein in question. Includes two measures of whether a molecule is one of the aggregators defined by Shoichet et. al. [McGovern-2003] [Seidler-2003] The first measure, `AGGREGATORS`, is whether the molecule is an exact match to one of the approximately 400 published aggregators. The second measure, `PRED_AGG`, is whether the molecule hits in Shoichet's QSAR model for predicting aggregators.

Aggregators

Whether a compound is known or predicted to aggregate in concentrations common in virtual screening.

```
AGGREGATORS true "Eliminate known aggregators"
PRED_AGG false "Eliminate predicted aggregators"
```

12.1.6 Elemental Filters

The elemental filters are applied in this order:

1. Test for the existence of any of the metals in the ELIMINATE_METALS filter in the molecule.
2. Remove salts by stripping away all the disconnected components except for the largest.
3. Test to make sure only atoms specified in ALLOWED_ELEMENTS filter are in the molecule.

See Also:

- *Metal Removal*
- *Salt Removal*

The format of the two elemental filter fields is the keyword followed by a comma delimited list of atomic symbols.

Eliminate Metals

Any molecule with the atoms indicated in ELIMINATE_METALS fail to pass the filter.

```
ELIMINATE_METALS Sc,Ti,V,Cr,Mn,Fe,Co,Ni,Cu,Zn,Y,Zr,Nb,Mo,Tc,Ru,Rh,Pd,Ag,Cd
```

Allowed Elements

Molecules with atoms other than those specified by ALLOWED_ELEMENTS fail to pass the filter.

```
ALLOWED_ELEMENTS H,C,N,O,F,P,S,Cl,Br,I
```

12.2 Functional Group Rules

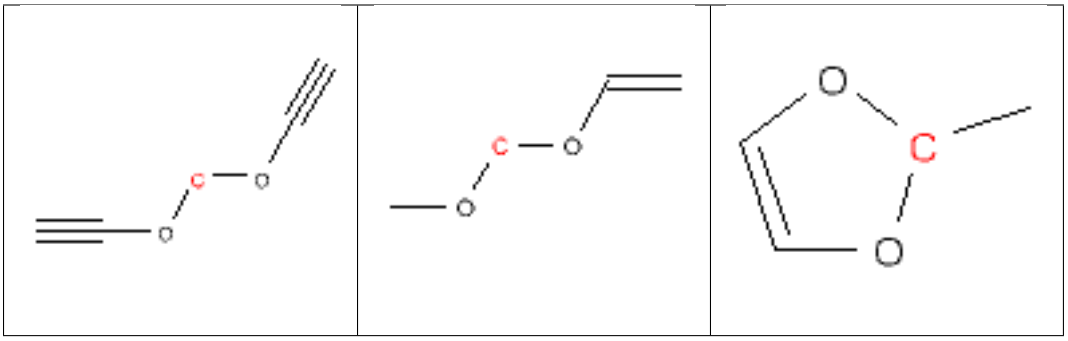
Rules statements set the limits for the maximum number of the specified type of functional group that may be allowed in the molecule.

The first field of a rule statement is the word RULE in all capital letters. The second field is a number indicating the maximum number of the group allowed in a molecule. The third field is the functional group keyword. Functional-group keywords are case sensitive.

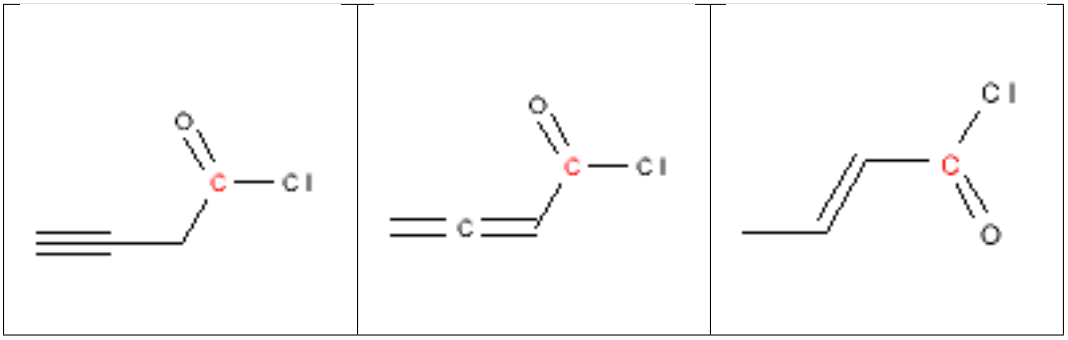
```
RULE 0 acid_halide
```

The following is a list of functional groups which filter recognizes by default. Three example matches are provided with the atoms that correspond to each other highlighted. Due to the highly complex nature of the patterns, in particular recursive SMARTS, it is not possible to fully highlight every atom that was included as part of the match.

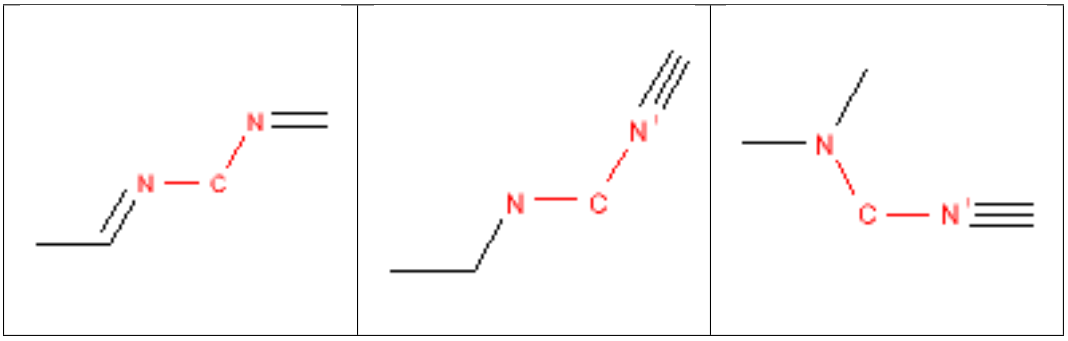
12.2.1 acetal



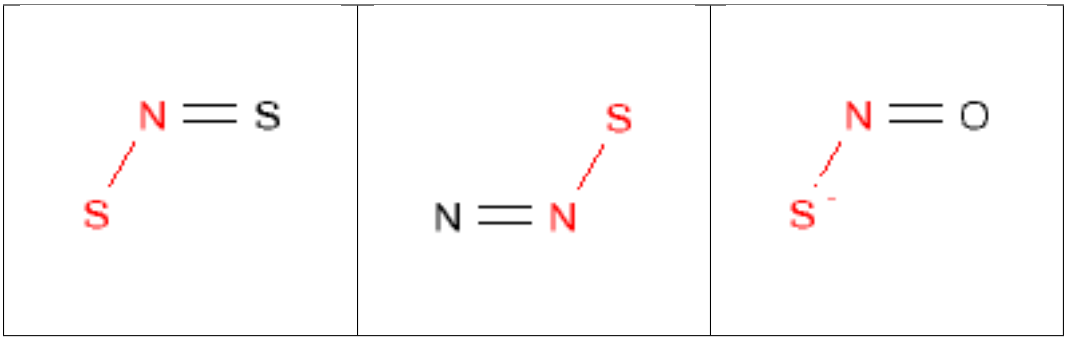
12.2.2 acid_halide



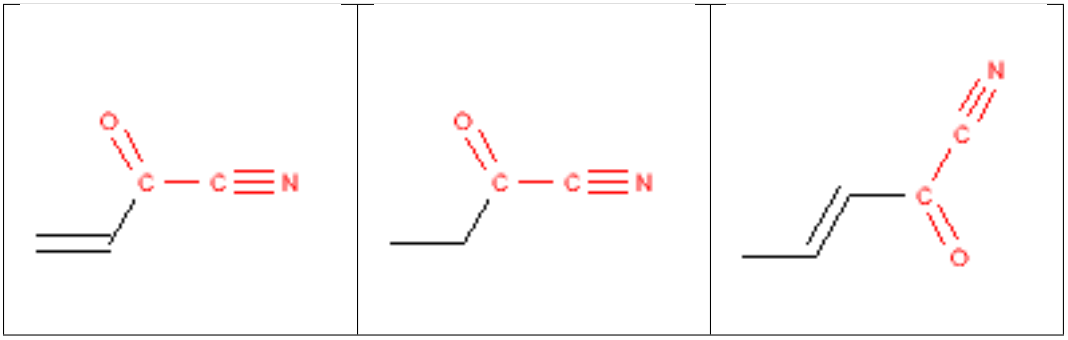
12.2.3 acyclic_NCN



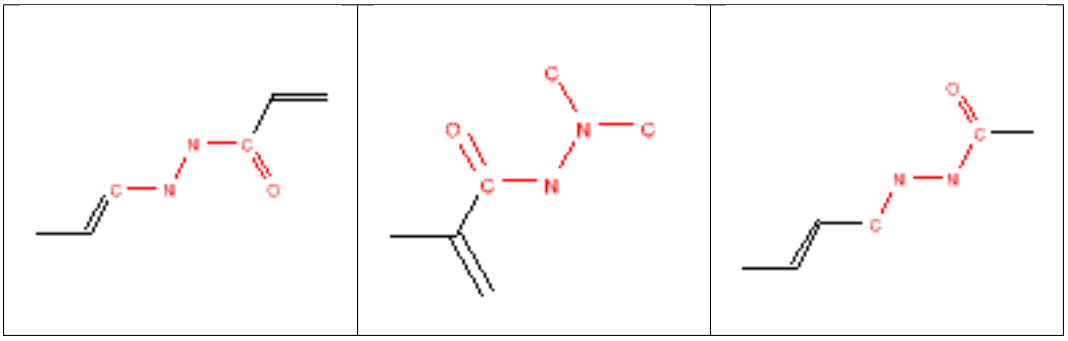
12.2.4 acyclic_NS



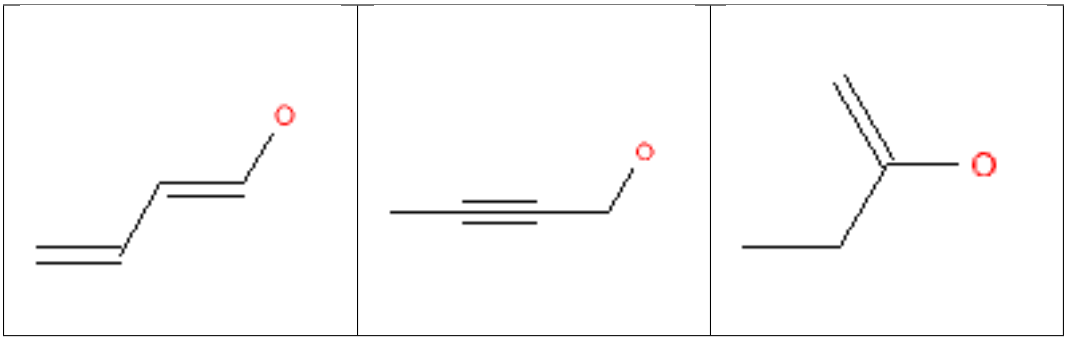
12.2.5 acyl_cyanides



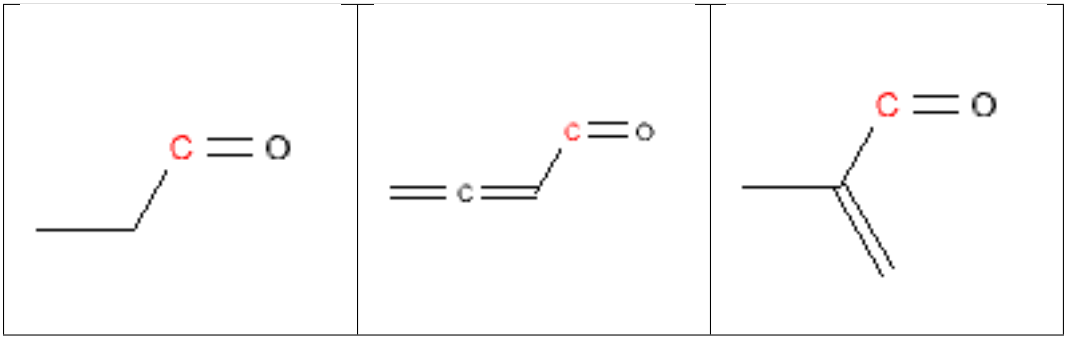
12.2.6 acylhydrazide



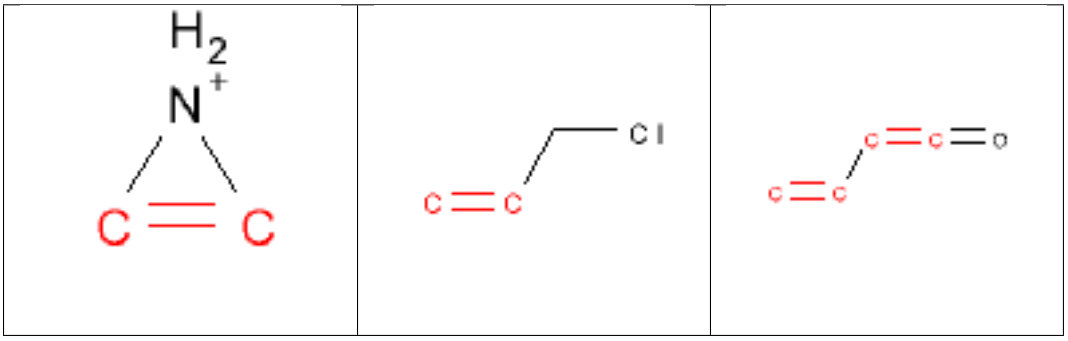
12.2.7 alcohol



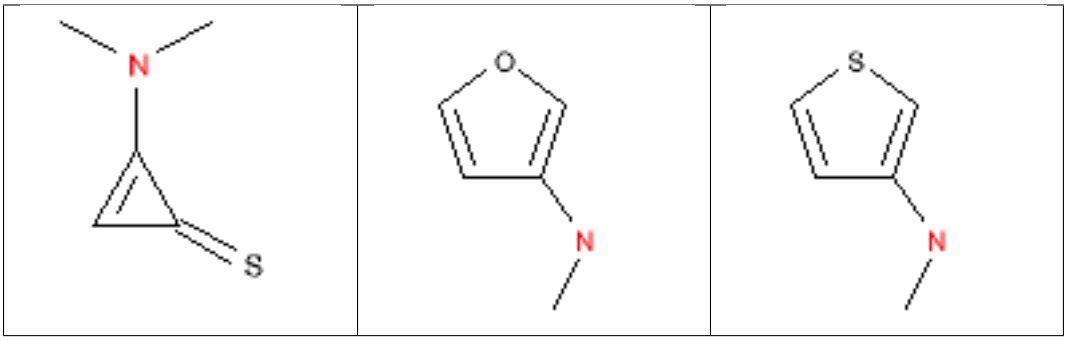
12.2.8 aldehyde



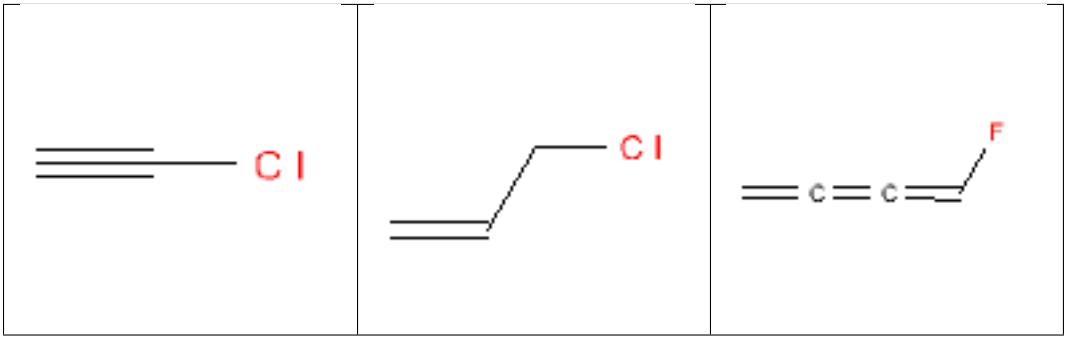
12.2.9 alkene



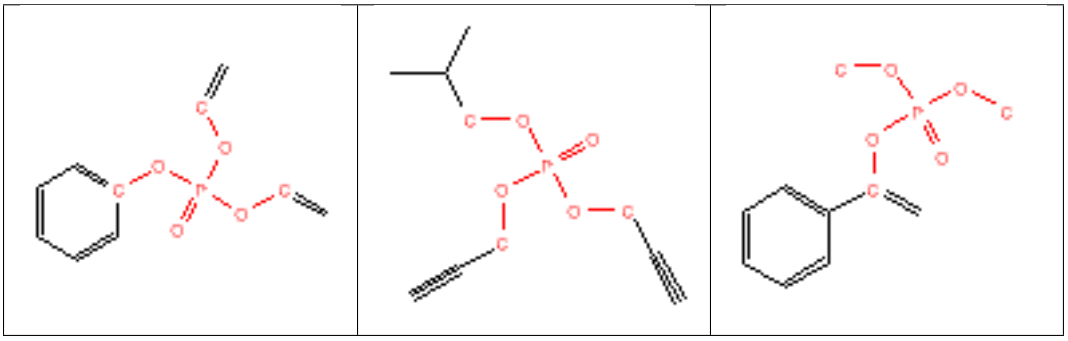
12.2.10 alkylaniline



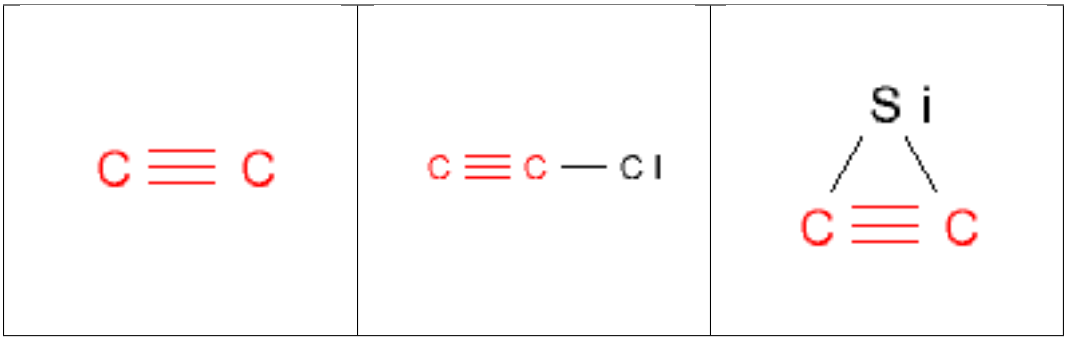
12.2.11 alkyl_halide



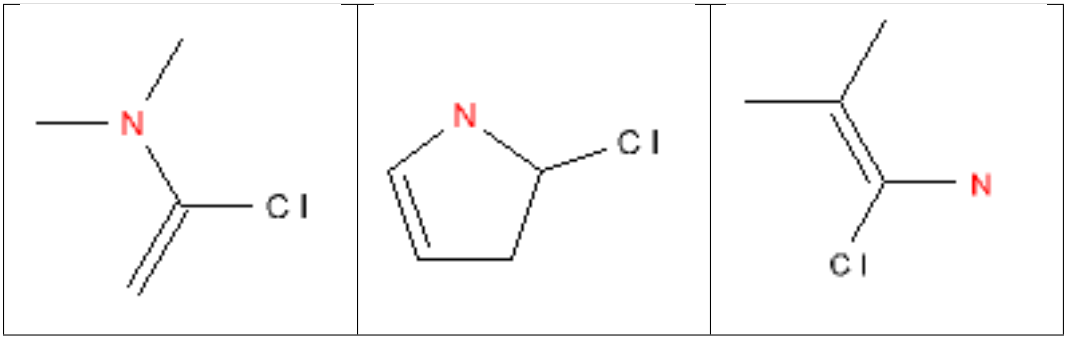
12.2.12 alkyl_phosphate



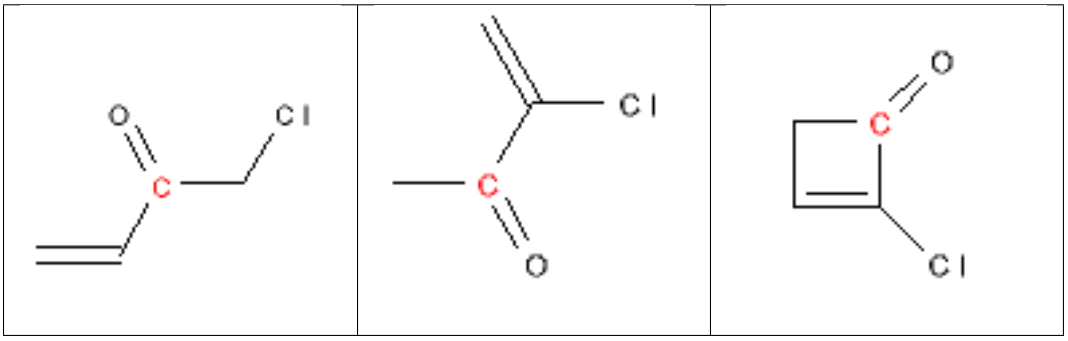
12.2.13 alkyne



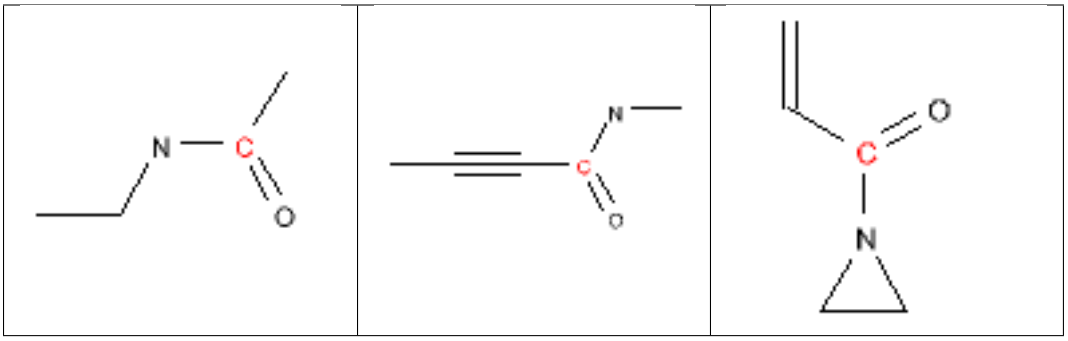
12.2.14 alphahalo_amine



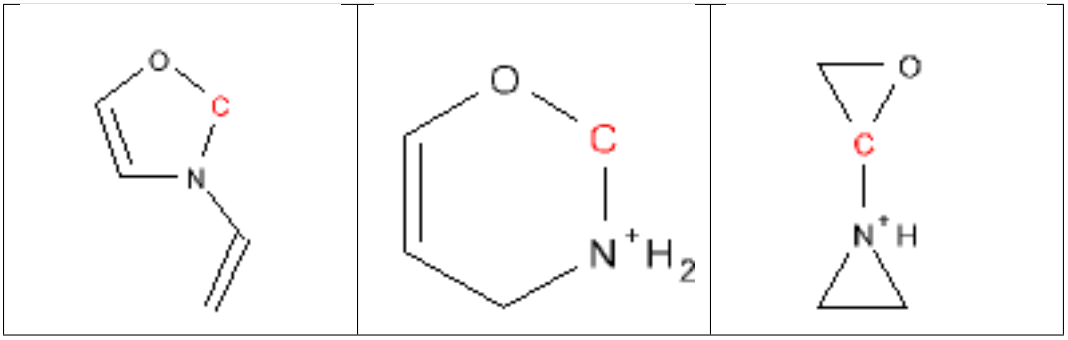
12.2.15 alphahalo_ketone



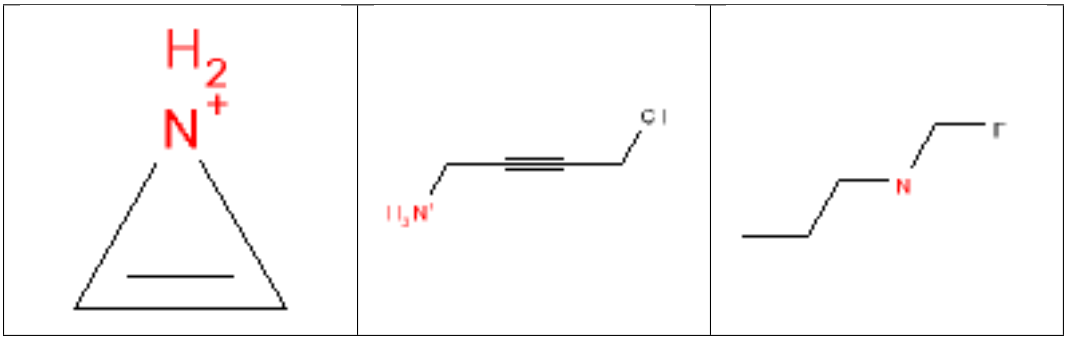
12.2.16 amide



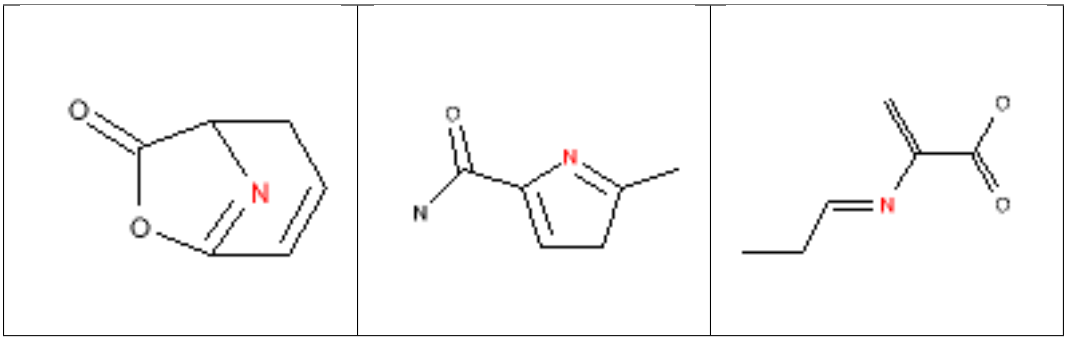
12.2.17 aminal



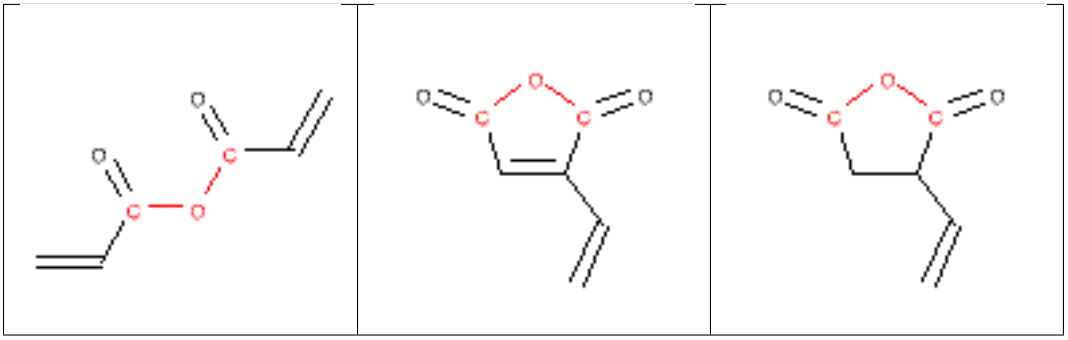
12.2.18 amine



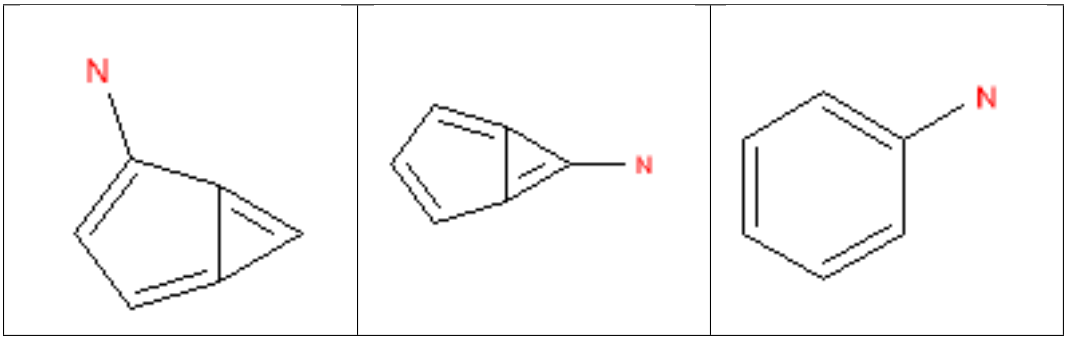
12.2.19 amino_acid



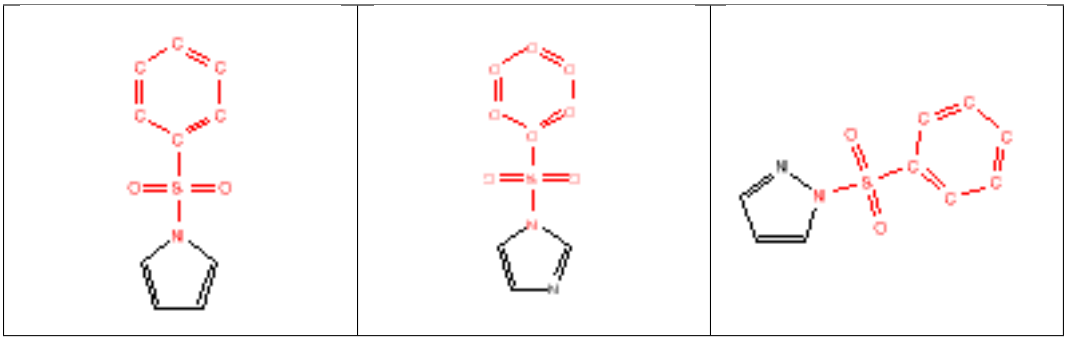
12.2.20 anhydride



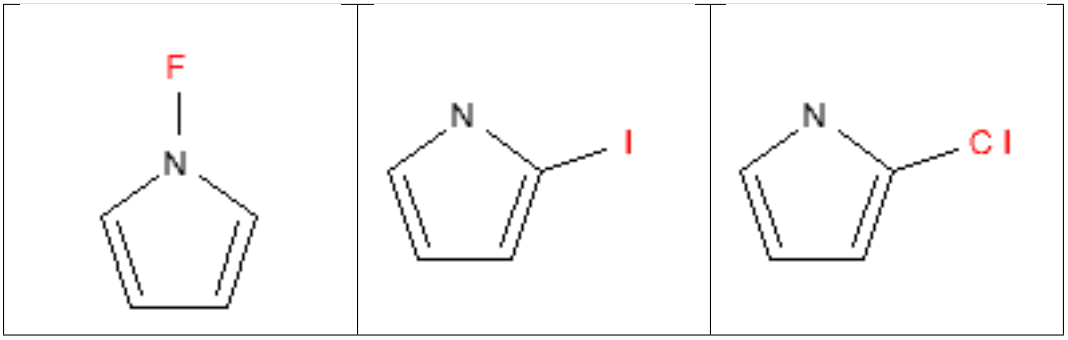
12.2.21 aniline



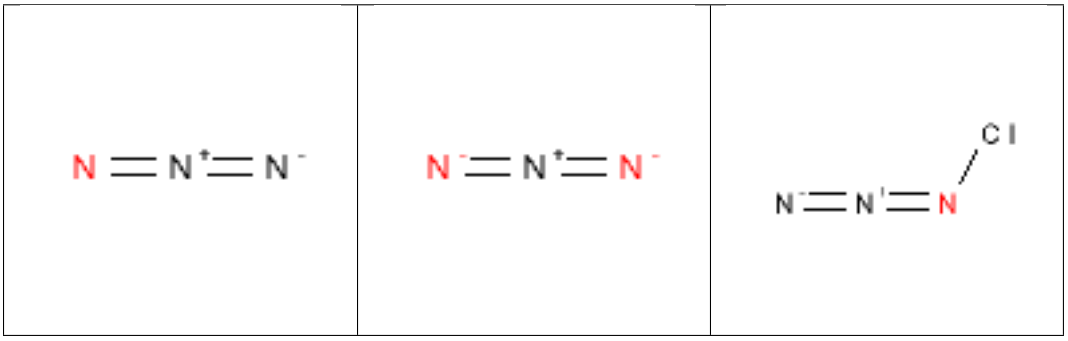
12.2.22 arenesulfonyl



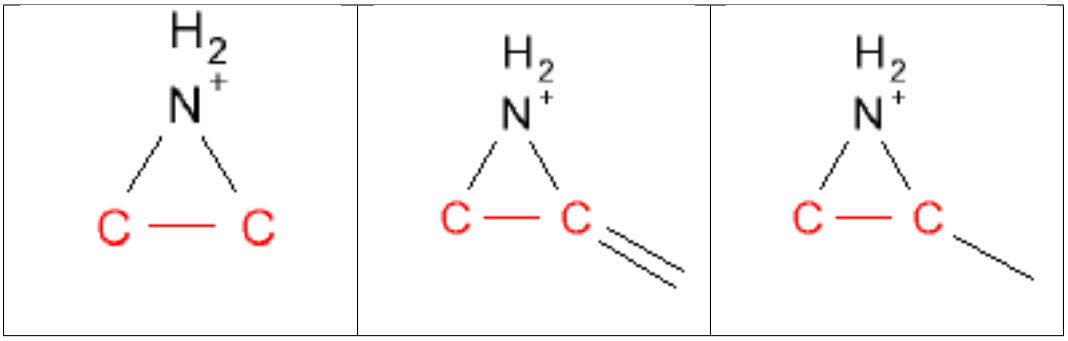
12.2.23 aryl_halide



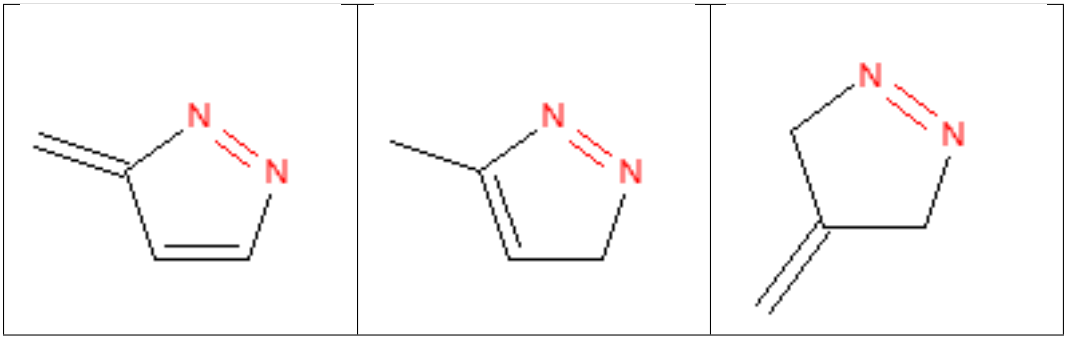
12.2.24 azide



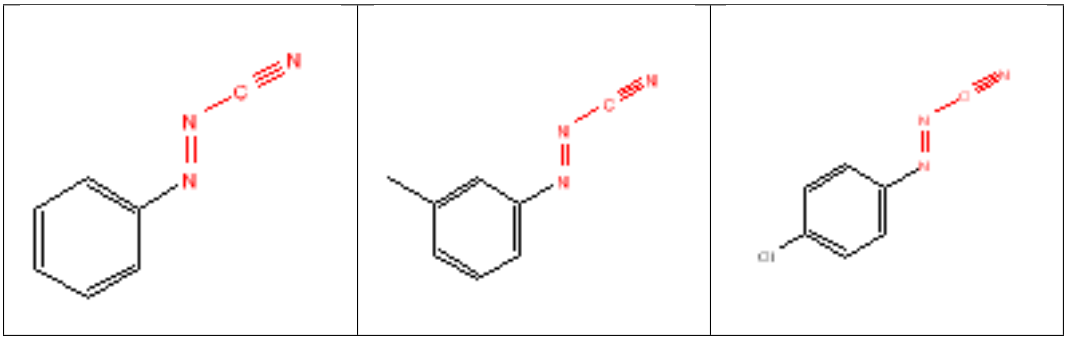
12.2.25 aziridine



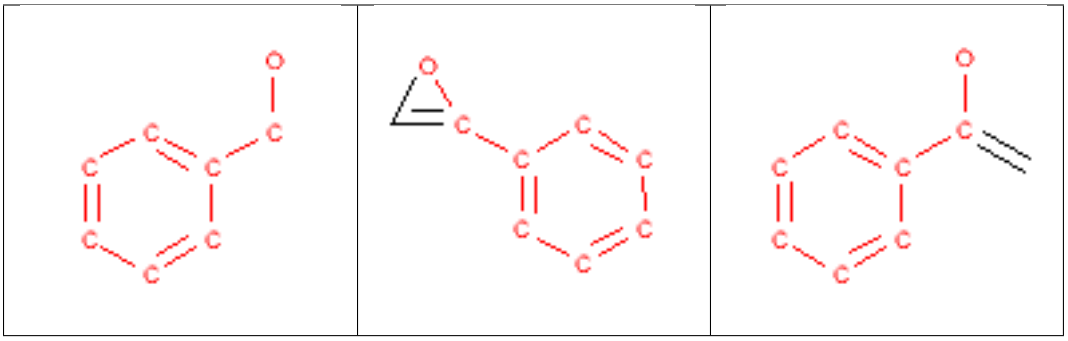
12.2.26 azo



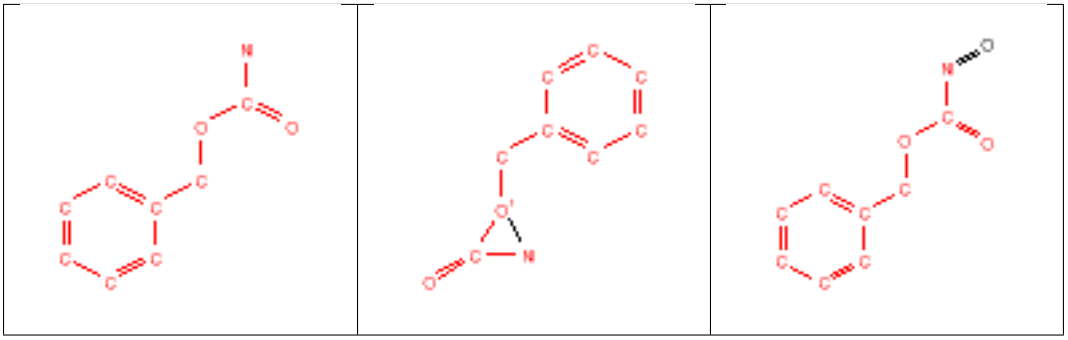
12.2.27 azocyanamides



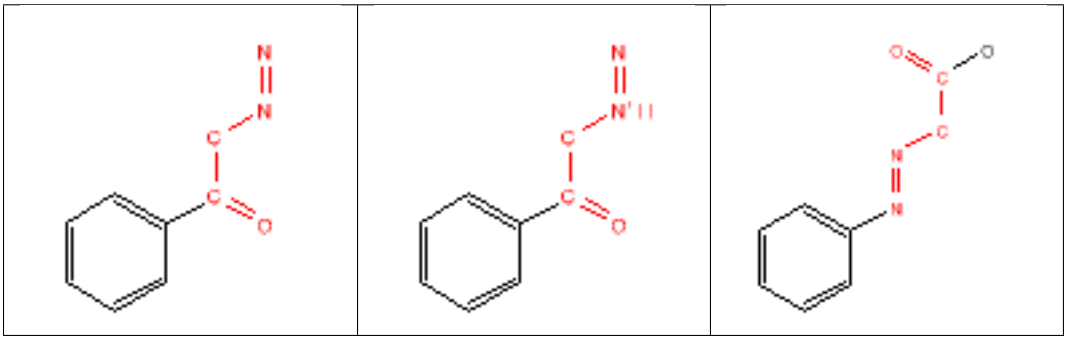
12.2.28 benzyl_ether



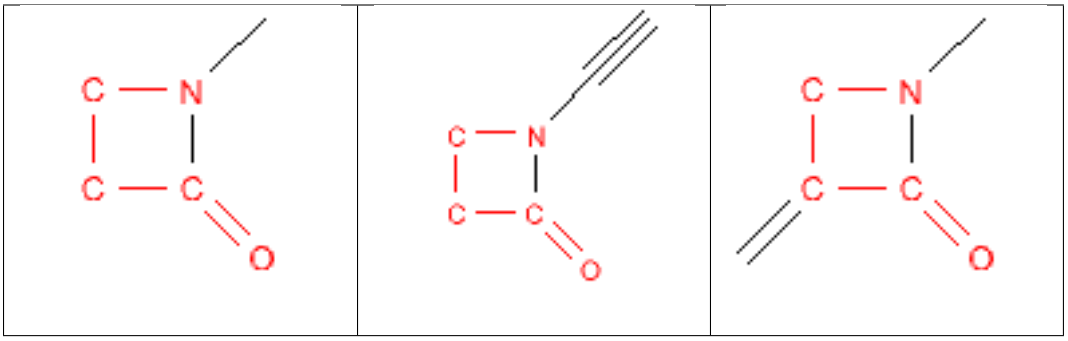
12.2.29 benzyloxycarbonyl_CBZ



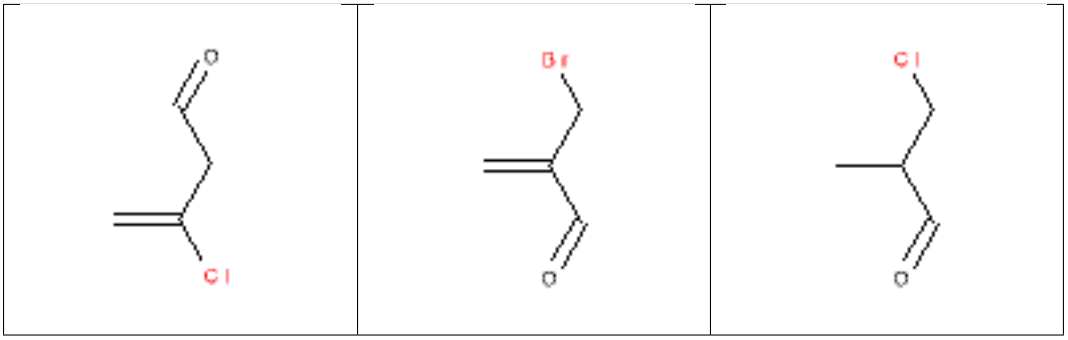
12.2.30 beta_azo_carbonyl



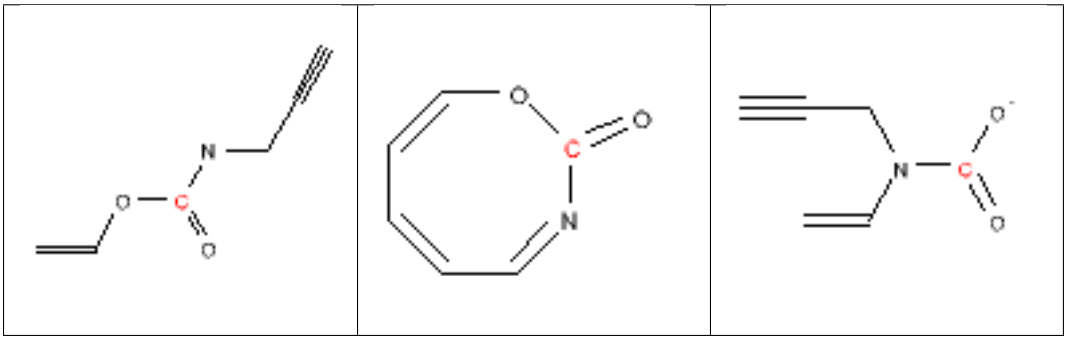
12.2.31 beta_carbonyl_quat_nitrogen



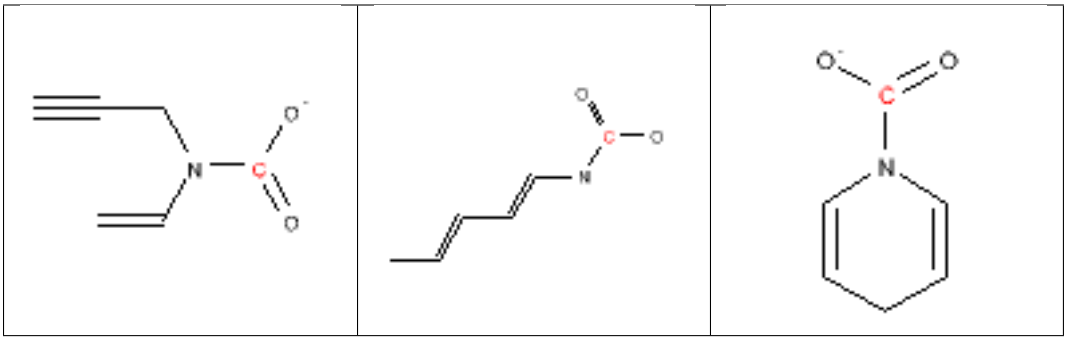
12.2.32 beta_halo_carbonyl



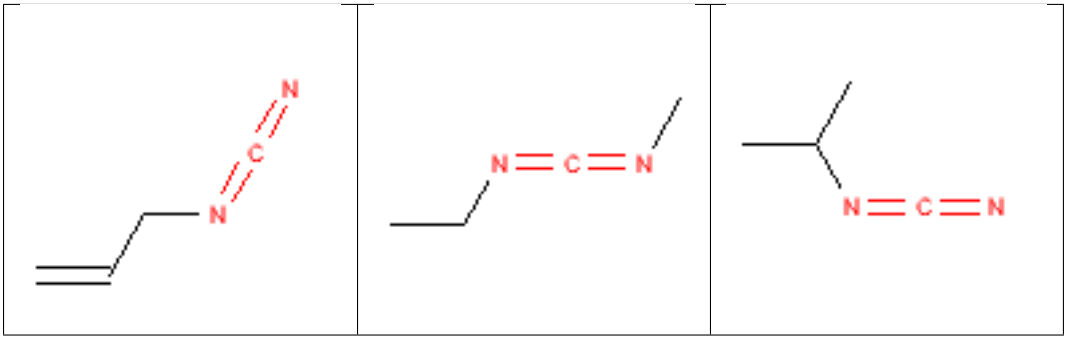
12.2.33 carbamate



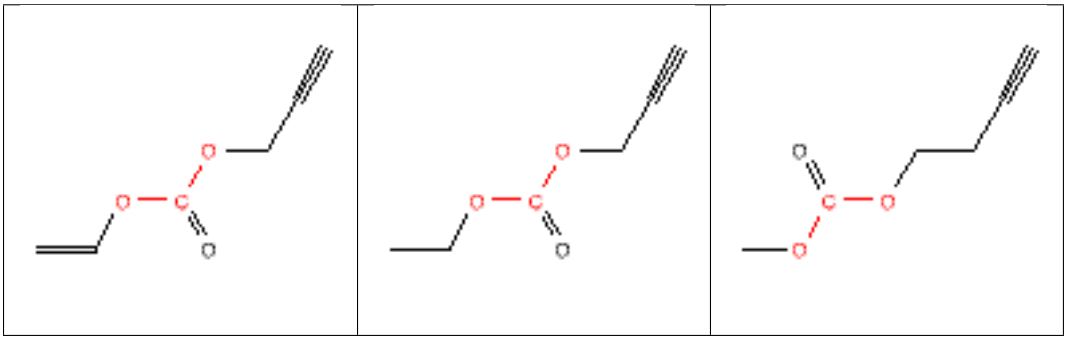
12.2.34 carbamic_acid



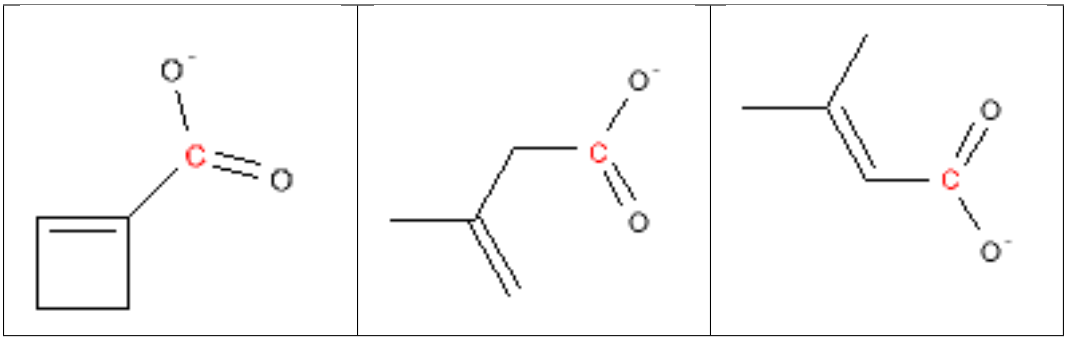
12.2.35 carbodiimide



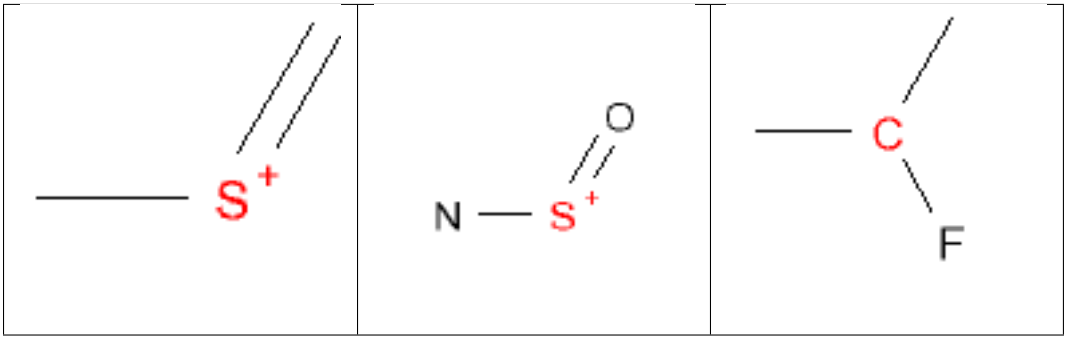
12.2.36 carbonate



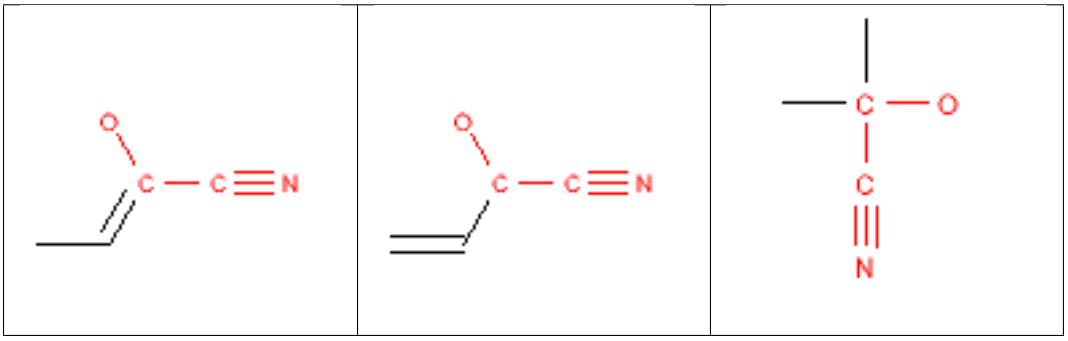
12.2.37 carboxylic_acid



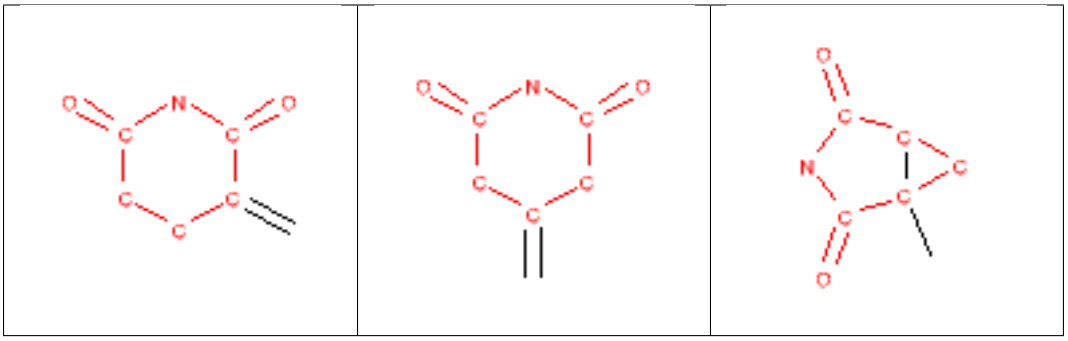
12.2.38 cation_C_Cl_I_P_or_S



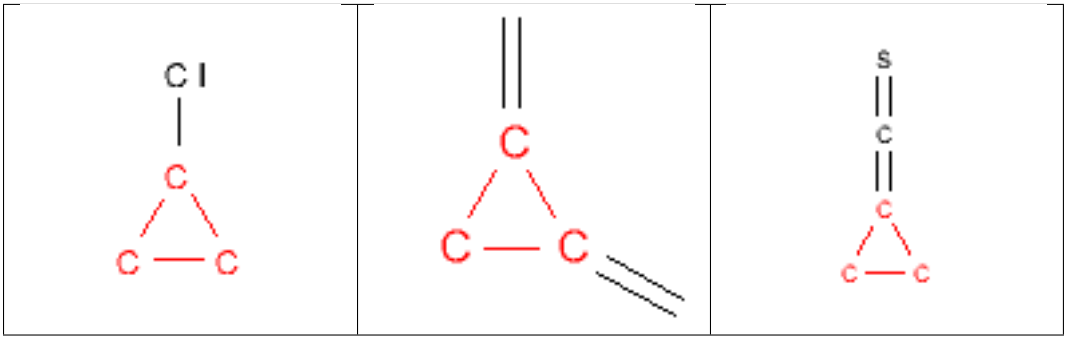
12.2.39 cyanohydrins



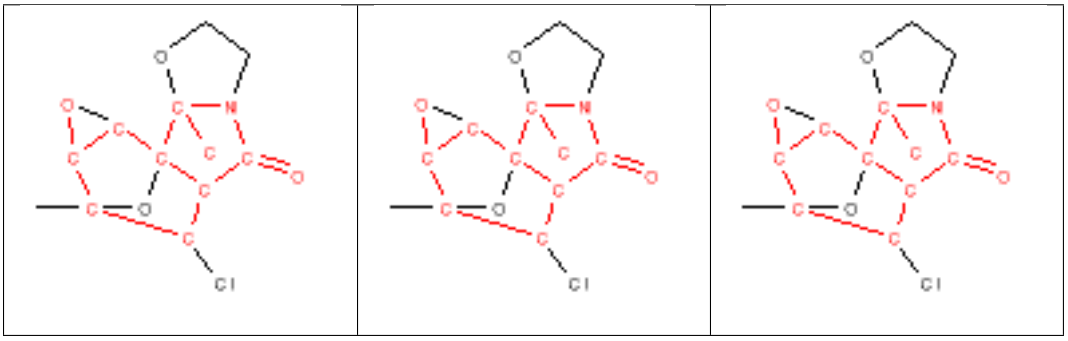
12.2.40 cycloheximide_derivatives



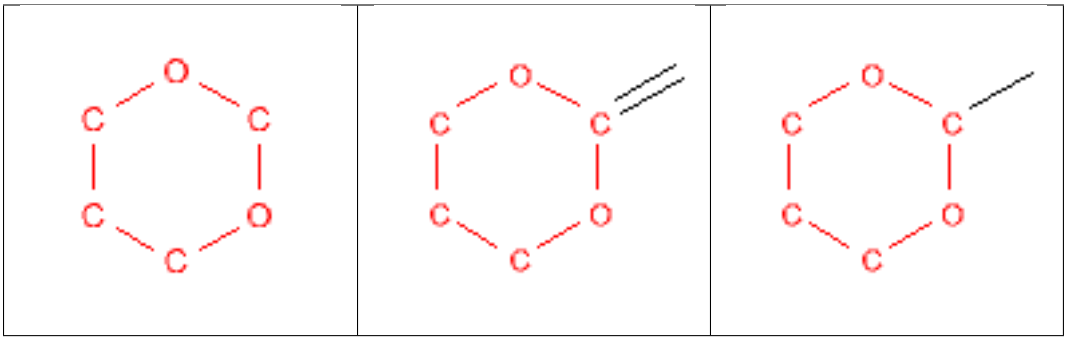
12.2.41 cyclopropyl



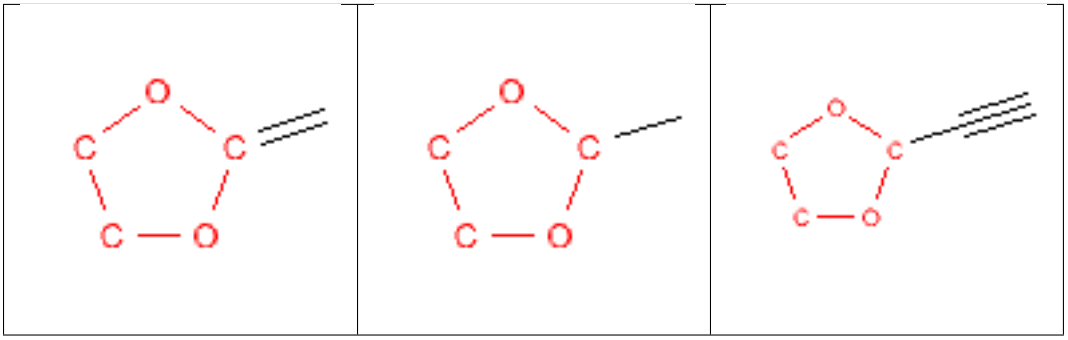
12.2.42 cytochalasin_derivatives



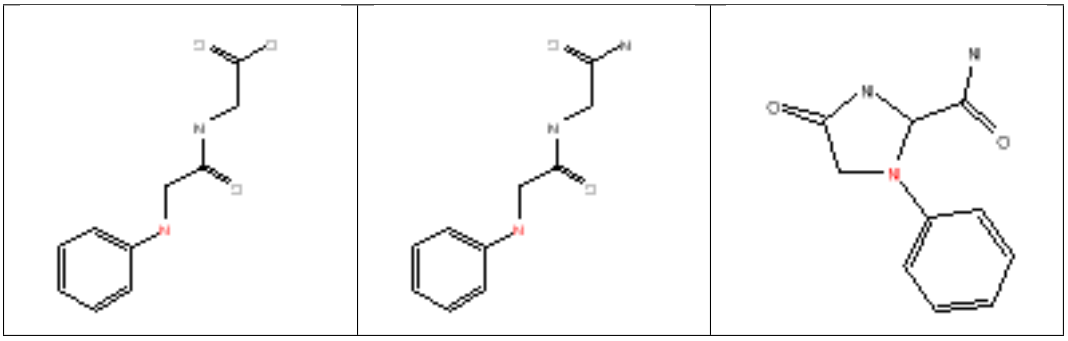
12.2.43 dioxane_6MR



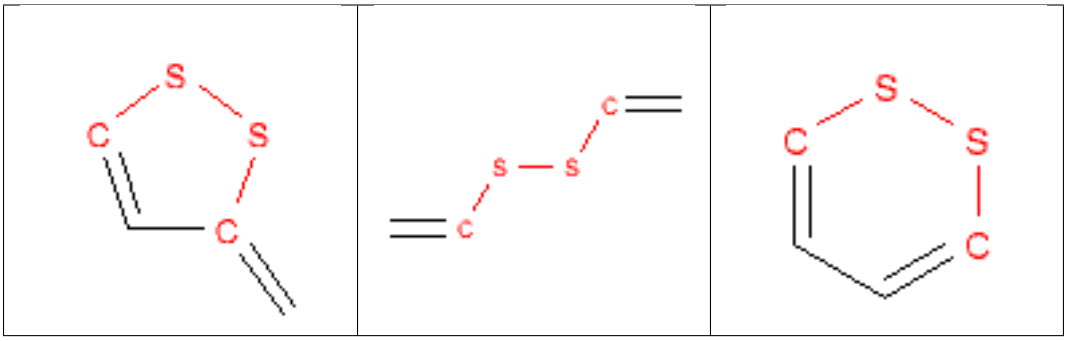
12.2.44 dioxolane_5MR



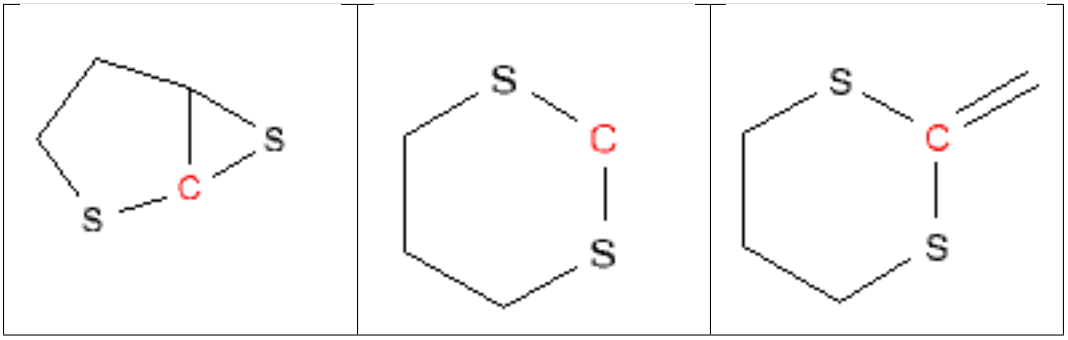
12.2.45 di_peptide



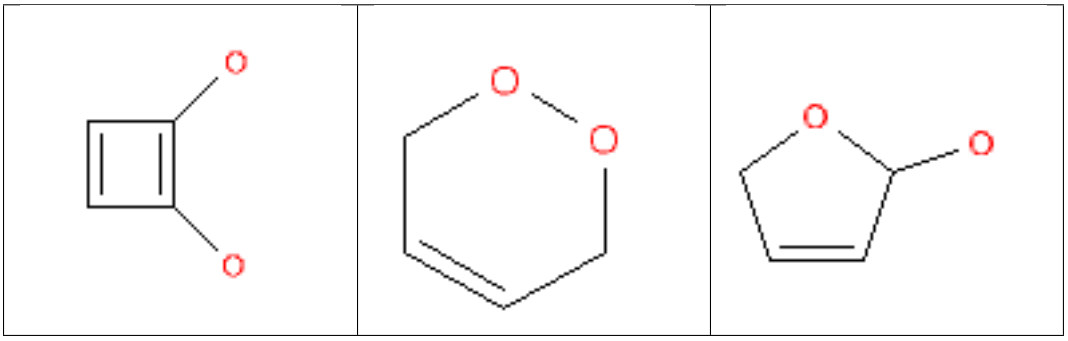
12.2.46 disulfide



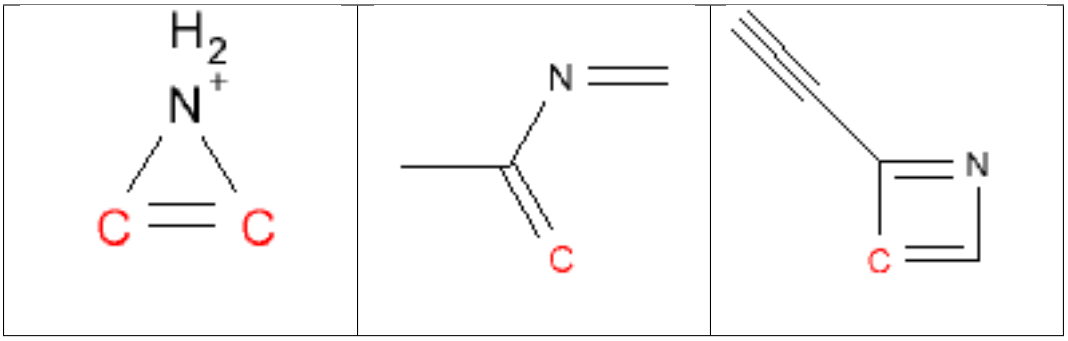
12.2.47 dithioacetal



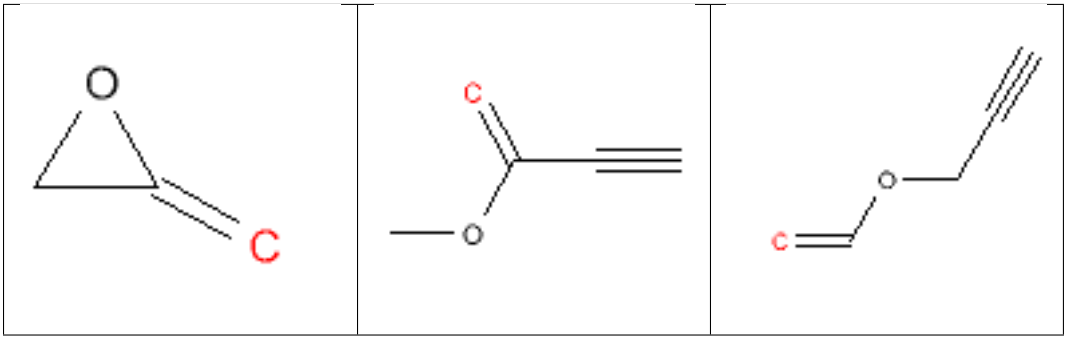
12.2.48 dye



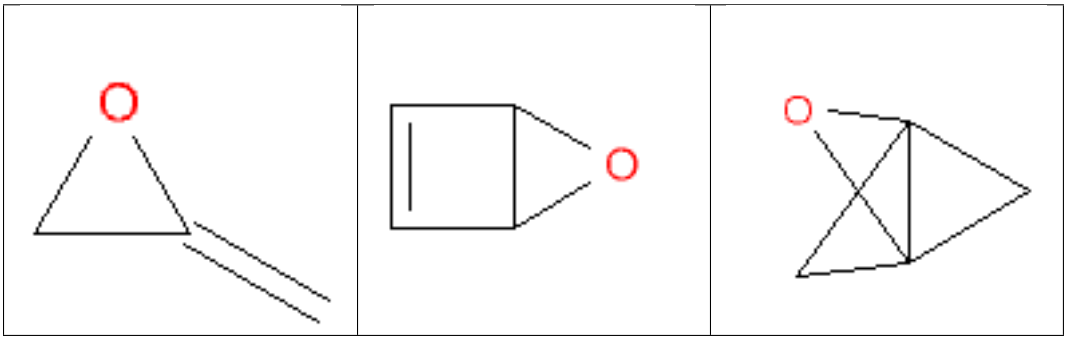
12.2.49 enamine



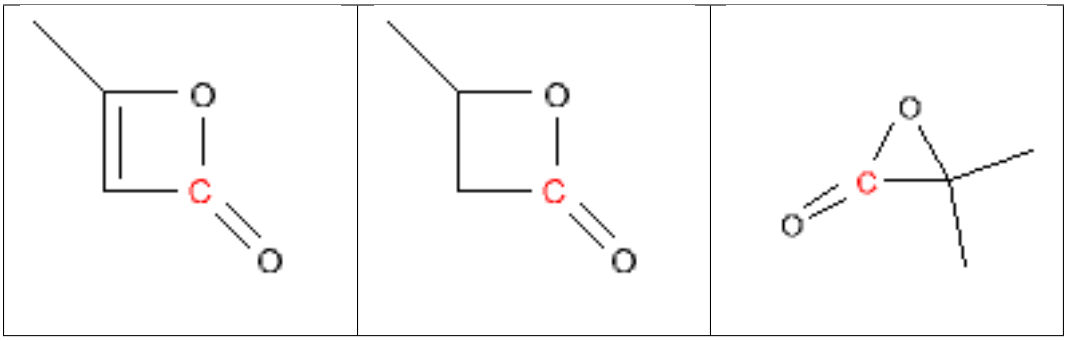
12.2.50 enol_ether



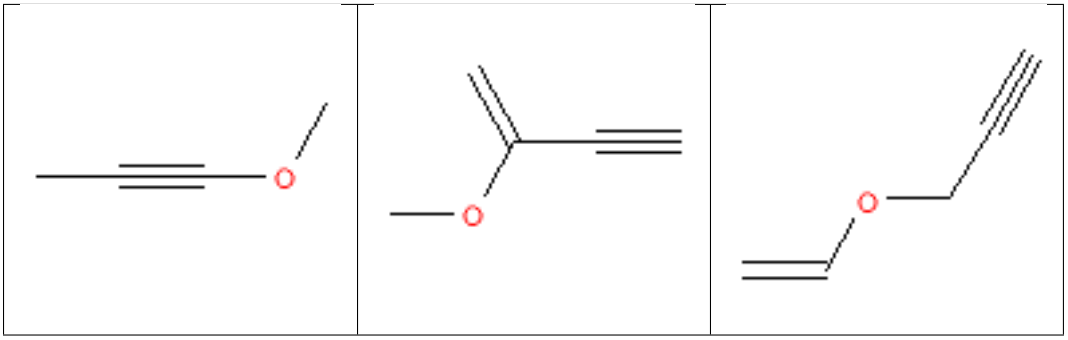
12.2.51 epoxide



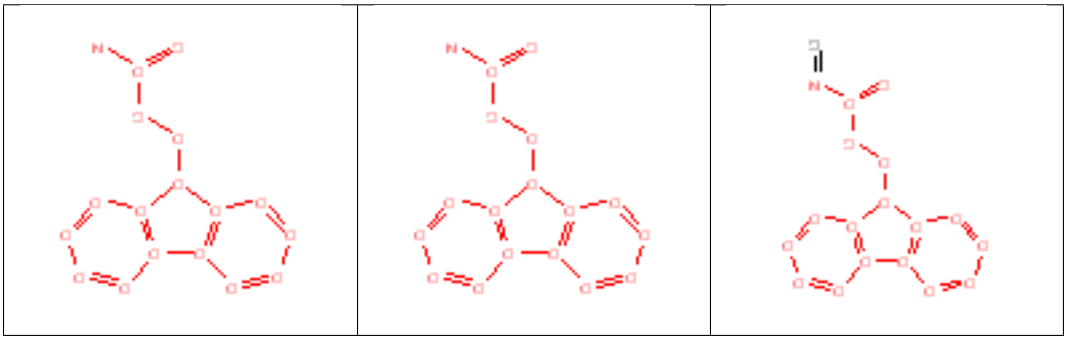
12.2.52 ester



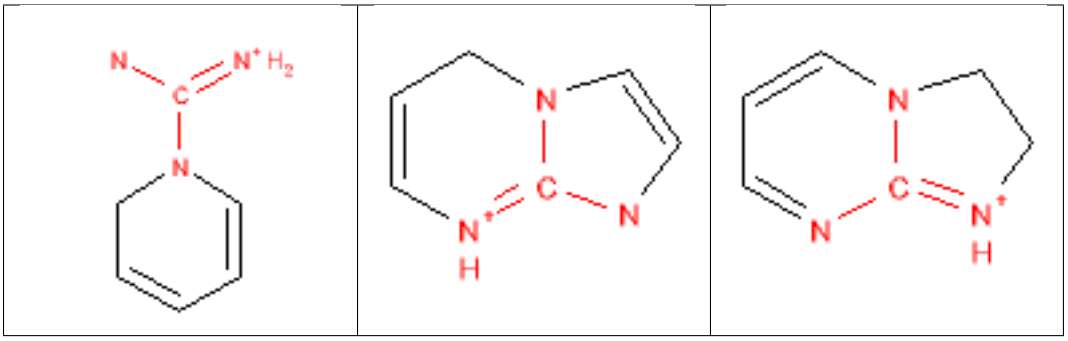
12.2.53 ether



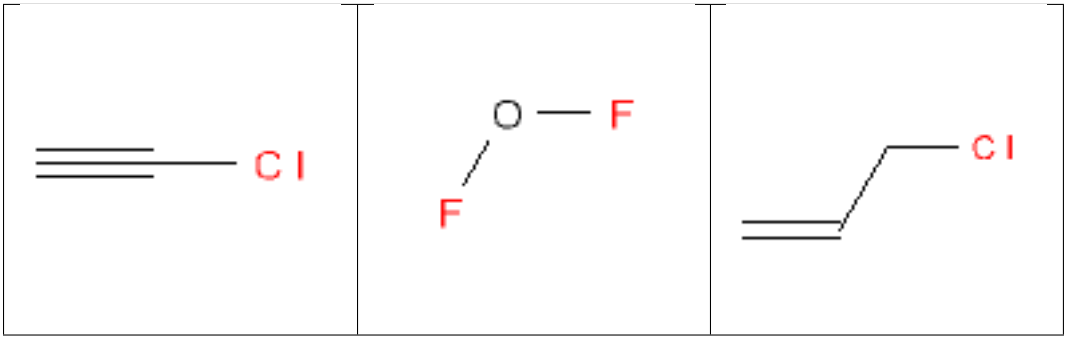
12.2.54 fluorenylmethoxycarbonyl_Fmoc



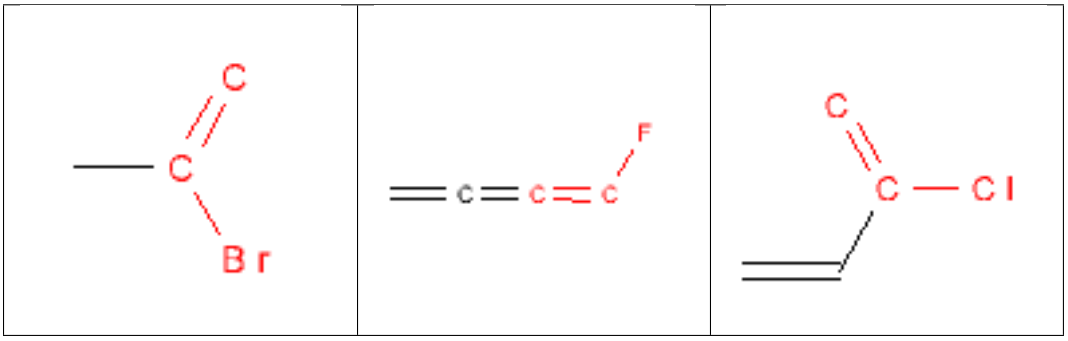
12.2.55 guanidine



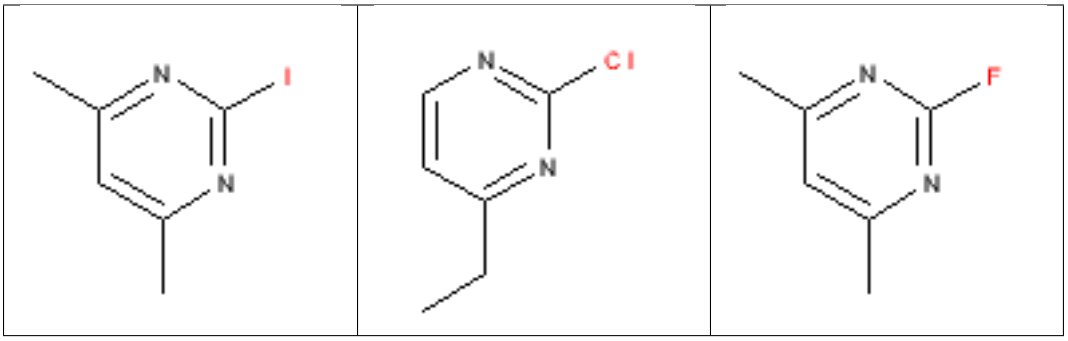
12.2.56 halide



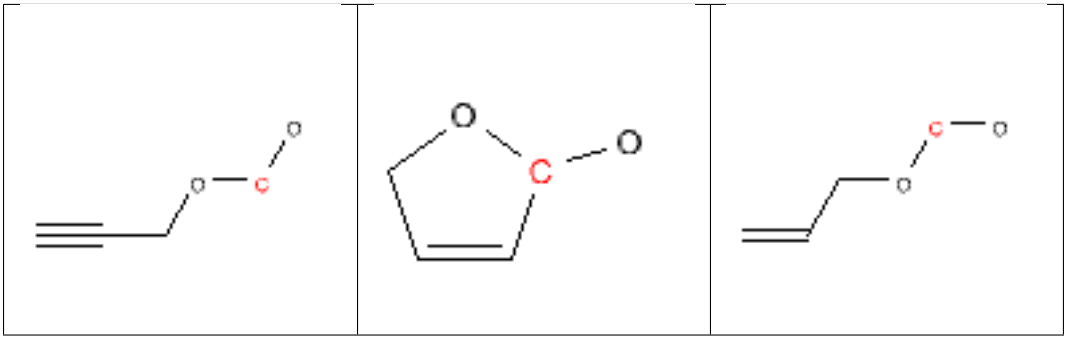
12.2.57 halo_alkene



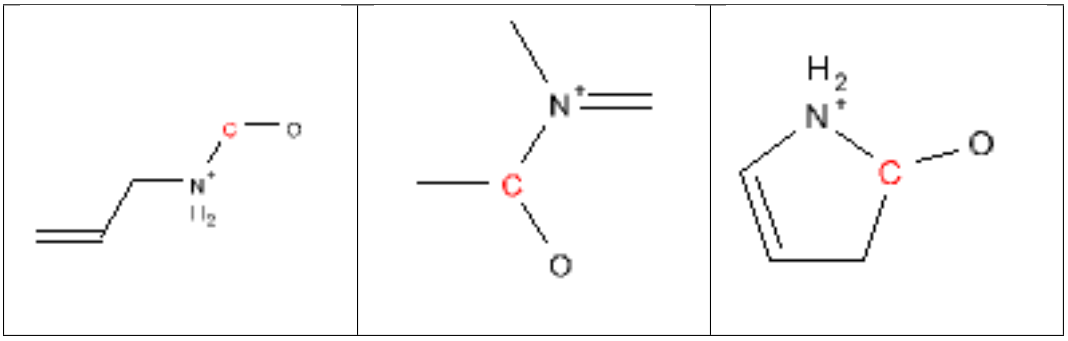
12.2.58 halopyrimidine



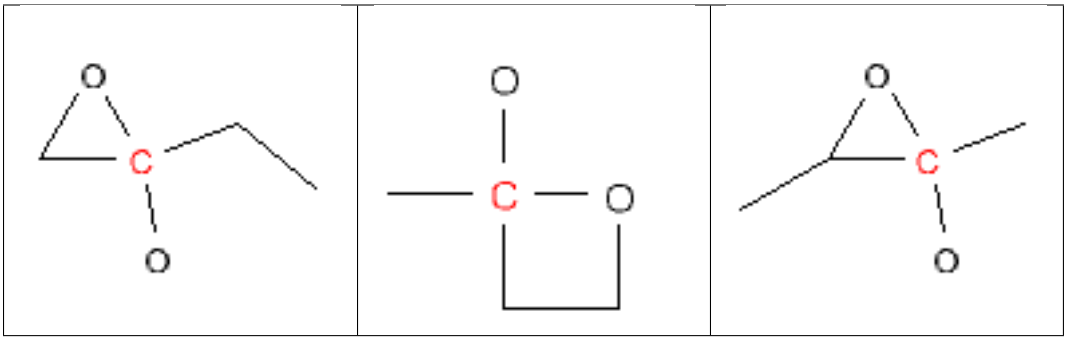
12.2.59 hemiacetal



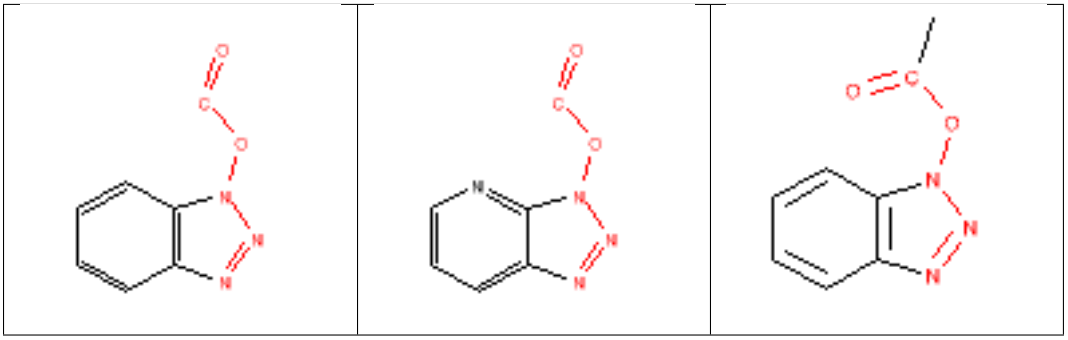
12.2.60 hemiaminal



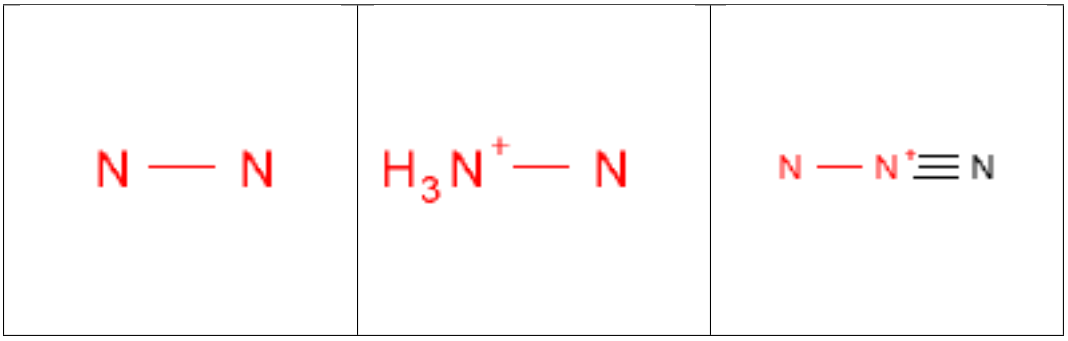
12.2.61 hemiketal



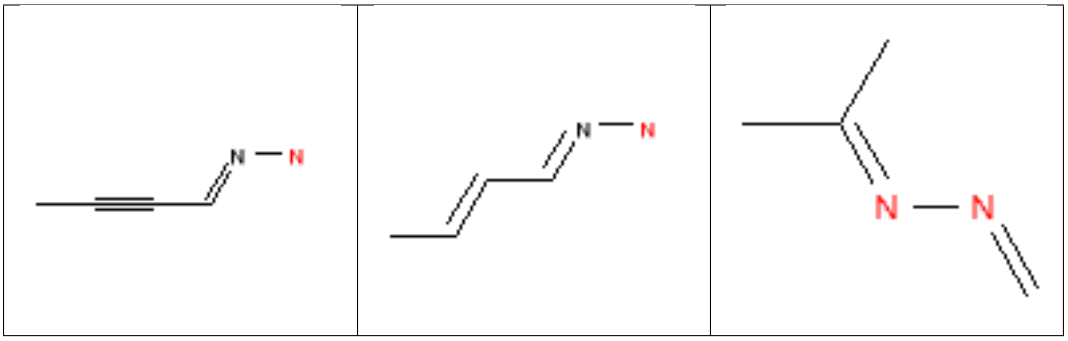
12.2.62 HOBT_esters



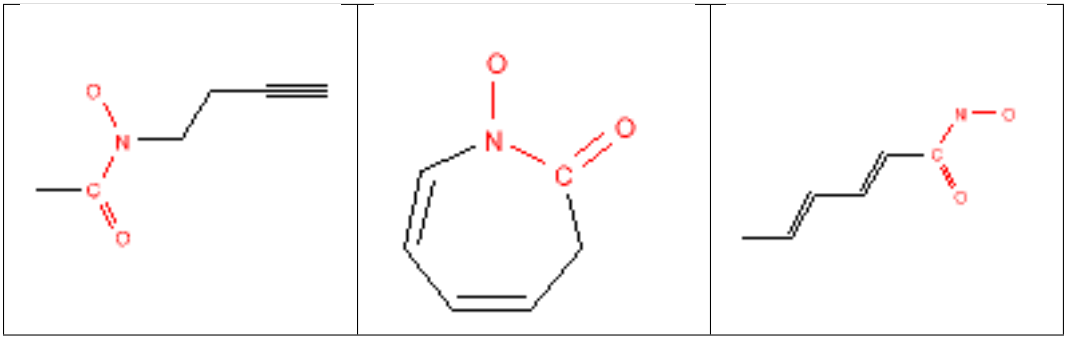
12.2.63 hydrazine



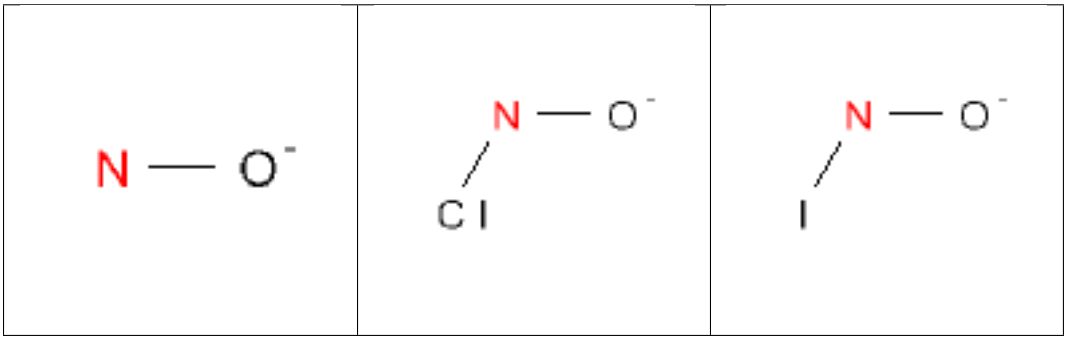
12.2.64 hydrazone



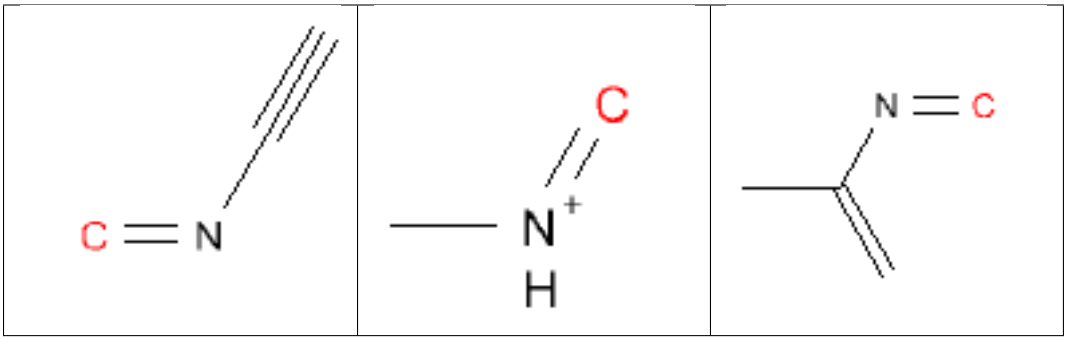
12.2.65 hydroxamic_acid



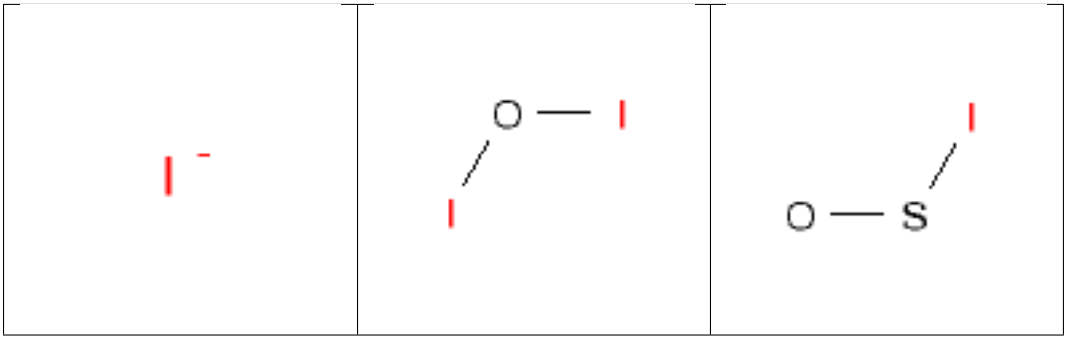
12.2.66 hydroxylamine



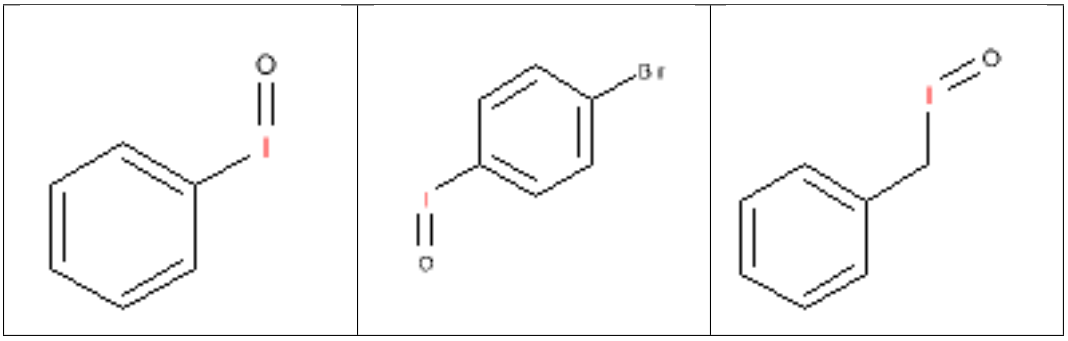
12.2.67 imine



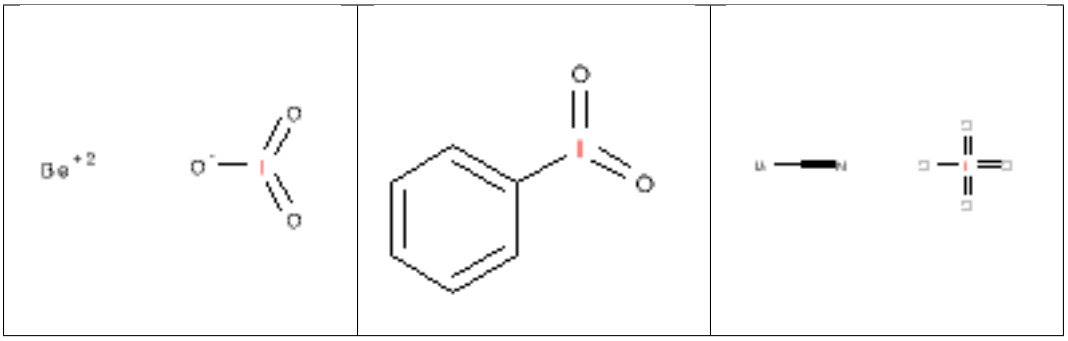
12.2.68 iodine



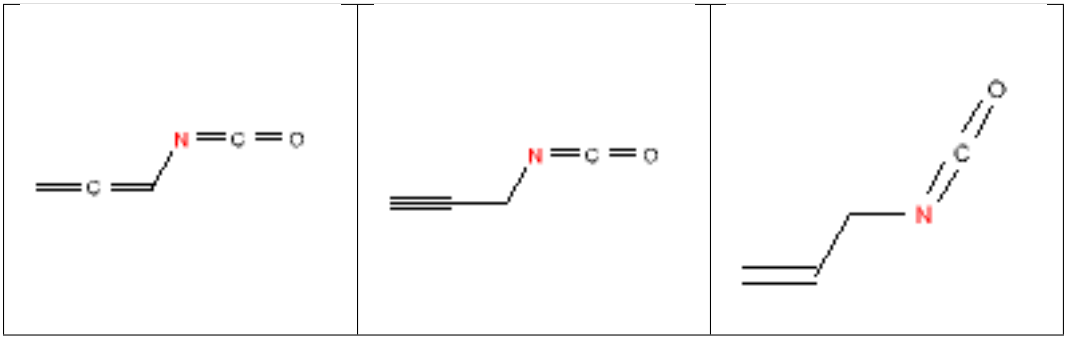
12.2.69 iodoso



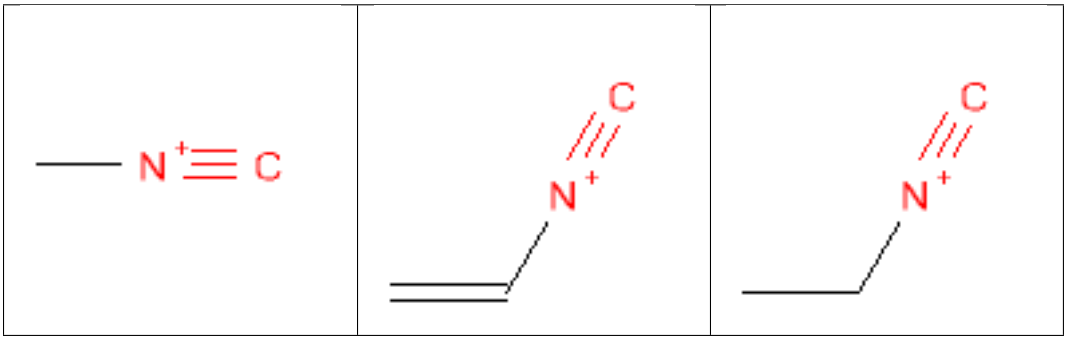
12.2.70 iodoxy



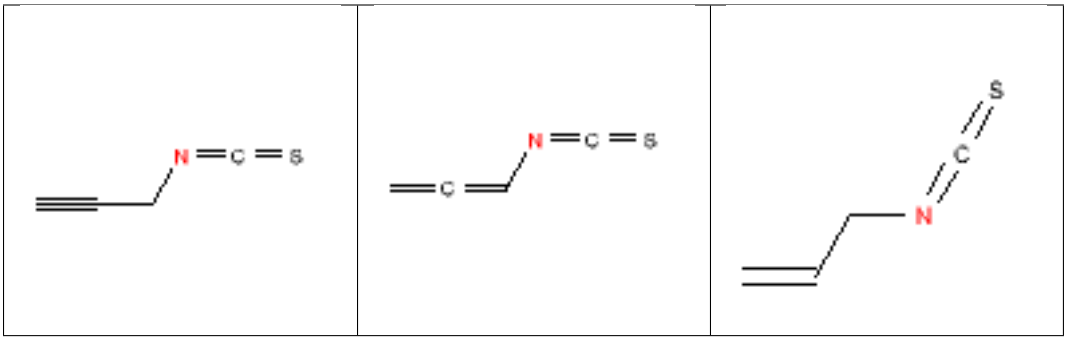
12.2.71 isocyanate



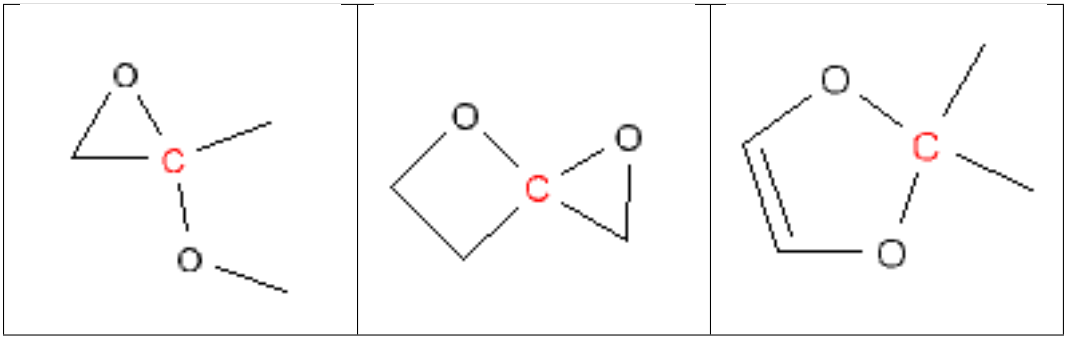
12.2.72 isonitrile



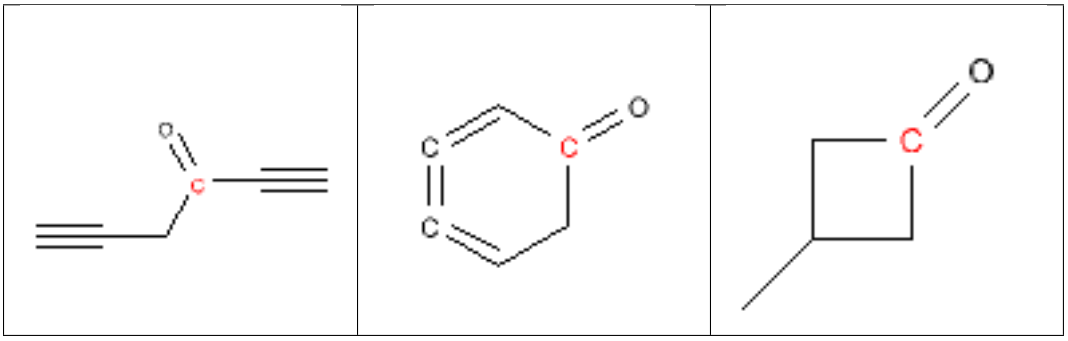
12.2.73 isothiocyanate



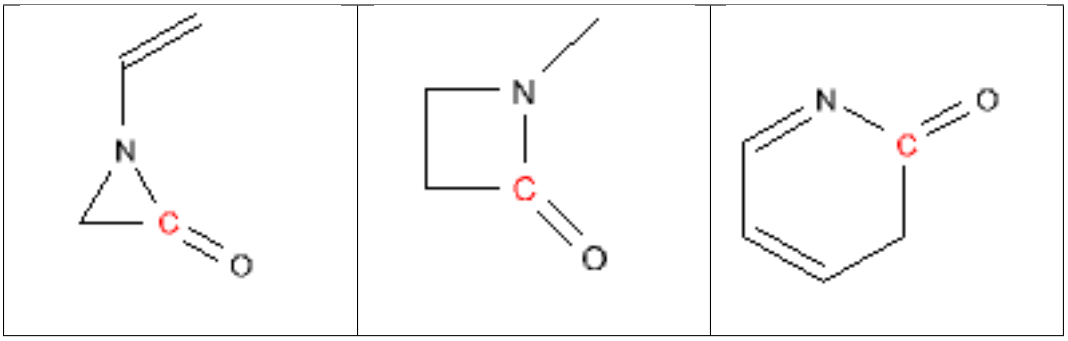
12.2.74 ketal



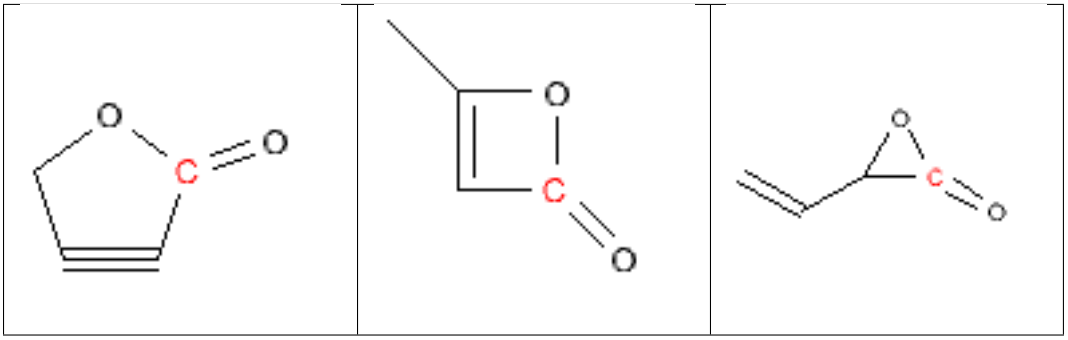
12.2.75 ketone



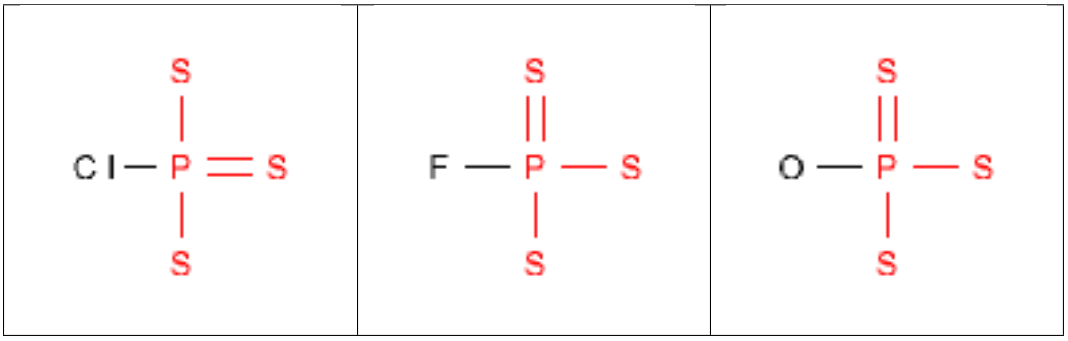
12.2.76 lactam



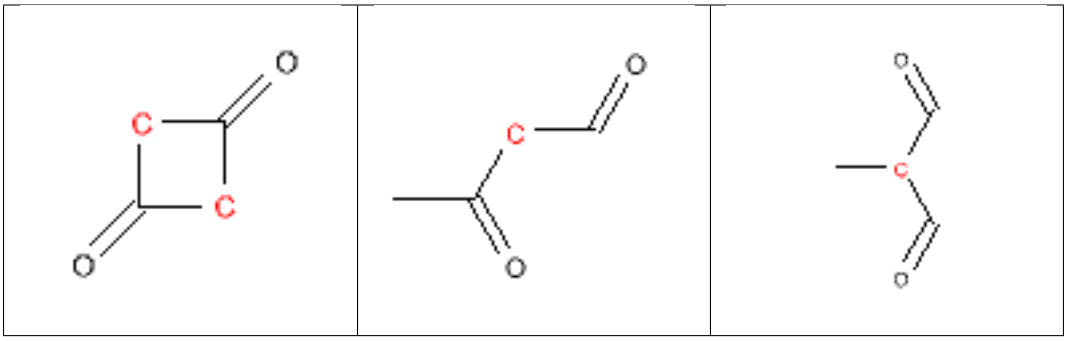
12.2.77 lactone



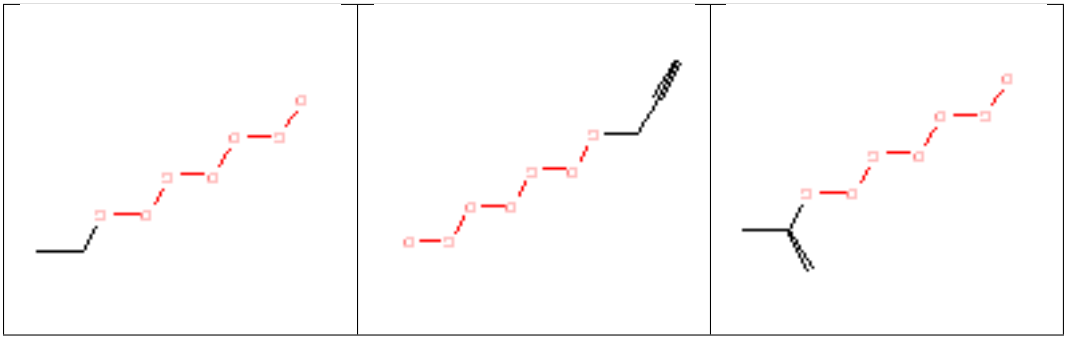
12.2.78 lawesson_s_reagent



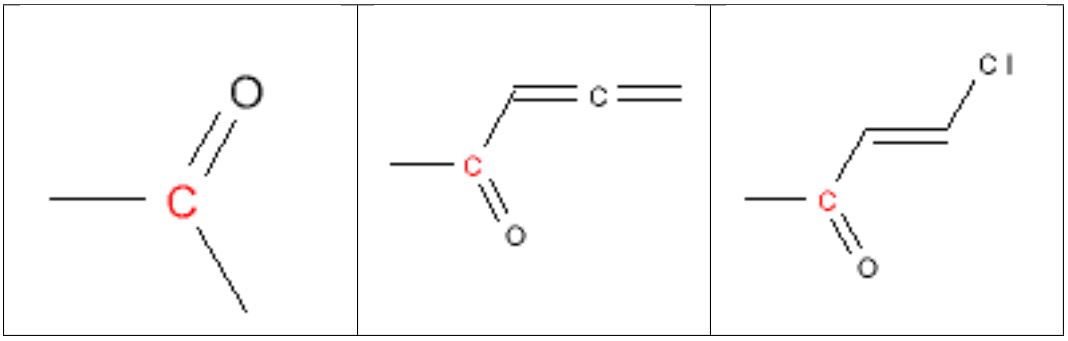
12.2.79 malonic



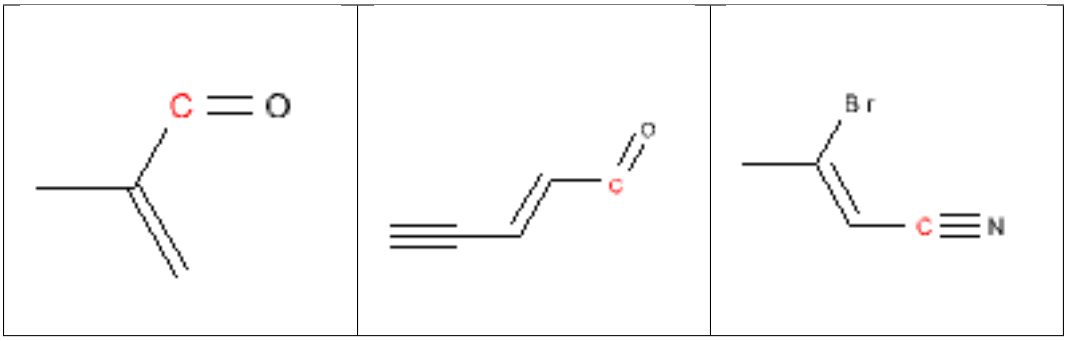
12.2.80 methoxyethoxymethyl_MEM



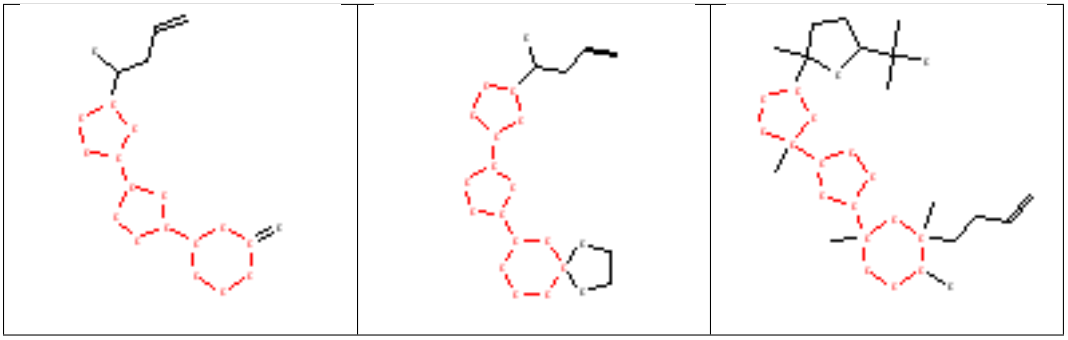
12.2.81 methyl_ketone



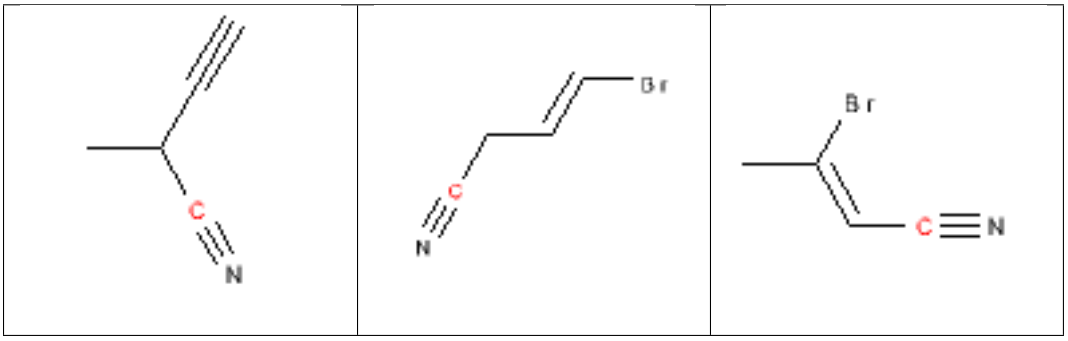
12.2.82 michael_acceptor



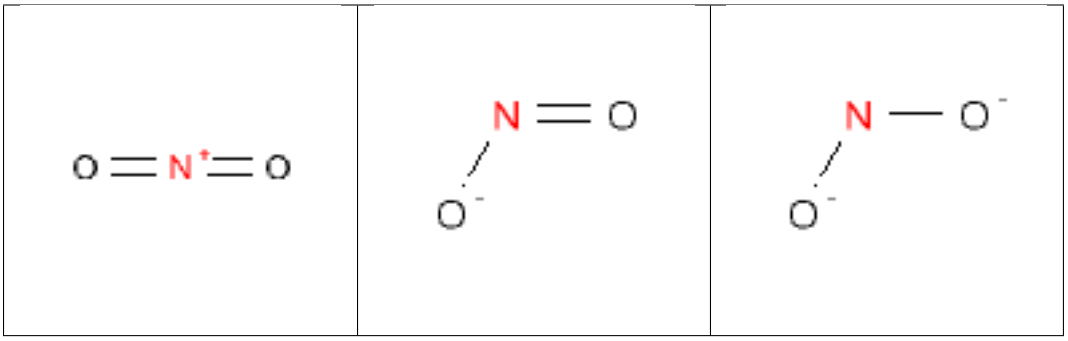
12.2.83 monensin_derivatives



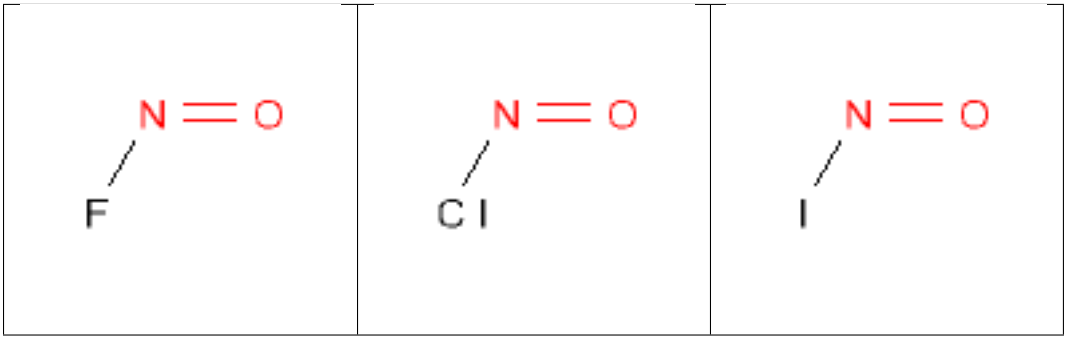
12.2.84 nitrile



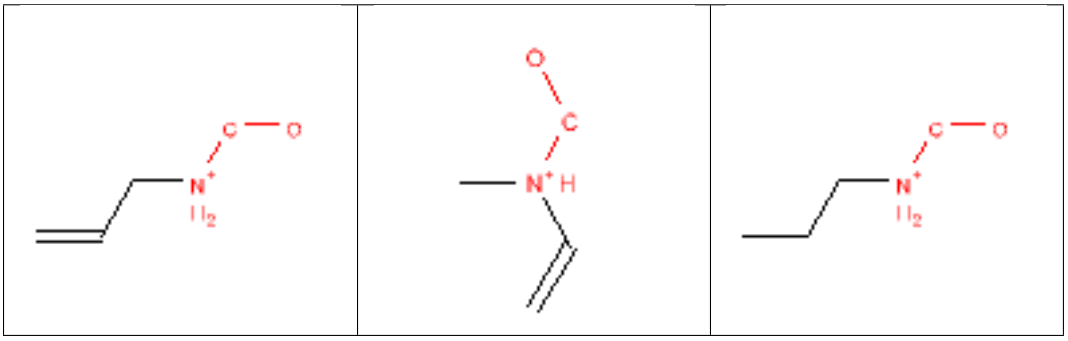
12.2.85 nitro



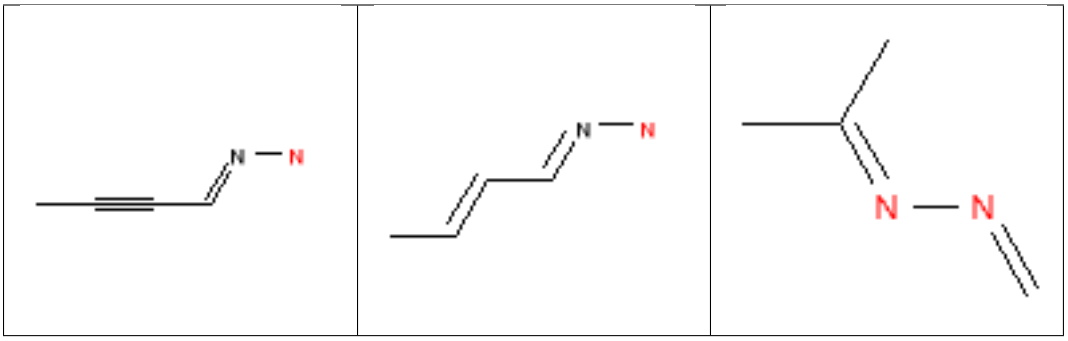
12.2.86 nitroso



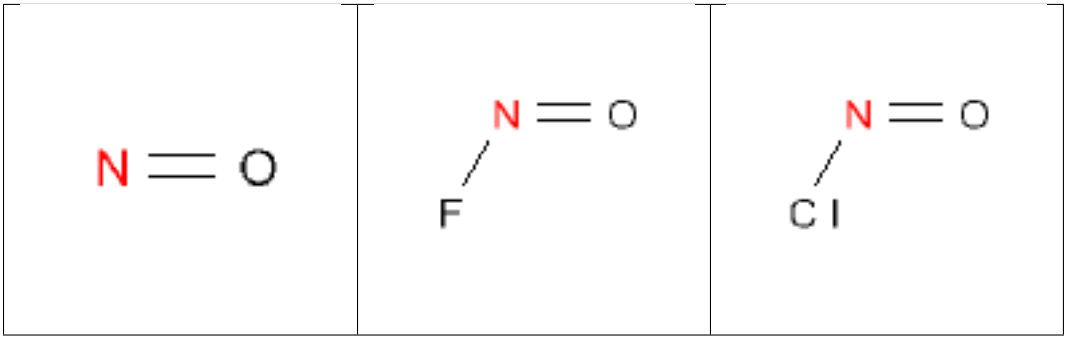
12.2.87 N_methoyl



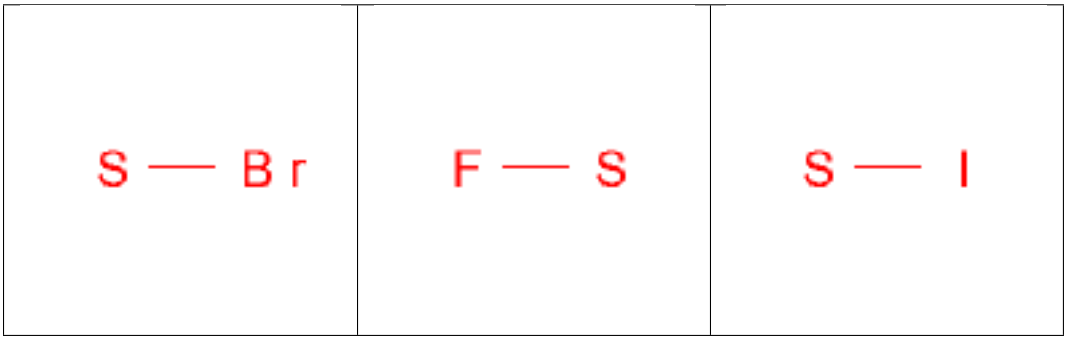
12.2.88 nonacylhydrazone



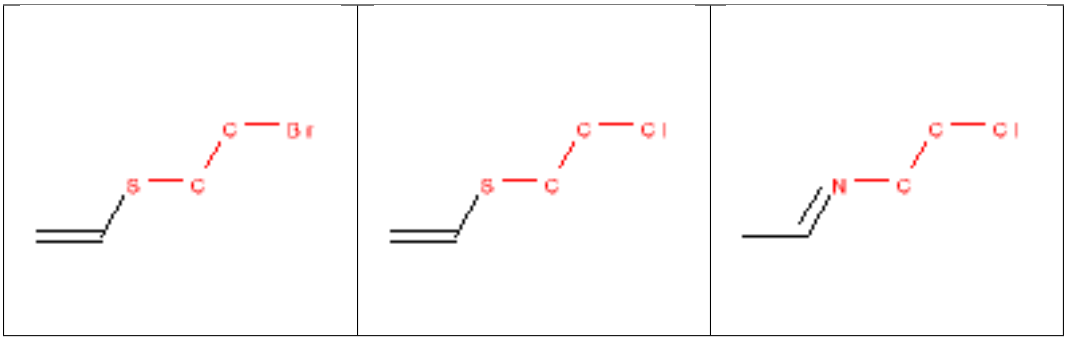
12.2.89 noxide



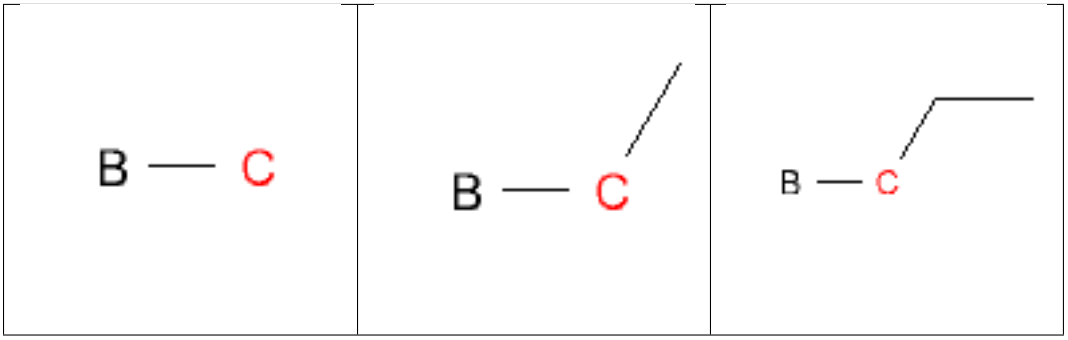
12.2.90 N_P_S_Halides



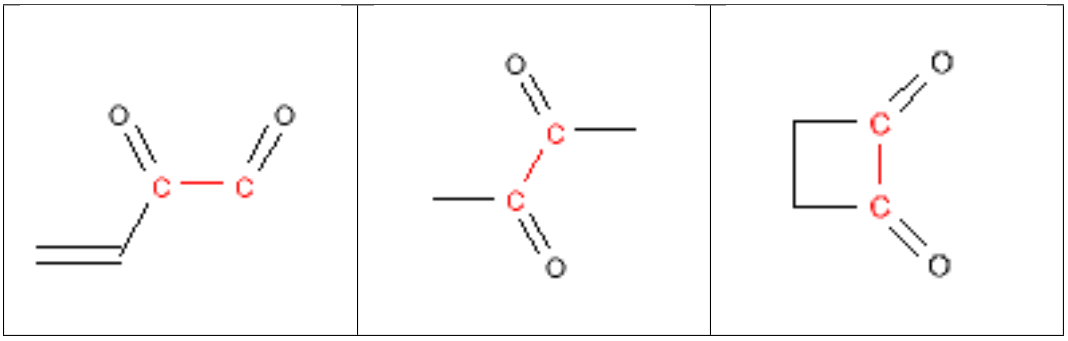
12.2.91 NS_beta_halothyl



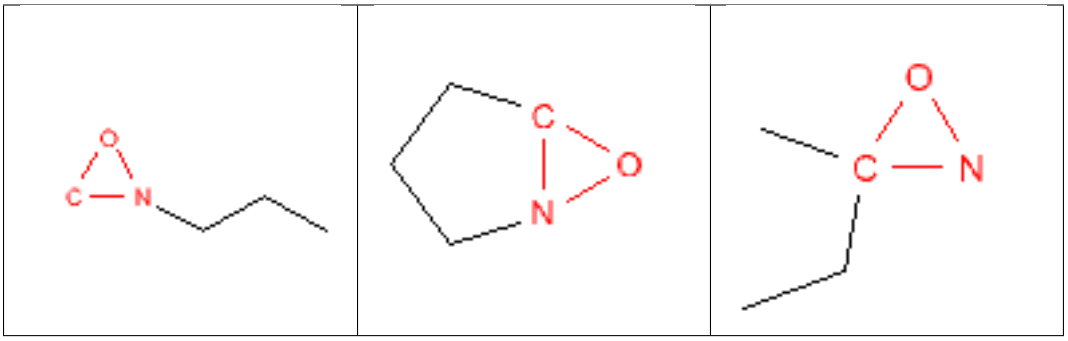
12.2.92 organometallic



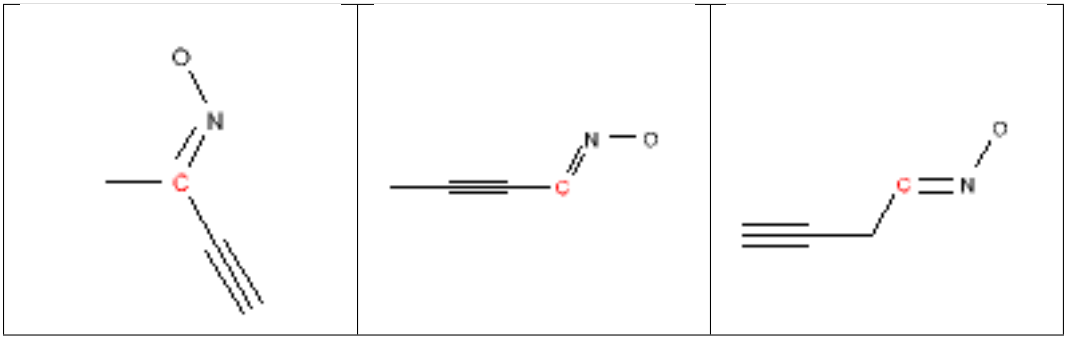
12.2.93 oxalyl



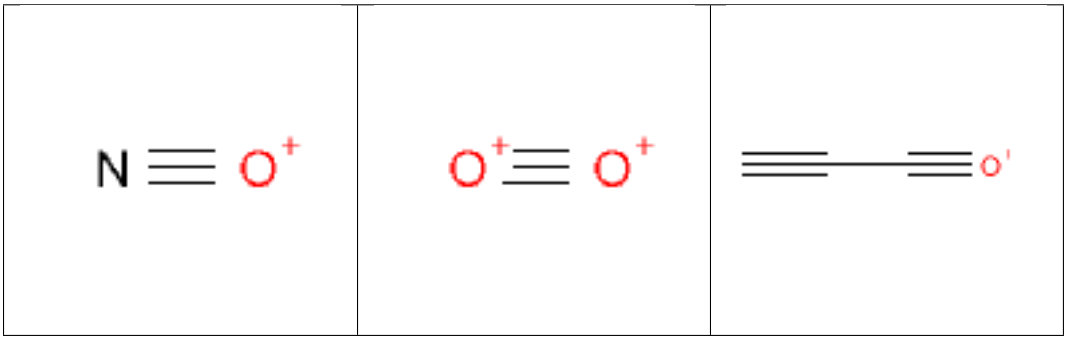
12.2.94 oxaziridine



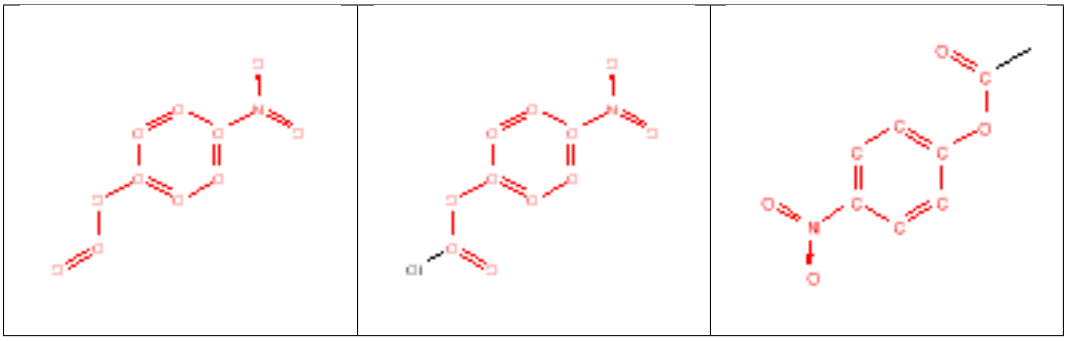
12.2.95 oxime



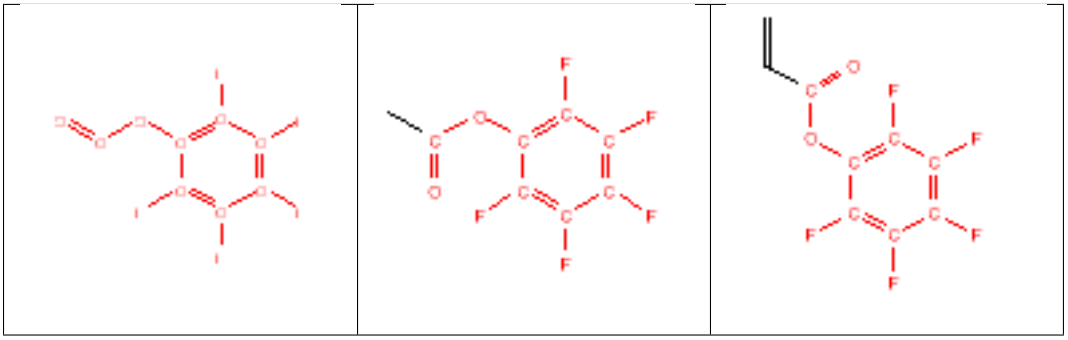
12.2.96 oxygen_cation



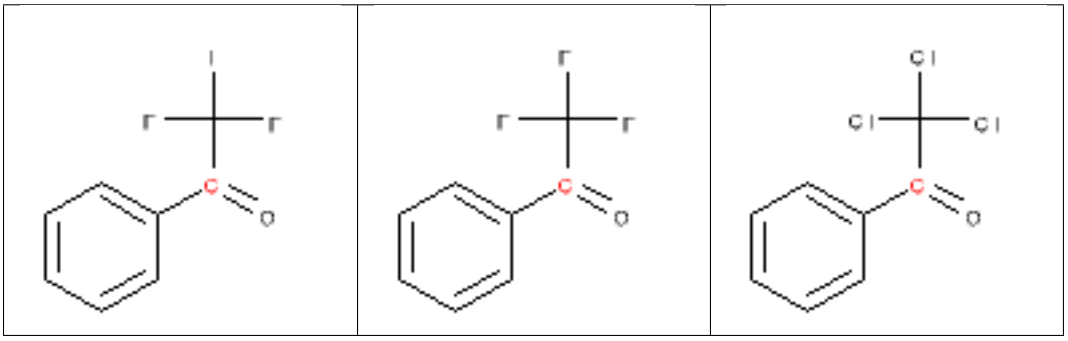
12.2.97 paranitrophenyl_esters



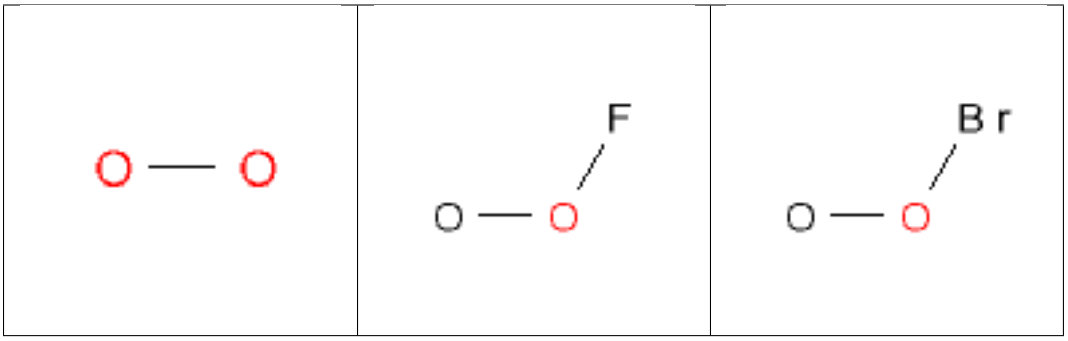
12.2.98 pentafluorophenyl_esters



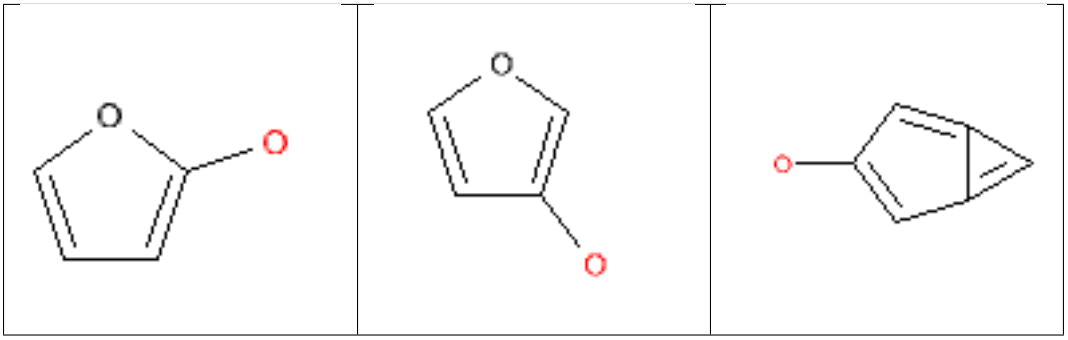
12.2.99 perhalo_ketone



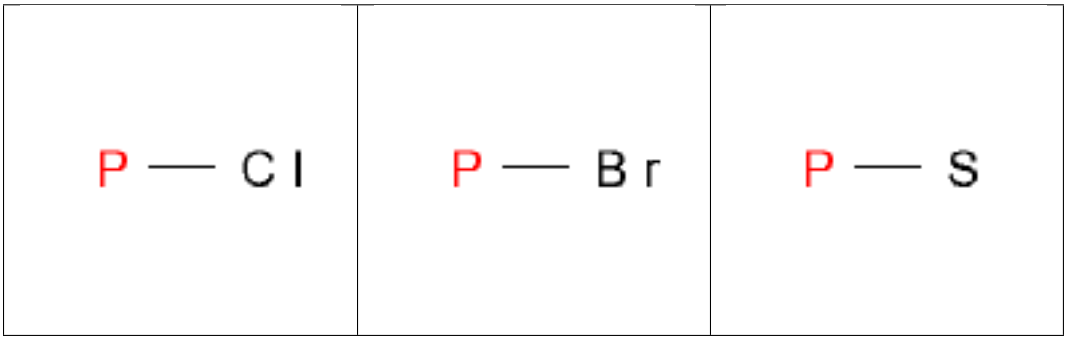
12.2.100 peroxide



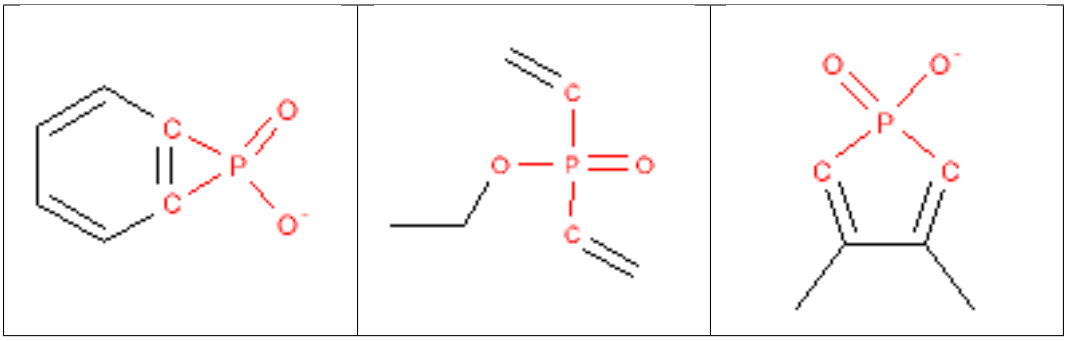
12.2.101 phenol



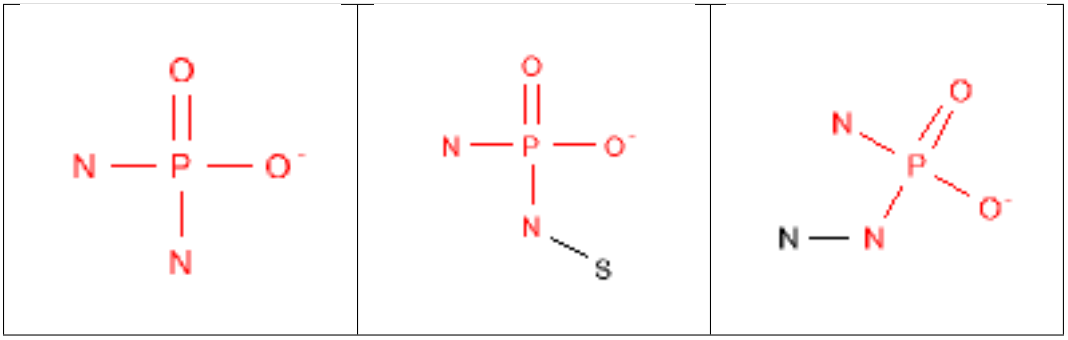
12.2.102 phosphanes



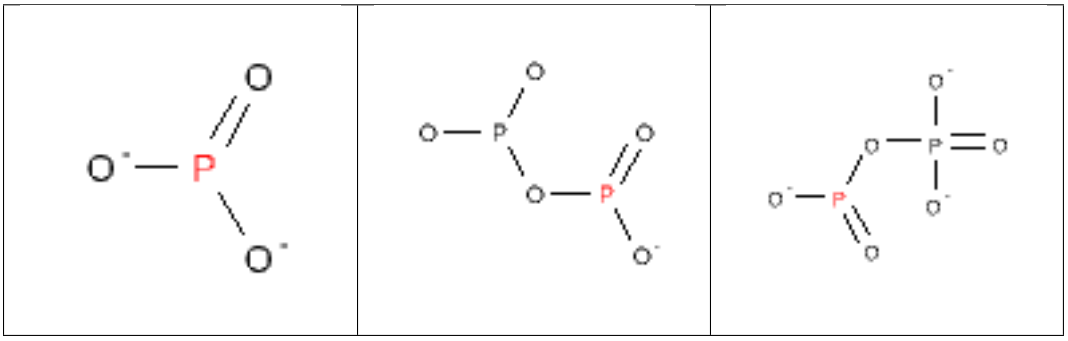
12.2.103 phosphinic_acid



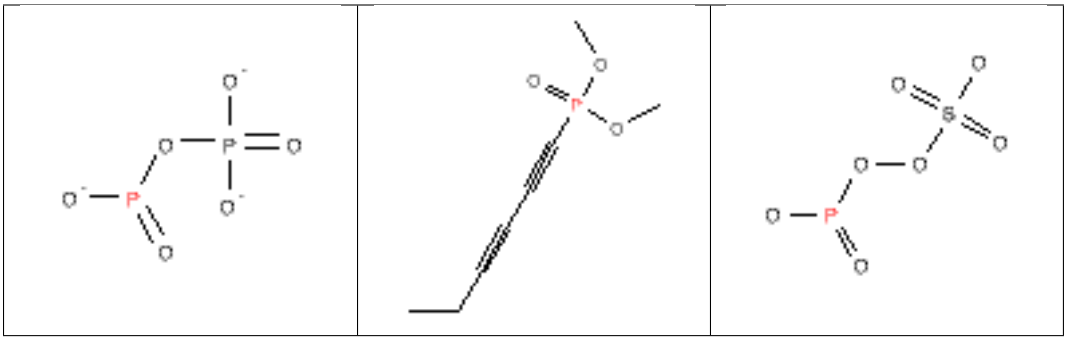
12.2.104 phosphonamide



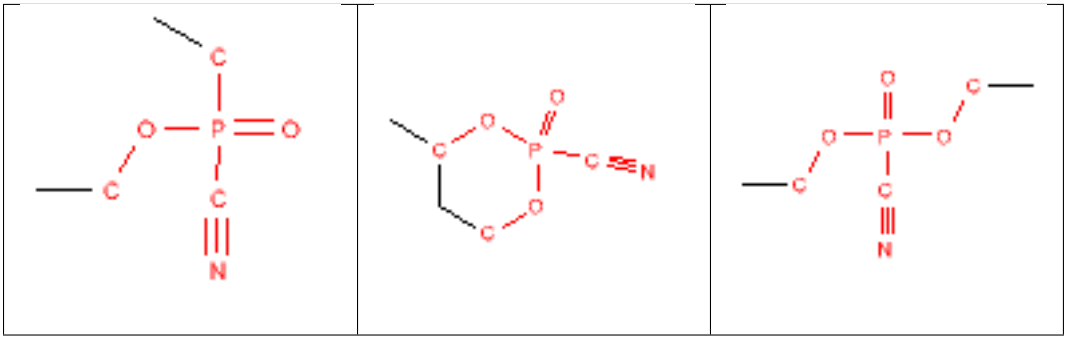
12.2.105 phosphonic_acid



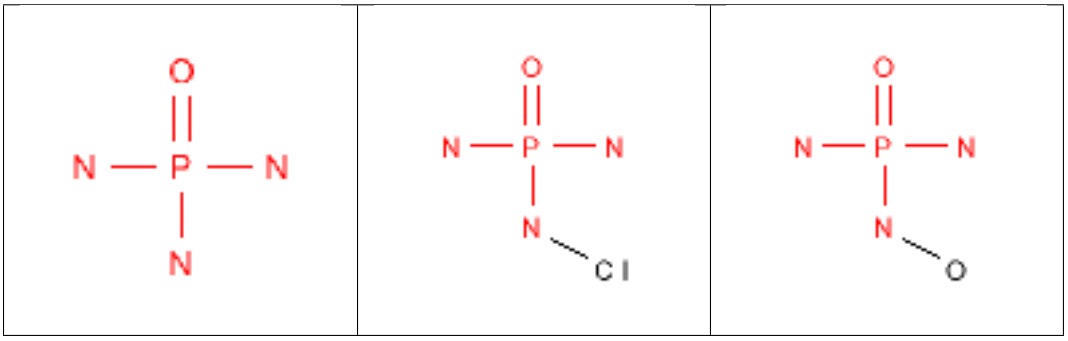
12.2.106 phosphonic_ester



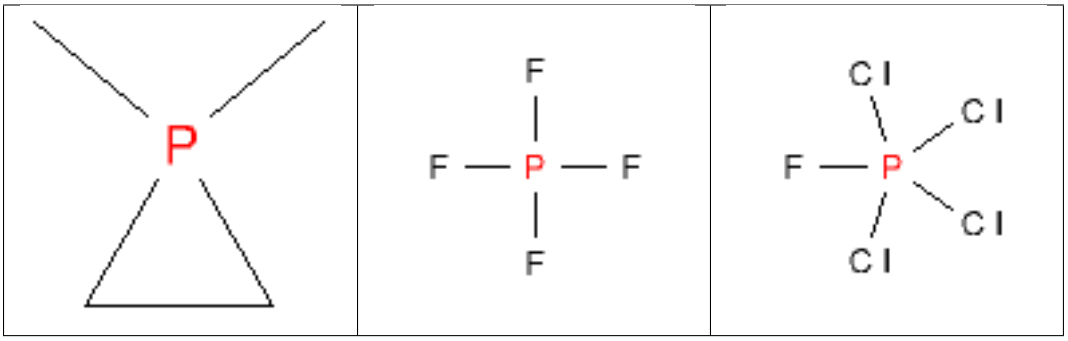
12.2.107 phosphonylnitrile



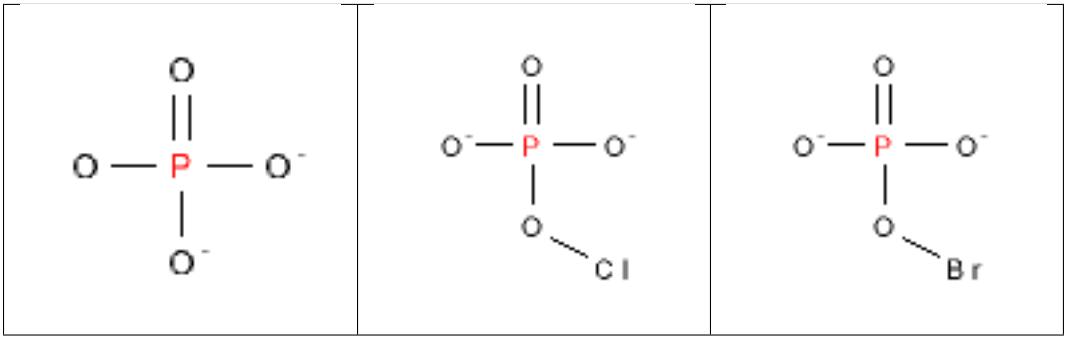
12.2.108 phosphoramides



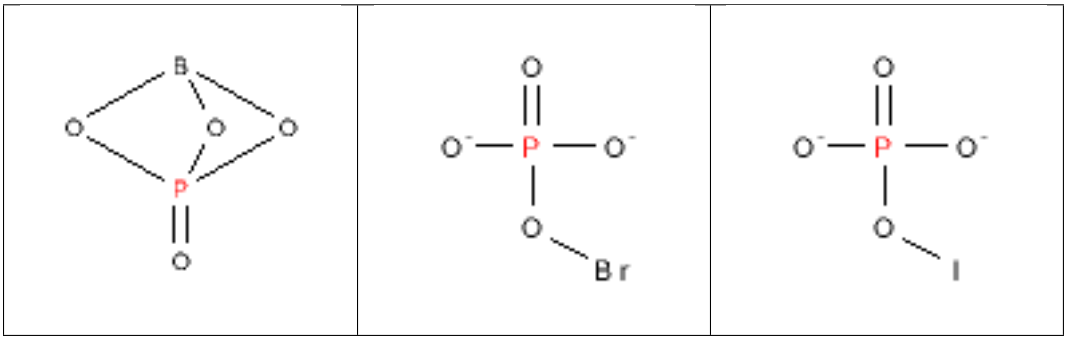
12.2.109 phosphoranes



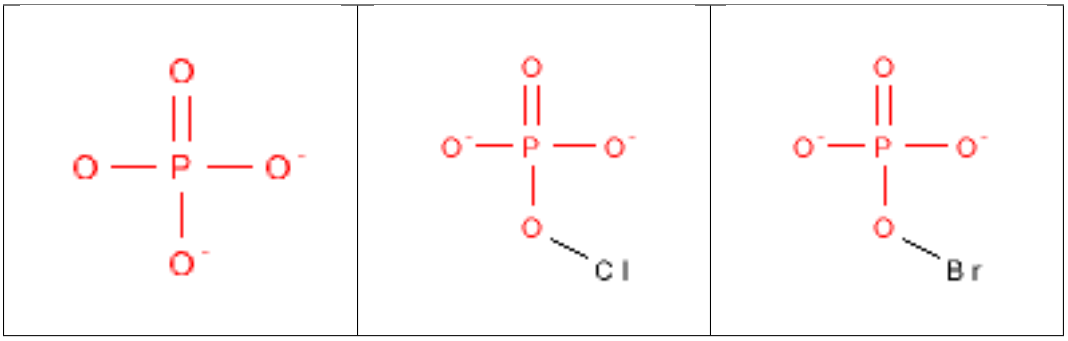
12.2.110 phosphoric_acid



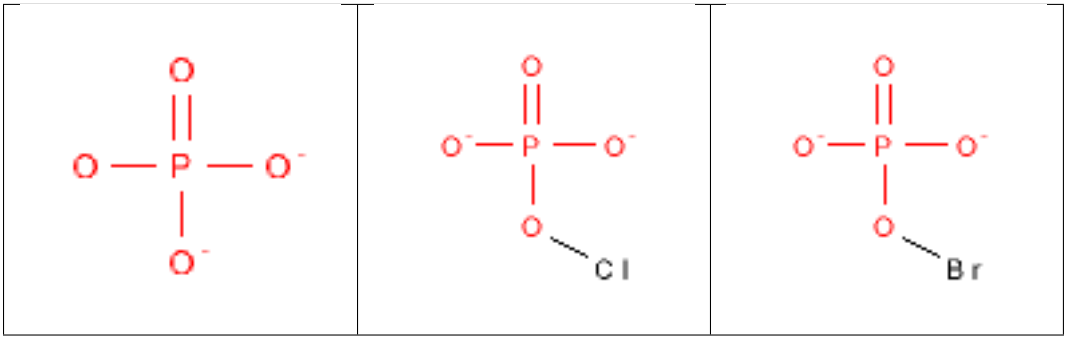
12.2.111 phosphoric_ester



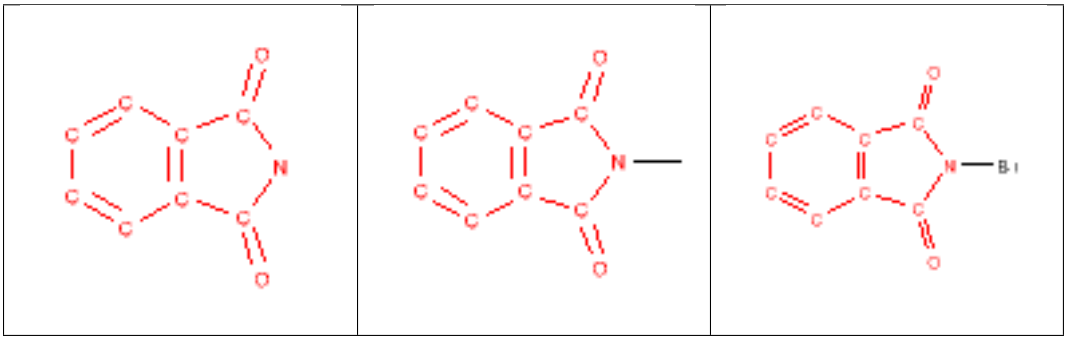
12.2.112 phosphoryl



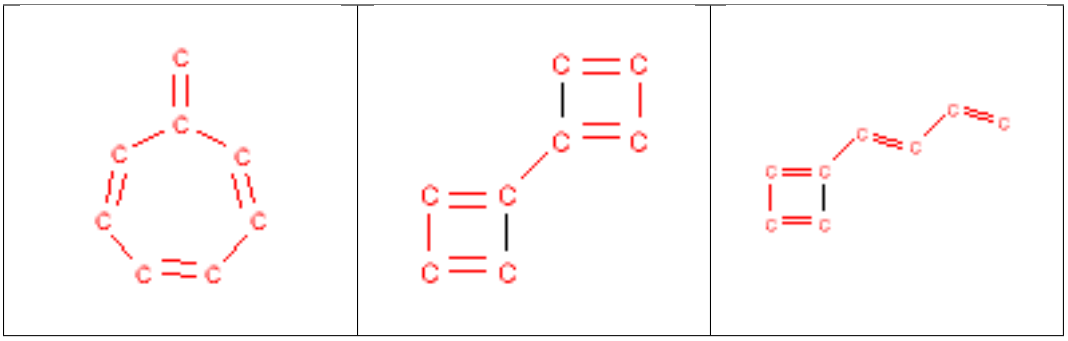
12.2.113 phosphoryl



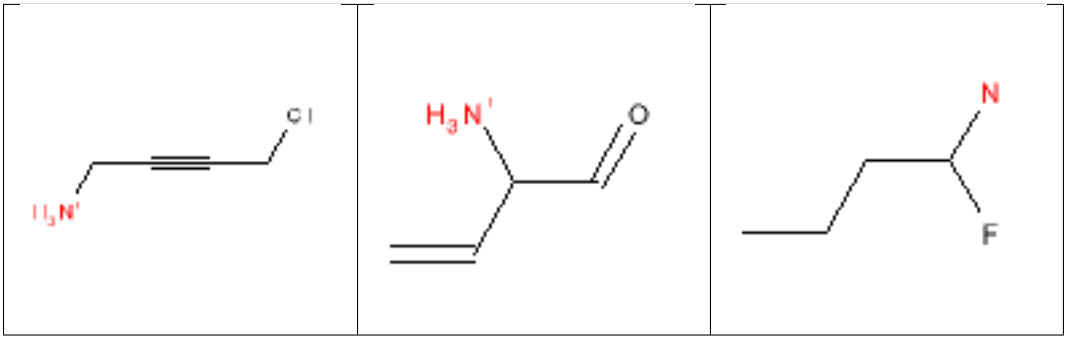
12.2.114 phthalimides_PHT



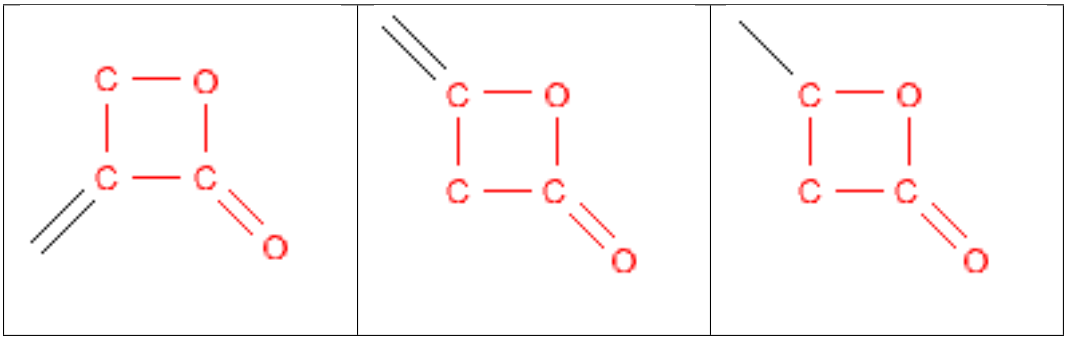
12.2.115 polyenes



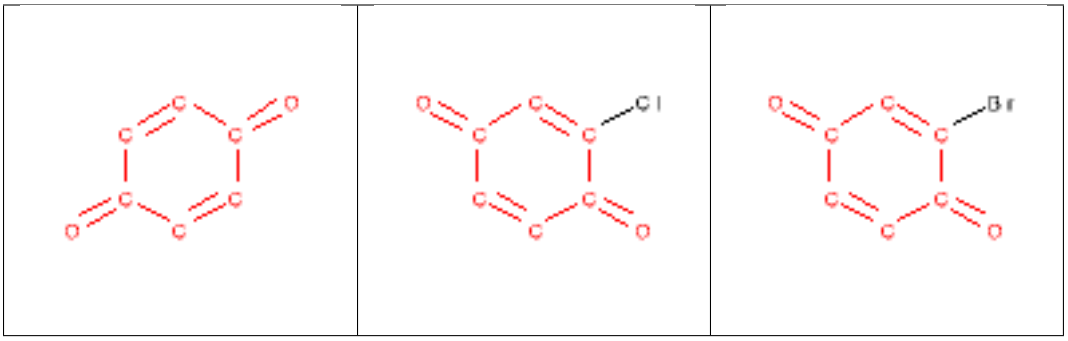
12.2.116 primary_amine



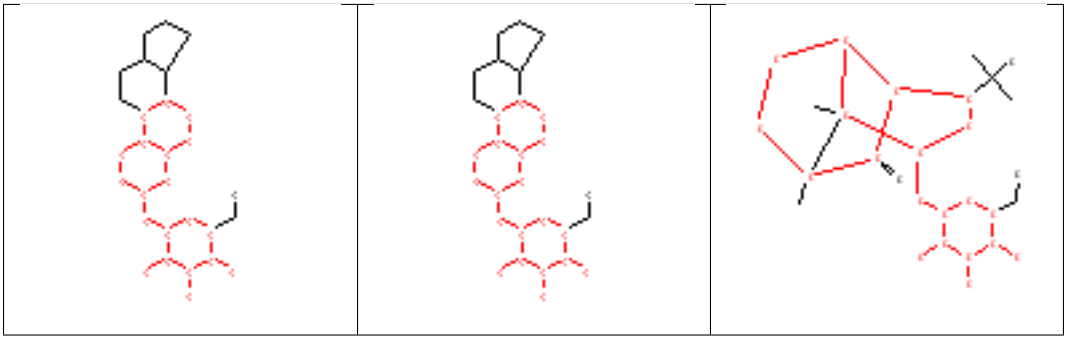
12.2.117 propiolactones



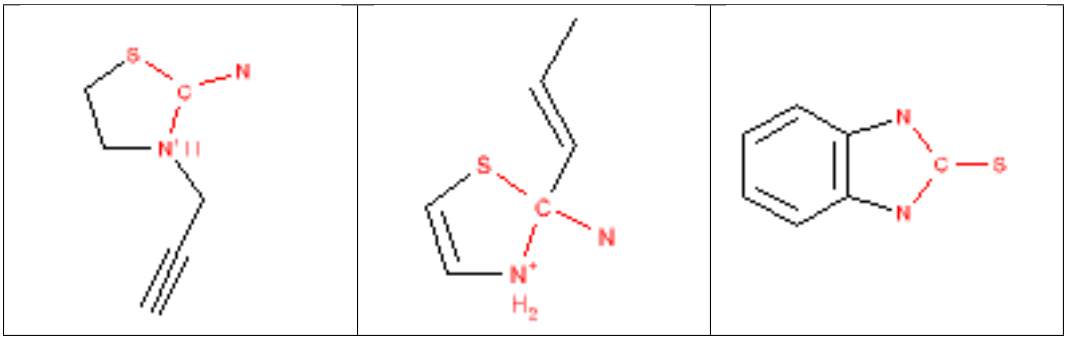
12.2.118 quinone



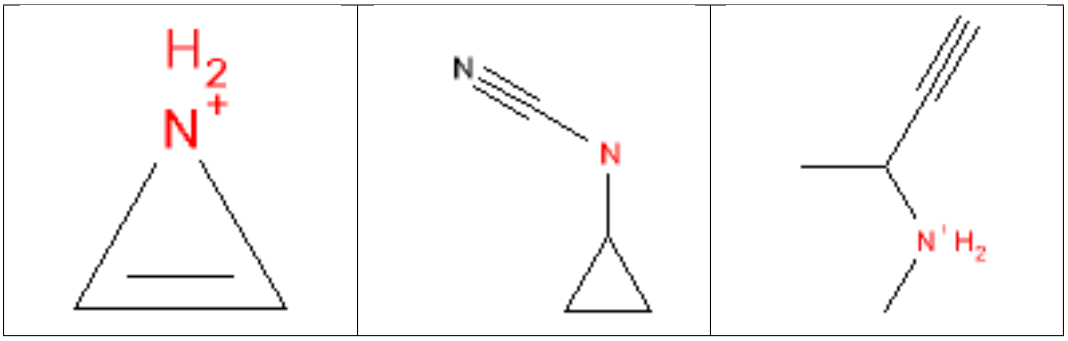
12.2.119 saponin_derivatives



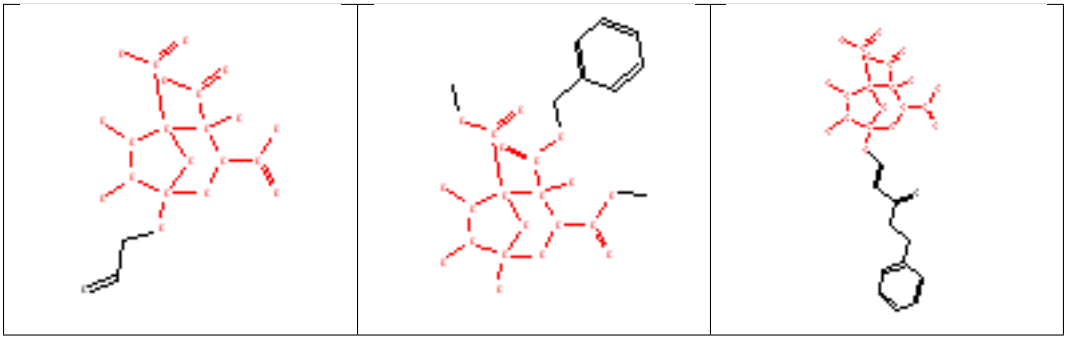
12.2.120 SCN2



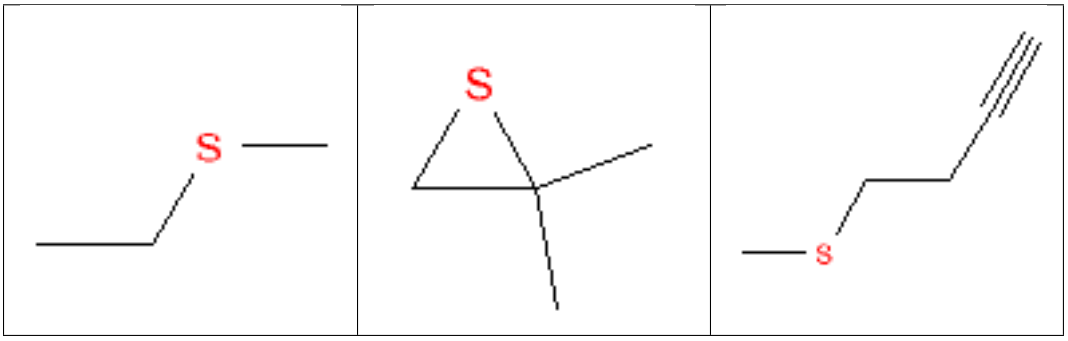
12.2.121 secondary_amine



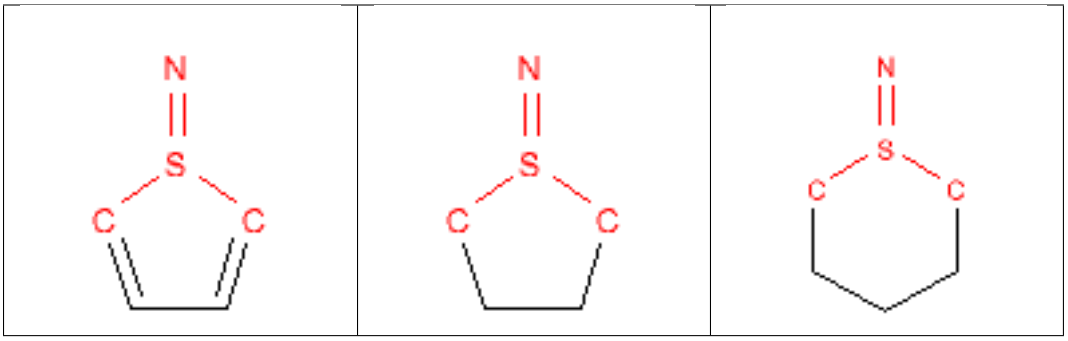
12.2.122 squalestatin_derivatives



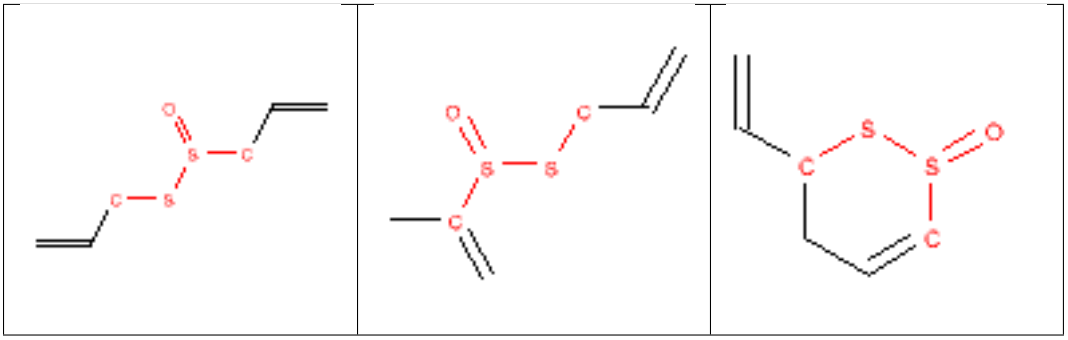
12.2.123 sulfide



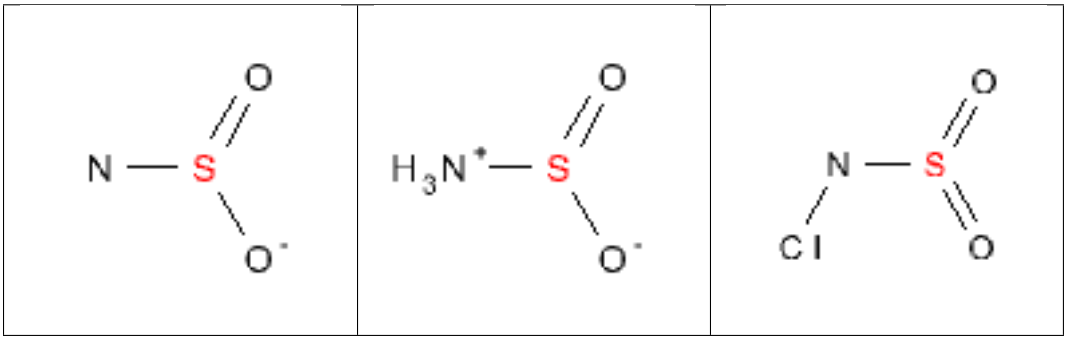
12.2.124 sulfinimine



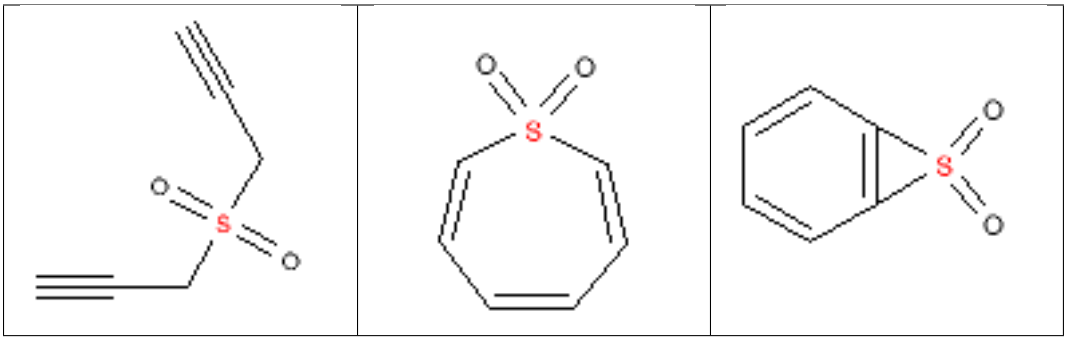
12.2.125 sulfinylthio



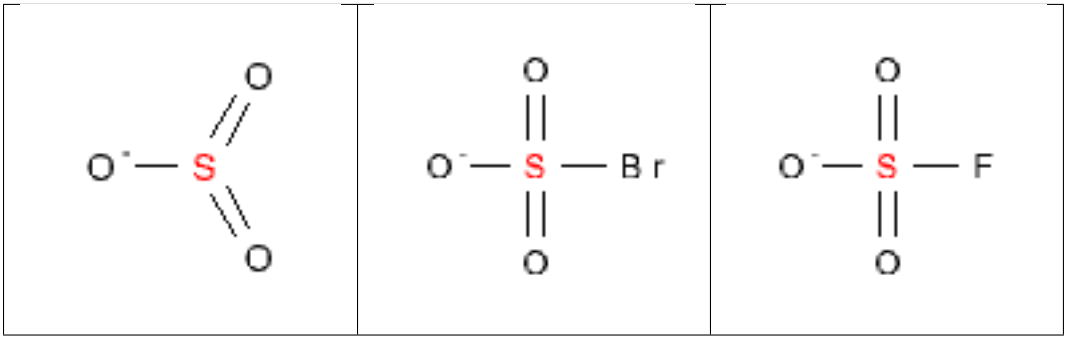
12.2.126 sulfonamide



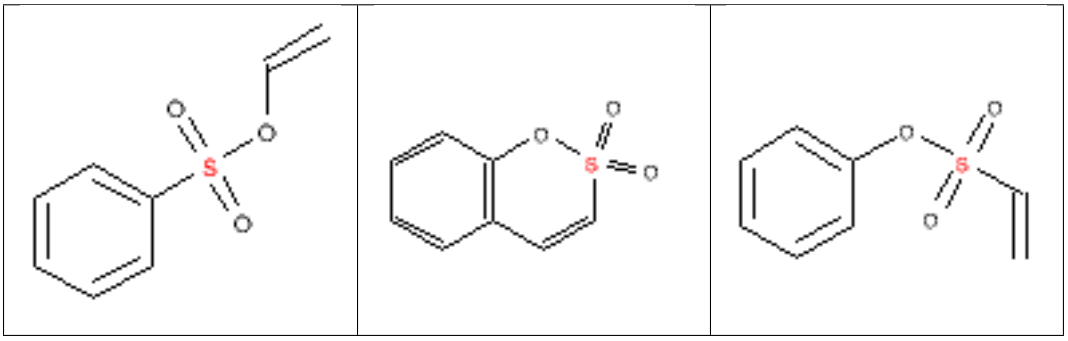
12.2.127 sulfone



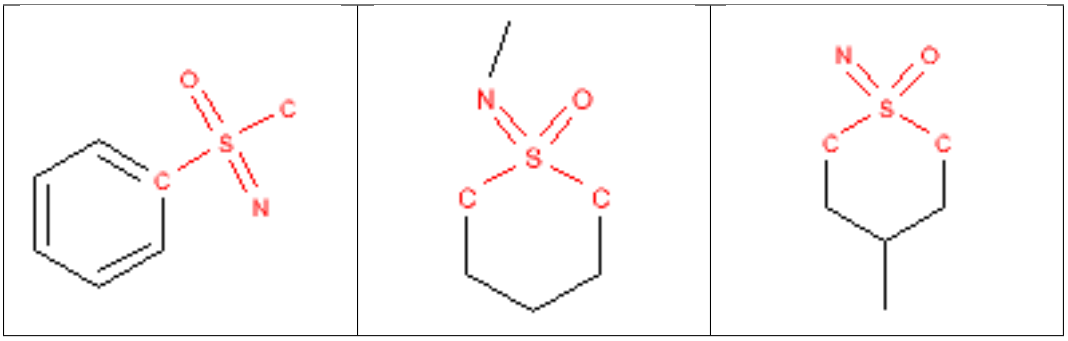
12.2.128 sulfonic_acid



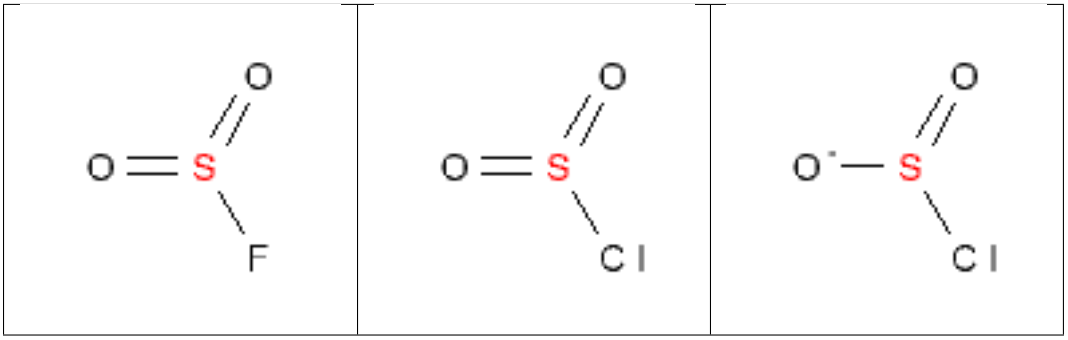
12.2.129 sulfonic_ester



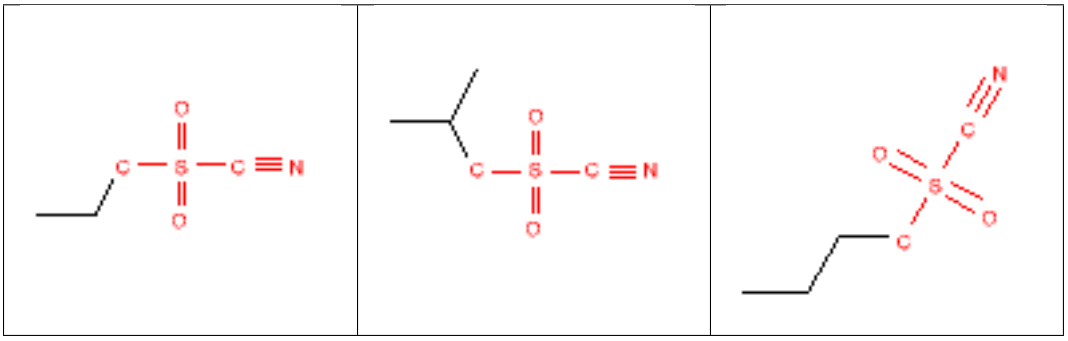
12.2.130 sulfonimine



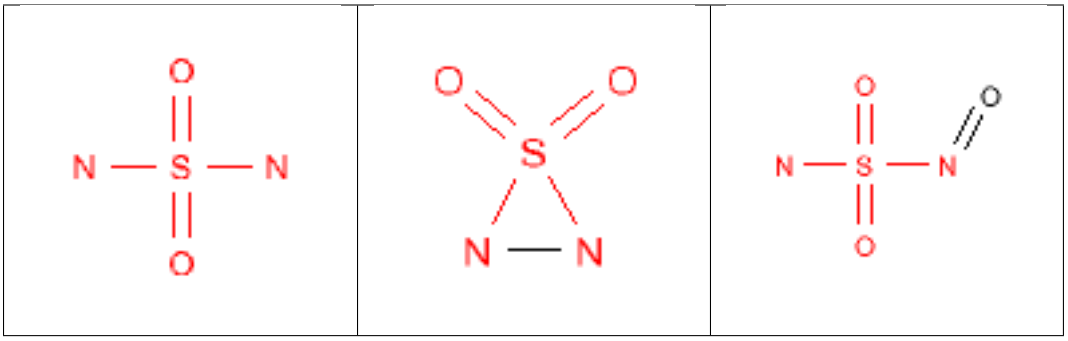
12.2.131 sulfonyl_halide



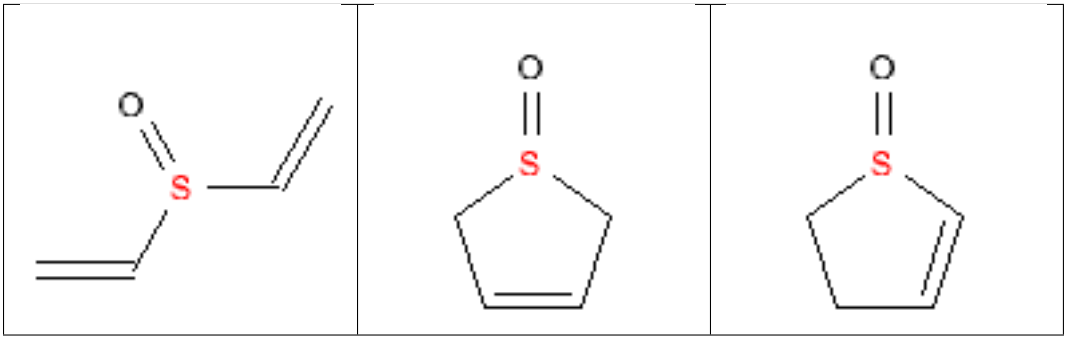
12.2.132 sulfonylnitrile



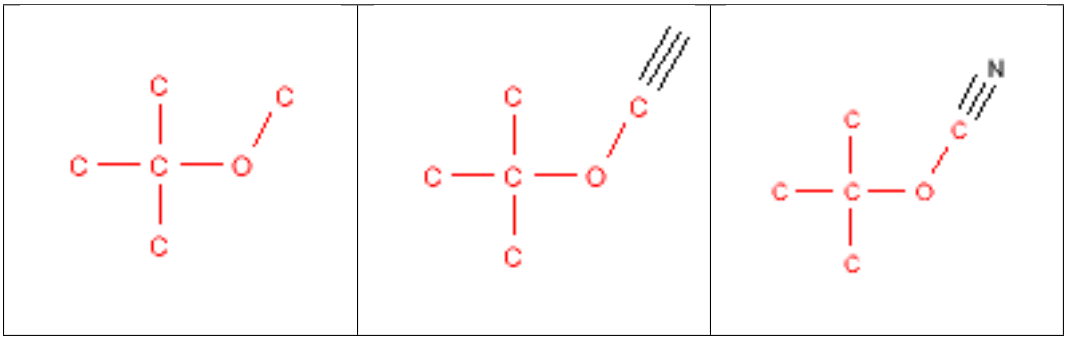
12.2.133 sulfonylurea



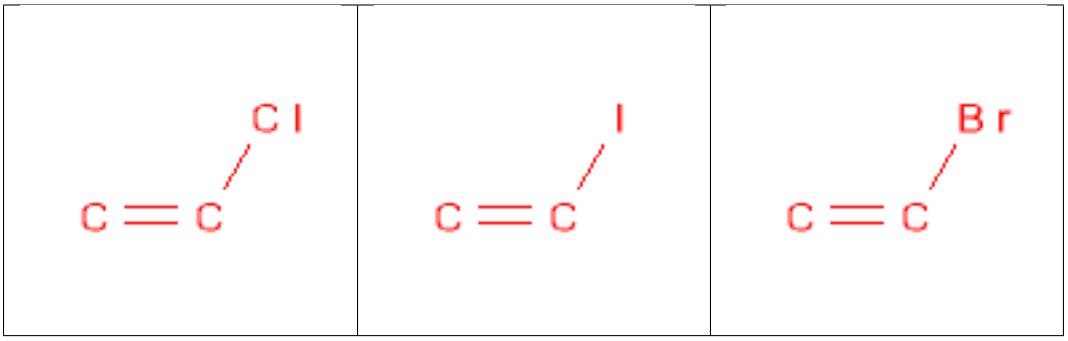
12.2.134 sulfoxide



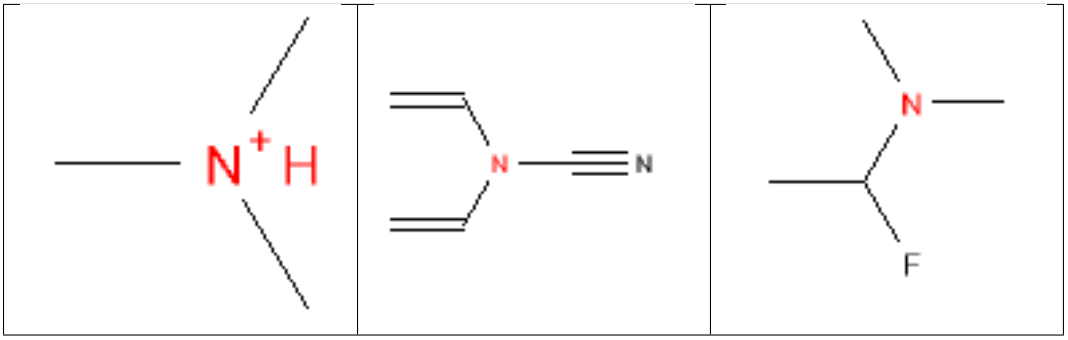
12.2.135 t_butyl_ether



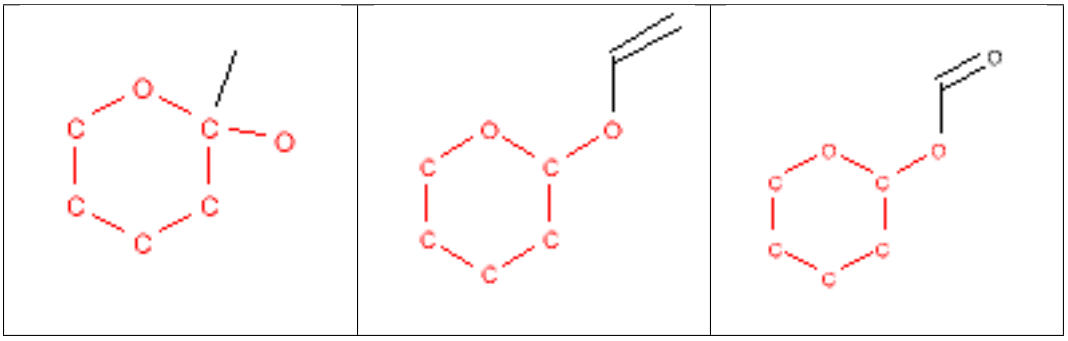
12.2.136 terminal_vinyl



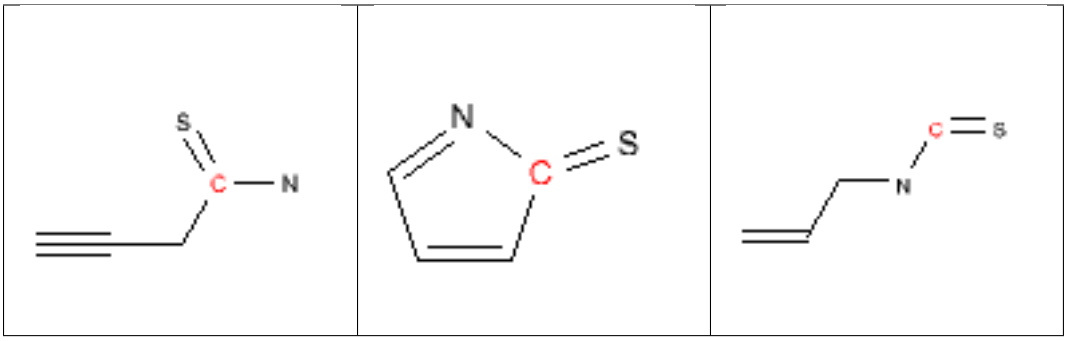
12.2.137 tertiary_amine



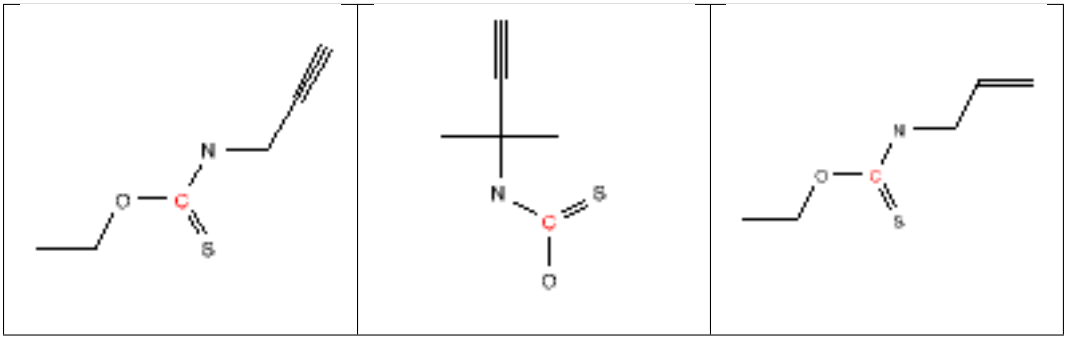
12.2.138 tetrahydropyran_THP



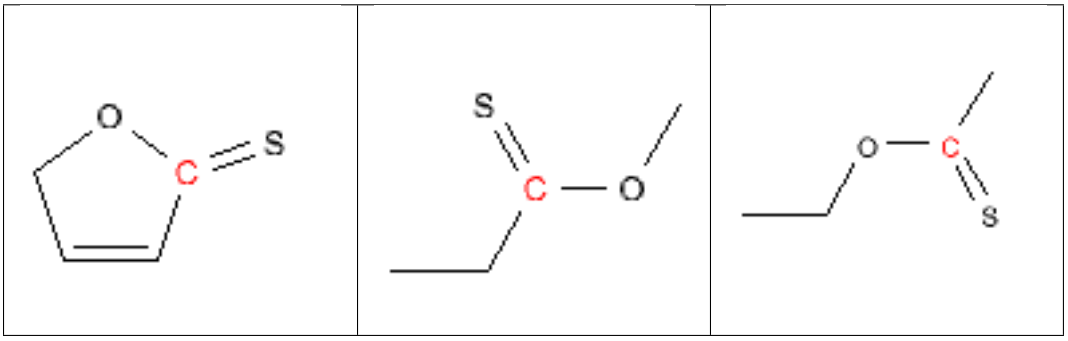
12.2.139 thioamide



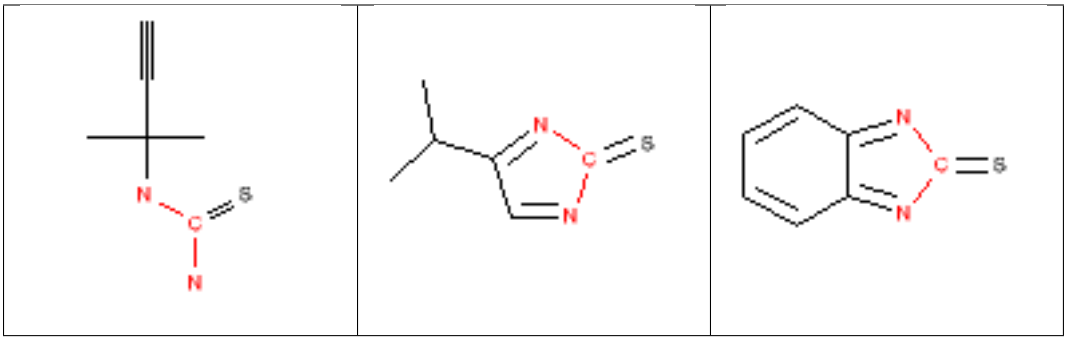
12.2.140 thiocarbamate



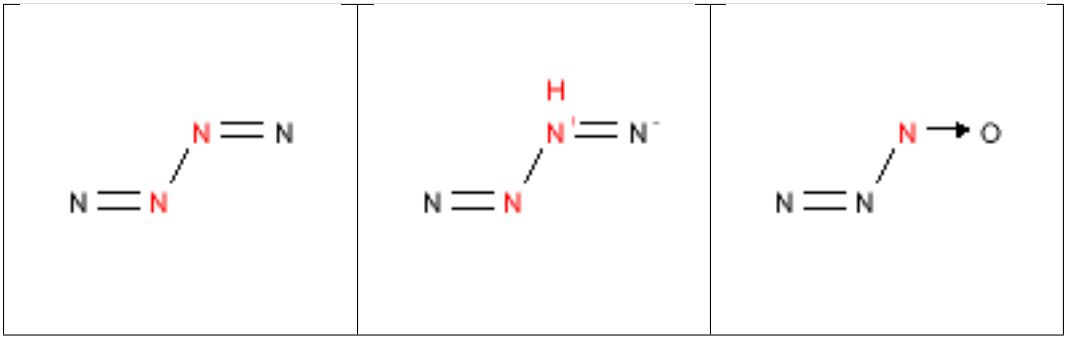
12.2.141 thioester



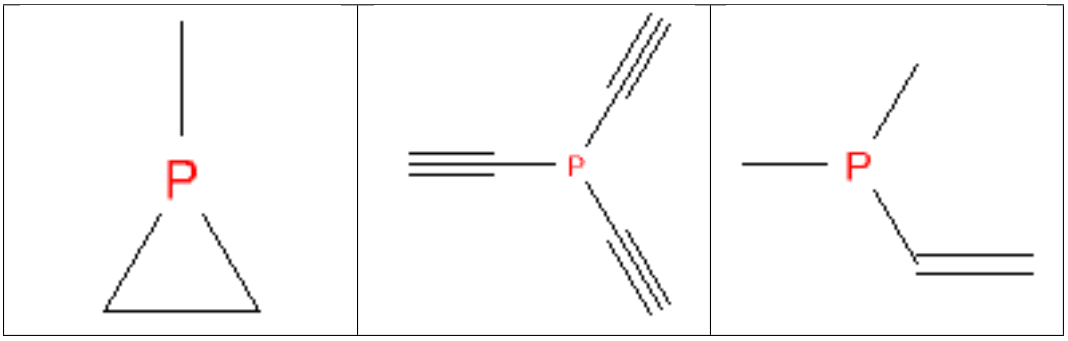
12.2.142 thiourea



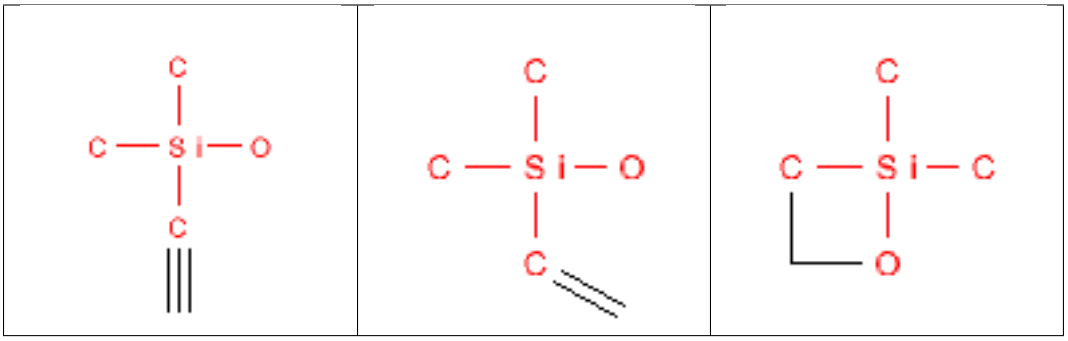
12.2.143 triazine



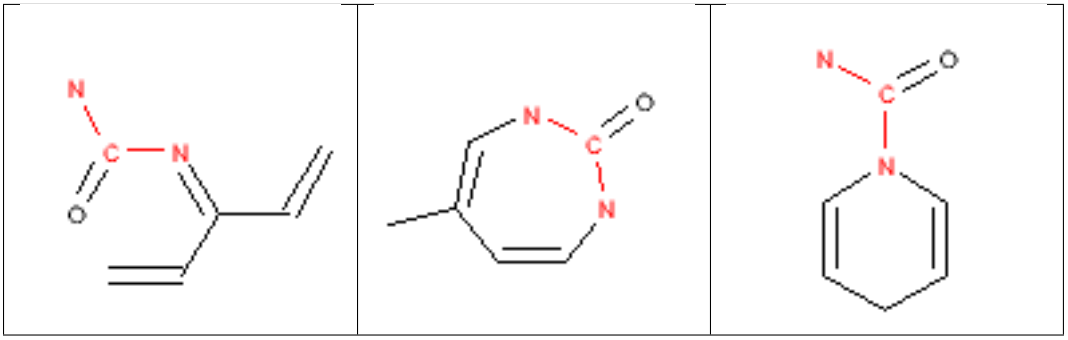
12.2.144 tricarbo_phosphene



12.2.145 trimethylsilyl_TMS



12.2.146 urea



12.3 New Rules

New rules specify additional functional groups or substructures that may be used. They must specify a substructure definition in the form of a SMARTS in addition to the substructure name and maximum limit. For example:

```
NEWRULE norbornane 1 C1CC2CCC1C2
```

The first field is the `NEWRULE` keyword. The second field defines the name associated with the substructure (primarily for logging purposes). The third field indicates the maximum number of the substructure that can be allowed. The fourth field is the SMARTS string for the substructure, norbornane in this case. This example rule would indicate that molecules with a single norbornane substructure would be allowable, but that those with 2 or more norbornanes would be eliminated.

New rules that have a name that is identical with one of the original rules take precedence over the original rule.

12.4 Selection Statements

The select statement allows a filter file to specify the required number of substructures in order to be able to pass the filter. These statements are similar to new rules except that they list a required range for passing the filter rather than the range for failing to pass the filter. For example:

```
SELECT amine 1 1 [N;!$(*-*[!#6;!#1]);!$(*-a);!$(*=,#*)]
```

The first field is the `SELECT` keyword. The second field indicates the name for the selection (again for logging purposes). The third field is the minimum number of substructures required to be in the molecule. The fourth field is the maximum number of substructures allowed in the molecule. The fifth field is the substructure defined by a SMARTS pattern. The example requires that molecules contain exactly one amine. Currently, only a single `SELECT` statement is allowed in the filter file. Any complex boolean substructure statements can be incorporated directly into the SMARTS. If multiple `SELECT` statements occur in a filter file, only the final one will be applied.

CHAPTER

THIRTEEN

RELEASE NOTES

13.1 OMEGA 2.5.1 *(May 2013)*

13.1.1 New features

- FILTER is now completely bundled into the OMEGA package. The manuals have been consolidated into this single manual and FILTER has taken on the OMEGA version number.
- An option has been added to allow hydrogen atoms in -OH, -SH, and amines to take part in conformational sampling. This new option can be enabled via the *-sampleHydrogens* parameter. By default, hydrogen atoms are not sampled. [Case #13115]
- A *-warts* option has been added to *flipper*. [Case #10655]

13.1.2 Major bug fixes

- Now using *-fixsmarts* without *-fixmol* will rematch for every input structure. Previously, this would only match the first input structure and reuse that match for the rest of the calculation. Using both *-fixsmarts* and *-fixmol* will continue to match against the fixmol and use that match for the entire calculation. [Case #14518]
- Partial ring matching with *-fixsmarts* or *-fixmol* could cause a crash. Now only full ring systems may be specified and any attempt at matching a partial ring system will fail gracefully.
- Planar molecules could have problems sprouting hydrogens when using *-fromCT false* because OMEGA was treating them as 2D. Now OMEGA will assume that planar molecules are properly defined 3D coordinates in this situation. [Case #15737]
- Stereochemistry at the attachment points of rings with four or less atoms could be lost. Now stereochemistry is saved for rings of all sizes. [Case #3553]
- A bug has been fixed where exceptionally long torsion rules could get cutoff early. [Case #16115]
- A bug has been fixed where older TK licenses in the license file could cause the OMEGA application to fail. [Case #13466]

13.1.3 Minor bug fixes

- Only 3D coordinates were being cleared in FLIPPER. Now 2D coordinates are cleared as well. [Case #13378]
- Fragment generation during an OMEGA calculation with *strictfrags* parameter enabled could yield slightly different fragments than the MAKEFRAGLIB app. This has been corrected for consistency. [Case #13115]

- Tags for rotor compression are no longer attached to the output molecules when rotoroffsetcompress is false. It was discovered that these tags confuse VIDA when they are attached but not used. [Case #14315]
- The torsion library has been updated with better angles for a biphenyl torsion. The entry for 90 degrees has been removed and +/-30 degrees has been added.
- FILTER has been fixed to no longer have an atom order dependence when setting a pH model. [Case #14368]

13.1.4 Other changes

- PVM (parallel virtual machine) is no longer supported. OpenMPI version 1.6 is supported on all plafforms. The *-mpi_np* and *-mpi_hostfile* flags are now used to run OMEGA and *makefraglib* in MPI mode. These new flags replace the *oempirun* script.
- This will be the last release to support SuSe 10.

13.2 OMEGA 2.4.6 *(February 2012)*

13.2.1 New features

- Now uses an extension of the MMFF94 force field for tricoordinate boron compounds. Most compounds containing B-X bonds where X=C,N,O,S, and H are covered with the following exceptions: X=N(imine),N(sulfonamide), N(pyridinium) and N(quaternary). Also not supported are compounds in which boron is bonded to X=F,Cl,Br,I,B and Si, or makes a bond angle BYX. Compounds in which boron is a part of four-membered rings of B1CCC1 type are also not available in the current parameterization because their existence is questionable: Ab initio calculations at the MP2/6-31G** level failed to identify stable structures for them (highly polar structures in which boron is four-coordinated are formed).

13.2.2 Bug fixes

- Fixed a bug where OMEGA wasn't checking if 3D construction of fragment generation was successful.
- Fixed a bug where output from flipper could still fail in OMEGA for missing stereo chemistry. Now the rules for required stereo chemistry are consistent.
- Fixed a bug where setting stereo chemistry in too small of a ring could cause a crash. Now only rings greater than 7 members and only one double bond will be set with flipper.
- Fixed a bug where OMEGA wasn't checking if user generated fragment libraries were in OEB format. Fragment libraries must be in OEB format.
- Fixed bugs related to nonsense values for command line parameters. User-supplied parameters must be a legal value. The '–help <parameter>' command will print out the legal values.
- Fixed a bug where SD tagged data wasn't passed along to output molecules.

13.3 OMEGA 2.4.3 *(August 2010)*

13.3.1 Bug fixes

- A bug was fixed where slave nodes required a license. This has been fixed and now only the master requires a license.

- The -warts flag did not work when writing to formats other than .oeb or .oeb.gz. This has been corrected in version 2.4.3.

13.4 OMEGA 2.4.1 *(June 2010)*

13.4.1 New features

- FILTER is now bundled with the OMEGA distribution and will run with a valid OMEGA license.
- OMEGA and *makefraglib* may now be run with MPI using the *oempirun* script. (Not available on all platforms)
- The default for OMEGA has been changed to require all stereocenters to be specified. Molecules with unspecified stereo will fail, except for unspecified invertible nitrogens which are allowed and will not cause the molecule to fail. The previous behavior was for OMEGA to choose a random stereoisomer. This default can be changed back to the previous behavior by setting *-strictstereo* to `false`.
- By default, FLIPPER will no longer flip invertible nitrogens. The default path is for FLIPPER to leave them unspecified and allow OMEGA to enumerate them. Flipping invertible nitrogens in both FLIPPER and then OMEGA will lead to duplication. These options can be changed with the *-enumNitrogen* flag available in both applications.
- The default for OMEGA has been changed to no longer use 'close enough' atom typing. If a proper MMFF atom type cannot be found for an atom then the molecule will fail. Previous versions would use a different atom type of the same element if it was available. The previous behavior for atom type substitution can be enabled by setting *-strictatomtyping* to `false`.
- OMEGA now has the option to require on-the-fly generation of fragments to be the same as the *makefraglib* utility by setting *-strictfrags* to `true`. This fragment generation is more rigorous but also more time consuming.
- The convenience flag *-strict* has been added to turn on or off all of the strict options. These include *-strictstereo*, *-strictatomtyping* and *-strictfrags*.
- On-screen progress has been enabled for OMEGA and *makefraglib* using the *-progress* flag. The options are *none*, *dots*, *log*, and *percent*.
- The *-maxconfgen* and *-maxpoolsize* flags have been removed. These options are set internally and adjusted automatically subject to the user defined variable of *-maxconfs*. Additionally, hard limits have been removed from these variables and they are now only limited by available memory.
- Improper formatting of the *-erange*, *-maxConfRange*, and *-rmsrange* parameters will cause OMEGA to exit immediately. Previously the application would throw a warning but continue the calculation without using the parameters.
- The flag *-fixsmarts* has been created to allow a smarts pattern to be used to fix a portion of the molecule. This requires 3D coordinates to be available from either a 3D input file with *-fromCT* set to `false` or from a molecule set using *-fixfile*.

13.4.2 Bug fixes

- A bug was fixed in *makefraglib* where the setting *-fromCT false* could give nonsense fragments.
- Crashes have been fixed that could occur when the *-fixfile* could not find a valid match or did not have enough atoms to match against in a ring system.
- Program no longer crashes if there is not enough memory for duplicate removal. If there is not enough memory available for allocation then the application will exit immediately.

13.5 OMEGA 2.4.0 *(November 2009)* (Toolkit only)

This release was for the toolkit only. For more details please refer to the OmegaTK documentation.

13.6 OMEGA 2.3.3 *(March 2009)* (Toolkit only)

This release was for the toolkit only. For more details please refer to the OmegaTK documentation.

13.7 OMEGA 2.3.2

13.7.1 Bug fixes

- Fixed a bug where molecules containing boron or selenium could cause a crash. These molecules now are written to the *'.fail'* file due to missing force field parameters and the program will proceed to the next molecule.

13.8 OMEGA 2.3.1

13.8.1 New features

- The OMEGA application now includes a utility named oeb2sdconf that allows OMEGA generated conformers to be used with the MOE program. Please contact Chemical Computing Group support if you require assistance with this utility.

13.8.2 Bug fixes

- Fixed a bug where the omega2 PVM features would not work on some platforms.
- Fixed a bug where the omega2 script would not function properly with filenames that had spaces in them.

13.9 OMEGA 2.3.0

13.9.1 New features

- The distribution and installation of OMEGA has been modified. The Windows distribution is now a standard .exe installer, and the OS X distribution is a dmg containing a standard pkg installer. The executables are now scripts that chose the correct version of the program at runtime. Please see the Installation and Platform Notes for details.
- The defaults for MaxConfs and RMSThreshold are changed in this version. The default for MaxConfs is reduced from 400 to 200, while the value for RMSThreshold goes from 0.8 down to 0.5. These changes are based on maintaining good reproduction of the crystal structure test set, virtual screening tests with ROCS and FRED, and pose prediction tests with FRED.
- The method `OEConfGen::OEOmega::ClearFixFile` is added to the OMEGA library. This allows an `OEConfGen::OEOmega` instance to be reused for an unrestrained search after it has previously been used for a fixed search.

- Untitled molecules are given unique titles of omega_1, omega_2, and so on. Titles allow molecules to be located in the log files.

13.9.2 Bug fixes

- Fixed a major bug where molecules that have missing torsion rules caused OMEGA to crash. Molecules with missing torsion rules are now sent to the *omega2.fail* file and are not processed. Missing torsion rules are typically caused by highly unusual molecules for which OEChem cannot properly assign valence to all atoms.
- Fixed a bug in duplicate conformer removal that caused some conformers that should have been removed to pass through as unique. This bug also caused occurrences where a reduction in maxconfs led to an increase in the number of conformers returned.
- Fixed a bug that caused SD data on single conformer molecules to be removed.
- Fixed a bug in the *flipper* library that left 3D coordinates intact, which caused stereo chemistry to be ambiguous. The *flipper* library now replaces 3D coordinates with zeros.
- Fixed a bug where the `SyM` tag from previous OMEGA calculations caused problems when those molecules where passed back in. Also, the `O2MolId` tag is now removed from molecules before they are sent to the output.
- When the number of stereo centers exceeds the number of -maxcenters M, *flipper* now generates 2^M random non-repeating flips. Previous versions generated 2^M random flips, leading to molecules with the same stereochemistry being returned multiple times.

13.10 OMEGA 2.2.2

13.10.1 Bug fixes

- Fixed a major bug that would cause highly-symmetric, highly-fluorinated compounds to crash the program or hang a slave in PVM mode.
- Fixed a bug in atom typing when building fragments from scratch that, in some rare cases, could result in differing fragment geometries depending on the order of input of the molecules.
- Added more places that check for time exceeding max time. This is more just to prevent overly large molecules from appearing to hang.
- If OMEGA is called with a fixed predicate, this now implies that FromCT is false. In other words, if you want to use a predicate to fix atoms, you need to use a 3D structure as input.

13.11 OMEGA 2.2.1

13.11.1 Bug fixes

- The new build algorithm introduced in v2.2.0 could, on rare occasions, generate a structure with heavy atom clashes. This release is mainly a bug fix for this problem.

CHAPTER

FOURTEEN

OLD FILTER RELEASE NOTES

14.1 Release Notes

14.1.1 FILTER 2.1.1 *(August 2010)*

New features

- Added several halide that are more strict to the fragment filter.
- Added several rules to remove bad ring systems to the fragment filter.

Note: The proper way to override a `RULE` line is to use a `NEWRULE` line.

Minor bug fixes

- Removing duplicate nitroso and phosphoryl filter file RULES from the default filters.

14.1.2 FILTER 2.1.0 *(May 2010)*

New Features

- FILTER now will run with a valid OMEGA license

14.1.3 FILTER 2.0.3 *(November 2009)* (Toolkit only)

This is a minor bug fix release that includes some missing filter definition files that were missing from FILTER 2.0.2.

Bug fixes

- FILTER distribution now includes filter_alldrug.txt filter definition file.
- FILTER distribution now includes filter_blockbuster.txt filter definition file.
- FILTER distribution now includes filter_2.5_blockbuster.txt filter definition file.

14.1.4 FILTER 2.0.2 *(March 2009)*

This is a minor bug fix release that updates FILTER to OpenEye's standardized distribution system.

Bug fixes

- FILTER installations now support having multiple versions and platform architectures in the same directory structure.
- FILTER installations on Microsoft Windows platforms now have documentation available in the start menu.
- FILTER installations on Microsoft Windows platforms now have OpenEye Command Prompts available from the start menu. These prompts set the appropriate environment variables for running OpenEye software.

14.1.5 FILTER 2.0.1 *(January 2007)*

This release is a broad-sweeping bug fix release with a dramatically improved introduction in the documentation. It includes many simple and complex bug fixes including; conceptual clarifications of several algorithms, fixes and improvements to the data representation, several advances in the pKa model, and significantly more documentation. While this is only a bug-fix release, it represents a significant step-forward in the refinement of the product. We hope you find it suits your needs.

New features

- Added three additional filters that are designed around the best-selling prescription drugs
- Predicted aggregator QSAR model introduced to further enhance aggregator recognition capability
- Implemented an algorithm for "total functional group count" that takes account of all functional groups rather than only those included in the filter patterns
- Added -tableFlag parameter to mark all failure values in the table file with an asterisk
- Extended the unique flag to allow values of "true", "false" or a filename used to pre-load a file of uniques
- Added pKa for phenol groups with significant electron withdrawing groups

Bug fixes

- Fixed two bugs in the algorithm for recognizing hydrogen-bond donors and acceptors
- Abbott Bioavailability Score corrected to be reported as a probability
- Aggregator function augmented to include more than 250 additional known aggregators
- XLogP and LogS corrected to always examine the hypervalent form of molecules (as are used in the training set)
- Improved data reporting and filter control for the pharmacokinetic properties
- Fixed category reporting for solubility calculation
- Changed the default TPSA calculation to ignore rather than include P and S surface area
- Renamed and clarified aliphatic_group filter to "maximum connected non-ring atoms"
- Fixed typos in names beta_carbonyl_quat_nitrogen and t_butoxycarbonyl_tBOC
- Updated chloramidine filter name to the more common imidoyl_chloride
- Fixed error in beta_azo_carbonyl pattern
- Made patterns for cytochalasin, monensin, squalestatin and saponin more specific
- Clarified the exclusion patterns for acyclic_NCN and SCN2

- Dramatically improved the specificity and range of the michael_acceptor pattern
- Fixed error in alkyl_halide pattern that limited it to bromine and iodine
- Fixed multiple patterns that matched their function group multiple times because of symmetry
- Limited enamine definition to exclude exceedingly delocalized instances
- Fix t_butyl_ether pattern to avoid matching t_butyl_alcohols
- Removed dependence of rigid bond count upon implicit or explicit hydrogens
- Cleaned-up table data for the select parameter
- Corrected problem that MIN_HALIDE_FRACTION and ALLOWED_ELEMENTS could not properly be removed from the filter file
- Fixed bug so NEWRULE lines with tailing white space are not ignored
- Fixed crash bug that occurred when a filter file was specified that was not a valid file
- Improved pKa value for barbiturate-like compounds
- Clarified alpha-acyl carbon deprotonation
- Removed 1,2,3 triazole pKa rule
- Clarified tertiary amine rule
- Improved efficiency for applying pKa rules to multi-conformer molecules

14.1.6 FILTER 2.0.0 *(October 2005)*

This is the first official "2.0" version of FILTER. It includes many bug-fixes and feature expansions based on the long beta period. Some of these include higher level filtering flags such as Lipinski and other similar measures. This release also includes a pKa model that was missed in the beta release as well as non-pH normalization capabilities. This version is makes exporting filtering data to other programs facile with both table and sdtag data export.

New features

- Neutral pKa model optionally applied to all molecules
- Support for molecular graph normalization
- Optional salt file can be used to remove salts from database
- Calculated properties can be attached to molecules as SD tag data
- All filtering data can be saved in tab-separated value format
- Filter on fractional halide weight
- Removal of known aggregator molecules
- Veber bioavailability filtering
- Lipinski violation filtering
- Abbott/Martin bioavailability filtering
- Pharmacopia bioavailability filtering
- Solubility class filtering
- Chiral center count filtering

- Continuous Medicinal Value

CHAPTER

FIFTEEN

CITATION

Note: To cite OMEGA please use the following:

OMEGA 2.5.1.4: OpenEye Scientific Software, Santa Fe, NM. http://www.eyesopen.com. Hawkins, P.C.D.; Skillman, A.G.; Warren, G.L.; Ellingson, B.A.; Stahl, M.T.

Conformer Generation with OMEGA: Algorithm and Validation Using High Quality Structures from the Protein Databank and the Cambridge Structural Database *J. Chem. Inf. Model.* **2010**, 50, 572-584.

BIBLIOGRAPHY

[Bostrom-2001] Jonas Bostrom, **Reproducing the Conformations of Protein-Bound Ligands: A Critical Evaluation of Several Popular Conformational Searching Tools**, *Journal of Computer-Aided Molecular Design (JCAMD)*, Vol. 15, pp. 1137, **2001**

[Bostrom-2003] J. Bostrom, J.R. Greenwood, and J. Gottfries, **Assessing the Performance of OMEGA with Respect to Retrieving Bioactive Conformations**, *Journal of Molecular Graphics and Modeling*, Vol. 21, pp. 449–462, **2003**

[Gasteiger-1990] J. Gasteiger, C. Rudolph and J. Sadowski, **Automatic Generation of 3D Atomic Coordinates for Organic Molecules**, *Tetrahedron Comp. Method*, Vol 3., pp. 537-547, **1990**

[Gasteiger-1993] J. Sadowski and J. Gasteiger, **From Atoms and Bonds to Three-dimensional Atomic Coordinates: Automatic Model Builders**, *Chemical Reviews*, Vol. 93, pp. 2567-2581, **1993**

[Grant-2001] J. Andrew Grant, Anthony Nicholls, A. Geoffrey Skillman, and Matthew T. Stahl, **Dude, where are my conformers?**, *222nd National ACS Meeting*, **2001**

[Grant-2007] J.A. Grant, B.T. Pickup, M.J. Sykes, C.A. Kitchen and A. Nicholls, **A Simple Formula for Dielectric Polarization Energies: The Sheffield Solvation Model**, *Chem. Phys. Letters*, Vol. 441, pp. 163-166, **2007**

[Halgren-1996-1] T.A. Halgren, **Merck Molecular Force Field: I. Basis, Form, Scope, Parameterization and Performance of MMFF94**, *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 490-519, **1996**

[Halgren-1996-2] T.A. Halgren, **Merck Molecular Force Field: II. MMFF94 van der Waals and Electrostatic Parameters for Intermolecular Interactions**, *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 520-552, **1996**

[Halgren-1996-3] T.A. Halgren, **Merck Molecular Force Field: III. Molecular Geometries and Vibrational Frequencies**, *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 553-586, **1996**

[Halgren-1996-4] T.A. Halgren, **Merck Molecular Force Field: IV. Conformational Energies amd Geometries for MMFF94**, *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 587-615, **1996**

[Halgren-1996-5] T.A. Halgren, **Merck Molecular Force Field: V. Extension of MMFF94 using Experimental Data, Additional Computational Data and Empirical Rules**, *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 616-641, **1996**

[Halgren-1999-1] T.A. Halgren, **MMFF VI. MMFF94s Option for Energy Minimization Studies**, *Journal of Computational Chemistry*, Vol. 20, No. 5, pp. 720-729, **1999**

[Halgren-1999-2] T.A. Halgren, **MMFF VII. Characterization of MMFF94, MMFF94s and Other Widely Available Force Fields for Conformational Energies and for Intermolecular Interaction Energies and Geometries**, *Journal of Computational Chemistry*, Vol. 20, No. 5, pp. 730-748, **1999**

[Hawkins-2010] P.C.D. Hawkins, A.G. Skillman, G.L. Warren, B.A. Ellingson and M.T. Stahl, **Conformer Generation with OMEGA: Algorithm and Validation Using High Quality Structures from the Protein Databank and Cambridge Structural Database**, *J. Chem. Inf. Model.*, Vol. 50, No. 4, pp 572-584, **2010**

[Sadowski-1994] J. Sadowski, J. Gasteiger and G. Klebe, **Comparison of Automatic Three-dimensional Model Builders using 639 X-Ray Structures**, *Journal of Chemical Information and Computer Science (JCICS)*, Vol. 34, pp. 1000-1008, **1994**

[Spellmeyer-1997] David C. Spellmeyer, A.K. Wong, M.J. Bower and J.M. Blaney, **Conformational Analysis using Distance Geometry Methods**, *Journal of Molecular Graphics and Modeling*, Vol. 15, No. 1, pp. 18-36 **1997**

[Stahl-2000] Matthew T. Stahl, **Rapid 3D Database Searching**, *IIR: Computational Drug Design*, **2000**

[Stahl-2001] Matthew T. Stahl, **You want conformers? I'll give you conformers!**, *2nd Annual OpenEye Customers, Users, and Programmers Meeting*, **2001**

[Stahl-2002] Matthew T. Stahl, **Omega, AESOP, and other cautionary tales of naming programs.**, *3rd Annual OpenEye Customers, Users, and Programmers Meeting*, **2002**

[Clark-1999] Clark, D.E., **Rapid Calculation of Polar Molecular Surface Area and its Application to the Prediction of Transport Phenomena. 1. Prediction of Intestinal Absorption**, *Journal of Pharmaceutical Sciences*, Vol. 88, pp. 807-814, **1999**

[Egan-2000] Egan, W.J., Merz, K.M. and Baldwin, J.J., **Prediction of Drug Absorption Using Multivariate Statistics**, *Journal of Medicinal Chemistry*, Vol. 43, pp. 3867-3877, **2000**

[Ertl-2000] Ertl, P., Rohde, B., and Selzer, P., **Fast Calculation of Molecular Polar Surface Area as a Sum of Fragment-Based Contributions and its Application to the Prediction of Drug Transport Properties**, *Journal of Medicinal Chemistry*, Vol. 43, pp. 3714-3717, **2000**

[Jeffrey-1997] George A. Jeffrey, **An Introduction to Hydrogen Bonding**, *Oxford University Press*, **1997**

[Lipinski-1997] Lipinski, C.A., Lombardo, F., Dominy, B.W. and Feeney, P.J., **Experimental and Computational Approaches to Estimate Solubility and Permeability in Drug Discovery and Development Settings**, *Advanced Drug Delivery Reviews*, Vol. 23, pp. 3-25, **1997**

[Lovering-2009] Frank Lovering, Jack Bikker, and Christine Humblet, **Escape from Flatland: Increasing Saturation as an Approach to Improving Clinical Success**, *Journal of Medicinal Chemistry*, 52 (21):6752-6756

[Martin-2005] Martin, Y.C., **A Bioavailability Score**, *Journal of Medicinal Chemistry*, Vol. 48, pp. 3164-3170, **2005**

[McGovern-2003] McGovern S.L., Helfand, B.T., Feng, B. and Shoichet, B.K., **A Specific Mechanism of Nonspecific Inhibition**, *Journal of Medicinal Chemistry*, Vol. 46, pp. 4265-4272, **2003**

[MillsDean-1996] Mills, J.E.J and Dean, P.M., **Three-Dimensional Hydrogen-Bond Geometry and Probability Information from a Crystal Survey**, *Journal of Computer-Aided Molecular Design*, Vol. 10, pp. 607-622, **1996**

[Oprea-2000] Oprea, T., **Property Distribution of Drug-Related Chemical Databases**, *Journal of Computer-Aided Molecular Design*, Vol. 14, pp. 251-264, **2000**

[Rishton-2003] Rishton, G.M., **Nonleadlikeness and Leadlikeness in Biochemical Screening**, *Drug Discovery Today*, Vol. 8, pp. 86-96, **2003**

[Ritchie-2009] Timothy J. Ritchie and Simon J.F. Macdonald, **The impact of aromatic ring count on compound developability - are too many aromatic rings a liability in drug design?**, *Drug Discovery Today*, 14(21):1011

[Seidler-2003] Seidler J., McGovern, S.L., Doman, T.N. and Shoichet, B.K., **Identification and Prediction of Promiscuous Aggregating Inhibitors Among Known Drugs**, *Journal of Medicinal Chemistry*, Vol. 46, pp. 4477-4486, **2003**

[Veber-2002] Veber, D.F., Johnson, S.R., Cheng, H.Y., Smith, B.R., Ward, K.W. and Kopple, K.D., **Molecular Properties that Influence the Oral Bioavailability of Drug Candidates**, *Journal of Medicinal Chemistry*, Vol. 45, pp. 2615-2623, **2002**

[Wang-1997] Wang, R., Ying, F., and Lai, L., **A New Atom-Additive Method for Calculating Partition Coefficients**, *Journal of Chemical Information and Computer Science*, Vol. 37, pp. 615-621, **1997**

[Wishart-2006] Wishart, D.S., **DrugBank: A Comprehensive Resource For In Silico Drug Discovery and Exploration**, *Nucleic Acids Research*, Vol. 34, pp. D668-672, **2006**

[Yalkowsky-1980] Yalkowsky, S.H. and Valvani, S.C., **Solubility and Partitioning 1: Solubility of Nonelectorlytes in Water**, *Journal of Pharmaceutical Sciences*, Vol. 69, pp. 912-922, **1980**

INDEX

Symbols

A

E

F

M

O

P