mol2chemfig Documentation

Version 1.4

August 14, 2015

Abstract

mol2chemfig is a Python program that generates TEX graphics of chemical structures provided in molfile or SMILES format. Its output is written in the syntax of the chemfig package, which in turn is based on TiKZ. Thus, both these packages are required in order to compile mol2chemfig's output. The program is available a) through a web interface, b) for local installation, and c) as a command line-driven thin client. The thin client is a Lua script that is designed to work with recent TeXLive installations.

Contents

1 A few examples

The following is a SMILES representation of caffeine, contained in the file caffeine.smi:

```
CN1C=NC2=C1C(=O)N(C(=O)N2C)C
```

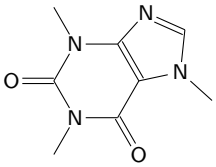
We can turn this into chemfig code with the following command:[1]

```
mol2chemfig -zw caffeine.smi > caffeine-smi.tex
```

This writes the following chemfig code to the target file:

```
\chemfig{-[:138]N-[:84]=^[:156]N-[:228]=^[:300](-[:12]\phantom{N})-[:240](%
=[:300]O)-[:180]N(-[:240])-[:120](=[:180]O)-[:60]N(-)-[:120]}
```

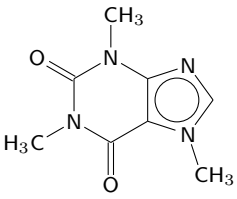
If we load this file with \input{caffeine-smi.tex}, we get the following result[2]



which is almost what we want. We adjust the rotation of the molecule and also tweak its appearance a little:

```
mol2chemfig -zwom -a -30 caffeine.smi > caffeine-smi2.tex
```

With these modifications, the structure looks as follows:



Small molecules like caffeine are fairly easy to create with chemfig alone. Hand-written chemfig code will look cleaner and more concise than the code generated by mol2chemfig. For example, here is the hand-coded chemfig version for caffeine that will produce the exact same graphic as the last mol2chemfig command:

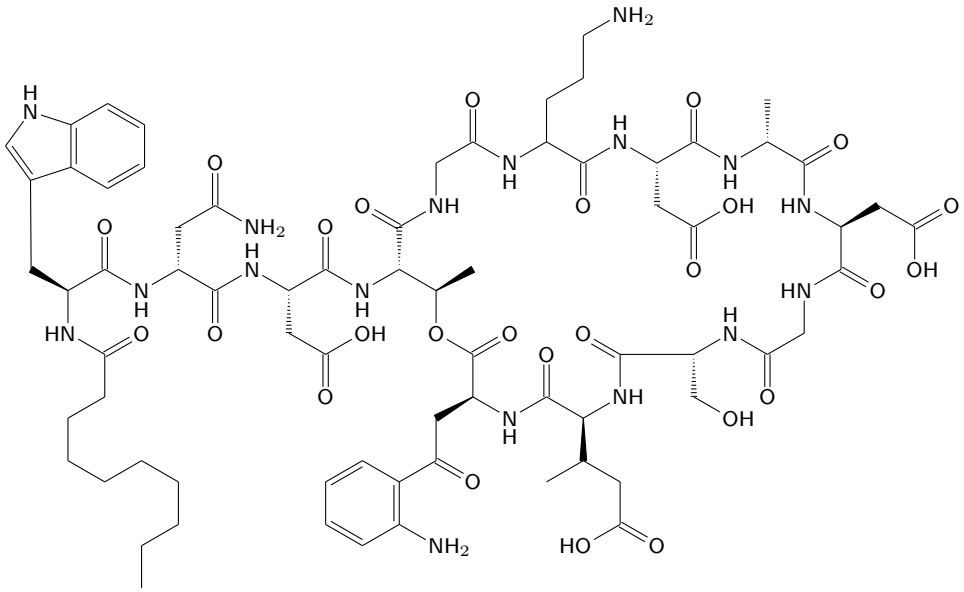
```
\chemfig{H_3C-[:30]N**6(-(=O)-(**5(-N(-CH_3)--N-))--N(-CH_3)-(=O)-)}
```

[1] The examples in this document assume that you have installed the local version of mol2chemfig. If you installed the web client version, you must use mol2chemfig.lua wherever the listed example commands use mol2chemfig.

[2] The appearance of the chemical formulas in this document has been tweaked using some settings that are provided by the chemfig package. The settings used here are listed in Section 6.

In particular, the syntax for specifying rings and ring substituents in `chemfig` is remarkably powerful and elegant. With a little practice, molecules such as this one can be hand-written with little effort, and I sincerely recommend that you learn this skill. However, if you need to depict many small molecules, or fairly large ones, a tool like `mol2chemfig` will come in handy. Take, for example, the lipopeptide antibiotic daptomycin, which we in this case load from a molfile:

```
mol2chemfig -wf daptomycin.mol > daptomycin.tex
```



This molecule might take a little longer to code by hand . . .

2 Installation

There are three ways to run `mol2chemfig`: 1. You can use the web interface. 2. You can install a web client version of `mol2chemfig` that is operated from the command line. 3. You can install the entire program locally.

With options 1 and 2, everything you need can hopefully be installed via TeXLive by the time you read this. In contrast, with option 3, some additional handiwork is required.

Update: `mol2chemfig` has been on CTAN for a good while now, but as far as I can tell adoption into TeXLive has not happened, and apparently isn't going to.

2.1 LaTeX requirements

In order to use the code generated by `mol2chemfig` in your documents, you need

1. `TiKZ`, a large and powerful general graphics package.
2. `chemfig`, which defines the code format used by `mol2chemfig` and uses `TiKZ` to render it to molecule graphics.

3. The `mol2chemfig.sty` package. This is a very small package that defines some auxiliary macros for `mol2chemfig`.

`TiKZ` and `chemfig` are available through TeXLive. Note that the `mol2chemfig` package will load the `chemfig` package for you, which in turn loads `TiKZ`. Therefore, in your documents, it is sufficient to include the clause `\usepackage{mol2chemfig}` in order to load all three of them.

2.2 Installing the Lua web client

The web client should be installed and properly configured when you install `mol2chemfig` through TeXLive; this is the preferred method. You should be able to invoke it from a shell window (or command prompt, in Windows parlance) with the command

```
$ mol2chemfig.lua
```

If you downloaded the `mol2chemfig` bundle from CTAN, no automatic installation is performed. In that case, follow the instructions in the included README file.

2.3 Local installation

For the full local install of `mol2chemfig`, you need a Python base installation, and additionally the `indigo` cheminformatics toolkit. On at least some Linux distributions, `indigo` is available through the package manager. On other systems, manual download (`http://lifescience.opensource.epam.com/indigo/index.html`) and installation may be required.

Once you have the prerequisites installed and working, download the `mol2chemfig` Python code from CTAN or from `http://chimpsky.uwaterloo.ca/mol2chemfig`, unzip in a convenient location and follow the included instructions.

2.4 ConTeXt compatibility

Currently not available. Volunteers for porting welcome.

3 Getting help

When you type `mol2chemfig -h` or just `mol2chemfig`, you will see a brief description of the program, as well as a list of all available options:

```
mol2chemfig v. 1.4,  by Eric Brefo-Mensah and Michael Palmer
mol2chemfig generates chemfig code from molfiles. Usage example:

mol2chemfig --angle=45 --aromatic-circles somefile.mol

Options:
-h, --help                      Print help message and exit (Default:
                                False)

-b, --version                   Print program version and exit (Default:
```

```
                                  False)

-i, --input                       How to interpret the argument. With 'file',
                                  mol2chemfig expects a filename. With
                                  'direct', the argument is intrepreted
                                  directly; don't forget to put quotes around
                                  it. With 'pubchem', the argument is treated
                                  as an identifier for the PubChem database.
                                  (Default: file)

-z, --terse                       Remove all whitespace and comments from the
                                  output. If you can still read it
                                  afterwards, Bill Gates wants your resume
                                  (Default: False)

-r, --strict                      Abide by Indigo's chemical structure
                                  validation. If true, mol2chemfig will fail
                                  if Indigo reports that something is wrong
                                  with the molecule, like a carbon with five
                                  bonds. If false, mol2chemfig will ignore
                                  such errors (Default: True)

-d, --indent                      Number of spaces to use for indenting
                                  molecule branches in generated code.
                                  Without effect when 'terse' option is
                                  passed. Affects only the generated tex
                                  code, not the rendered molecule (Default:
                                  4)

-u, --recalculate-coordinates     Discard existing coordinate and calculate
                                  new ones from covalent structure. For
                                  smiles input, this is performed implicitly
                                  (Default: False)

-a, --angle                       Rotate molecule counterclockwise by this
                                  angle (Default: 0.0)

-v, --relative-angles             Use relative bond angles (Default: False)

-p, --flip                        Flip the structure horizontally (Default:
                                  False)

-q, --flop                        Flip the structure vertically (Default:
                                  False)

-c, --show-carbons                Show element symbol for carbon atoms
                                  (Default: False)

-m, --show-methyls                Show element symbols for methyl groups
                                  (implied if show-carbons is True) (Default:
                                  False)

-y, --hydrogens                   How to deal with explicit hydrogen atoms.
                                  One of 'keep', 'add' or 'delete'. Note that
```

```
                              'add' will also trigger calculation of new
                              coordinates for the entire molecule. Option
                              'keep' does nothing (Default: keep)

-o, --aromatic-circles        Draw circles instead of double bonds inside
                              aromatic rings (Default: False)

-f, --fancy-bonds             Draw fancier double and triple bonds
                              (Default: False)

-g, --markers                 Give each atom and each bond a unique
                              marker that can be used for attaching
                              electron movement arrows. With value 'a',
                              atom 2 will be labeled @{a2}, and its bond
                              to atom 5 @{a2-5}. (Default: None)

-n, --atom-numbers            Show the molfile number of each atom next
                              to it. When this option is set, charges and
                              implicit hydrogens will not be shown
                              (Default: False)

-s, --bond-scale              How to scale the lengths of bonds (one of
                              'keep', 'scale', or 'normalize') (Default:
                              normalize)

-t, --bond-stretch            Used as scaling factor (with --bond-
                              scale=scale) or average (with --bond-
                              scale=normalize) for bond lengths (Default:
                              1.0)

-w, --wrap-chemfig            Wrap generated code into \chemfig{...}
                              command (Default: False)

-l, --submol-name             If a name is given, wrap generated code
                              into chemfig \definesubmol{name}{...}
                              command (Default: None)

-e, --entry-atom              Number of first atom to be rendered.
                              Relevant only if generated code is to be
                              used as sub-molecule (Default: None)

-x, --exit-atom               Number of last atom to be rendered.
                              Relevant only if generated code is to be
                              used as sub-molecule (Default: None)

-k, --cross-bond              Specify bonds that should be drawn on top
                              of others they cross over. Give the start
                              and the end atoms. Example for one bond:
                              --cross-bond=5-6 Example for two bonds:
                              --crossbond=4-8,12-13 (Default: None)
```

4 Tutorial

In the following, we will work through a series of examples to illustrate the use of some of `mol2chemfig`'s options. With all but the most basic examples, it will be assumed that you are familiar with `chemfig` and its syntax. If you aren't, you can still use `mol2chemfig`, but you will not be able to manually enhance the `chemfig` output that it generates—your loss.

4.1 Input modes

By default, if you type `mol2chemfig myinput`, `mol2chemfig` expects `myinput` to be the name of a file that contains a molecule's description in either `molfile` or `SMILES` format. These are widely used file formats that can be exported from any chemical drawing program that I am aware of. If you want `myinput` to be treated verbatim, rather than as a file name, you say:

```
mol2chemfig --input=direct 'C1=CC=C(C=C1)O'
```

or, shorter:

```
mol2chemfig -i direct 'C1=CC=C(C=C1)O'
```

When passing a smiles string as in this example, don't forget to put quotes around it; without them, the shell will try to expand it and likely cause an error.

There is also the input format 'pubchem', which makes `mol2chemfig` expect a numerical identifier for the pubchem database:

```
mol2chemfig -i pubchem 996
```

This retrieves the `SDF` file for compound no. 996 (which happens to be phenol) from the PubChem database and uses it as input. The `SDF` format is a superset of the `molfile` format and is understood by the program as well. Obviously, you have to be online for this input mode to work.

4.2 Output destination and formats

By default, `mol2chemfig` simply writes to the terminal (stdout). Use output redirection to send the output to a file instead:

```
mol2chemfig -i direct 'C1=CC=C(C=C1)O' > phenol-smi.tex
```

This writes the following `chemfig` code to the file:

```
            HO% 7
      -[,,2]% 4
    =^[:300]% 3
           -% 2
     =^[:60]% 1
     -[:120]% 6
    =^[:180]% 5
               (
         -[:240]% -> 4
               )
```

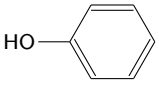
If we input this code directly, we will not produce a graphic; instead, we will just see the code itself, only without the line breaks. To produce a graphic, the code must be enclosed in a `\chemfig{...}` macro. This could be done manually by copying and pasting. It is usually more convenient to use the `-w` or `--wrap-chemfig` option:

```
mol2chemfig -w -i direct 'C1=CC=C(C=C1)O' > phenol-smi-wrapped.tex
```

which will produce

```
\chemfig{
            HO% 7
      -[,,2]% 4
    =^[:300]% 3
           -% 2
     =^[:60]% 1
     -[:120]% 6
    =^[:180]% 5
              (
         -[:240]% -> 4
              )
}
```

This file can then be used with `\input{phenol-smi-wrapped}` directly:



Note that the following will *not* work:

```
\chemfig{\input{phenol-smi.tex}}
```

This is because the `\chemfig{...}` command puts TEX into an altogether different frame of mind, in which many other commands, including `\input`, no longer work as usual.

In addition to the option `--wrap-chemfig` or `-w`, there is also the option `--submol-name` or `-l`, which will wrap the generated code into a `\definesubmol` macro:

```
mol2chemfig -l phenol -i direct 'C1=CC=C(C=C1)O' > phenol-as-submol.tex
```

This produces

```
\definesubmol{phenol}{
            HO% 7
      -[,,2]% 4
    =^[:300]% 3
           -% 2
     =^[:60]% 1
     -[:120]% 6
    =^[:180]% 5
              (
         -[:240]% -> 4
              )
}
```

The `\definesubmol` macro is implemented by `chemfig` and defines a named shortcut for a molecule or fragment. This is useful if you want to integrate the generated code into larger, manually assembled structures or drawings. We will revisit this topic below.

In the file listings above, each atom appears on a separate line and is annotated by a comment with its number. The `% ->` in the last line indicates that this bond does not create a new atom but instead connects back to atom number 4 to close a ring.

The formatting and annotation in the code example above is useful if you want to manually edit the generated code. If this is not required, you can create more compact output with option `--terse` or `-z`:

```
mol2chemfig -zw -i direct 'C1=CC=C(C=C1)O' > phenol-smi-terse.tex
```

The generated code is equivalent, but with comments and whitespace stripped out:

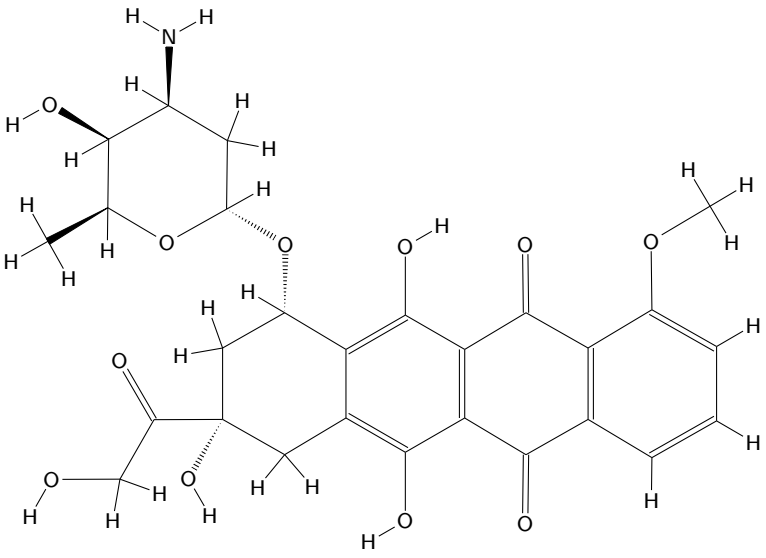
```
\chemfig{HO-[,,2]=^[:300]-=^[:60]-[:120]=^[:180](-[:240])}
```

4.3 Adding and deleting hydrogens

The next example renders an SDF file for doxorubicin, downloaded from PubChem[3]

```
mol2chemfig -w doxorubicin.sdf > doxo-raw.tex
```

This gives us (yikes)

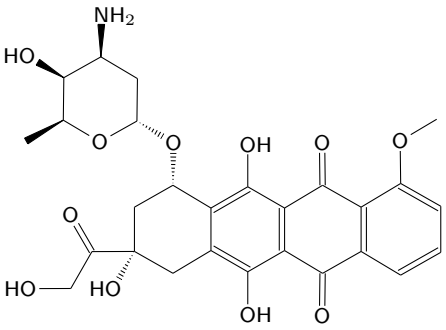


Let's remove the explicit hydrogens with option `--hydrogens=delete` or `-y delete`:

```
mol2chemfig -w -y delete doxorubicin.sdf > doxo-stripped.tex
```

This gives

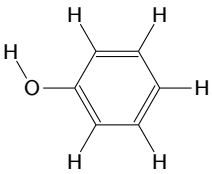
[3] We could also just have used the `-i pubchem` option, but since we are going to reuse the file, a local copy is handy.



It is also possible to *add* hydrogens to a structure that does not supply them; for example:

```
mol2chemfig -y add -w -i direct 'C1=CC=C(C=C1)O' > phenol-with-hydrogens.tex
```

produces



By default, `mol2chemfig` neither removes nor adds hydrogens. Note also that adding hydrogens will trigger coordinate recalculation (see section 4.6).

4.4 Bond lengths

You will have noticed that, in the above doxorubicin example, the removal of the hydrogens was accompanied by a reduction in length of the remaining bonds. This happened because, by default, `mol2chemfig` scales all bond lengths such that the *most frequently occurring* bond length is set equal to 1. In the unstripped version, the most frequent bonds were those to the explicit hydrogens. These were very short, which caused all other bonds to be unduly extended. Stripping the hydrogens restored a more proportioned appearance.

There are two mechanisms with which we can explicitly change the bond lengths:

1. Within `mol2chemfig`, you can use the `--bond-scale` or `-s` option, usually in combination with the `--bond-stretch` or `-t` option.
2. Within `chemfig`, you can set the unit bond length—that is, the length of a bond whose length is equals 1 inside the `\chemfig` macro—with `chemfig`'s `\setatomsep` macro.

The `\setatomsep` approach is straightforward; in this document, `\setatomsep{16pt}` has been used throughout. The two `mol2chemfig` options need a bit more explaining. The `--bond-scale` or `-s` option defines `mol2chemfig`'s overall behavior:

- With a setting of `--bond-scale=keep`, `mol2chemfig` will leave the bond lengths entirely alone; the `--bond-stretch` option will have no effect in this case.
- With `--bond-scale=normalize`, which is the default, all bonds will be scaled such that the most frequently occurring bond length is set to the value of the `--bond-stretch` option, which defaults to 1.

- With `--bond-scale=scale`, the value of `--bond-stretch` will be used as a multiplier to the native length of each bond, as contained in the input file; no normalization will occur in this case.

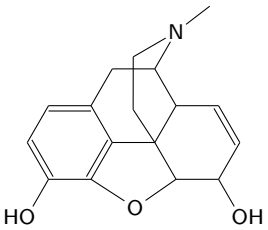
Thus, the meaning of the `--bond-stretch` option depends on the setting of the `--bond-scale` option.

4.5 Tweaking the appearance of bonds

The plain command

```
mol2chemfig -w morphine.mol > morphine.tex
```

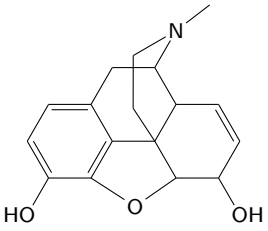
produces



which can be improved. First, we may want to give the double (and, if present, triple) bonds a more well-proportioned look. To do so, use option `--fancy-bonds` or `-f`:

```
mol2chemfig -f -w morphine.mol > morphine-f.tex
```

which gives

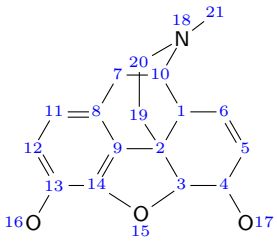


Note that the look of the double bonds involves some `TiKZ` trickery; in the generated code, the double bonds are no longer represented by = symbols but instead by something like `-[:120,,,,drh]`. Also note that, for this to work, you need the current version of the `mol2chemfig` LaTeX package.

Next, the molecule contains two bonds that cross each other, and we would like to draw the vertical one in the foreground. To specify this bond, we need the numbers of the adjoining atoms. We first can let `mol2chemfig` print the atom numbers:

```
mol2chemfig -n -w morphine.mol > morphine-n.tex
```

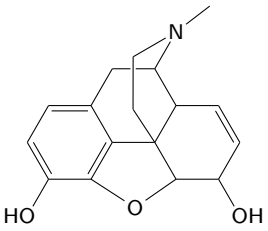
which gives



The bond that we are looking at connects atoms 19 and 20. We now can use the `-k` or `--crossbond` option to put it in the foreground:

```
mol2chemfig -k 19-20 -wf morphine.mol > morphine-k.tex
```

which gives us



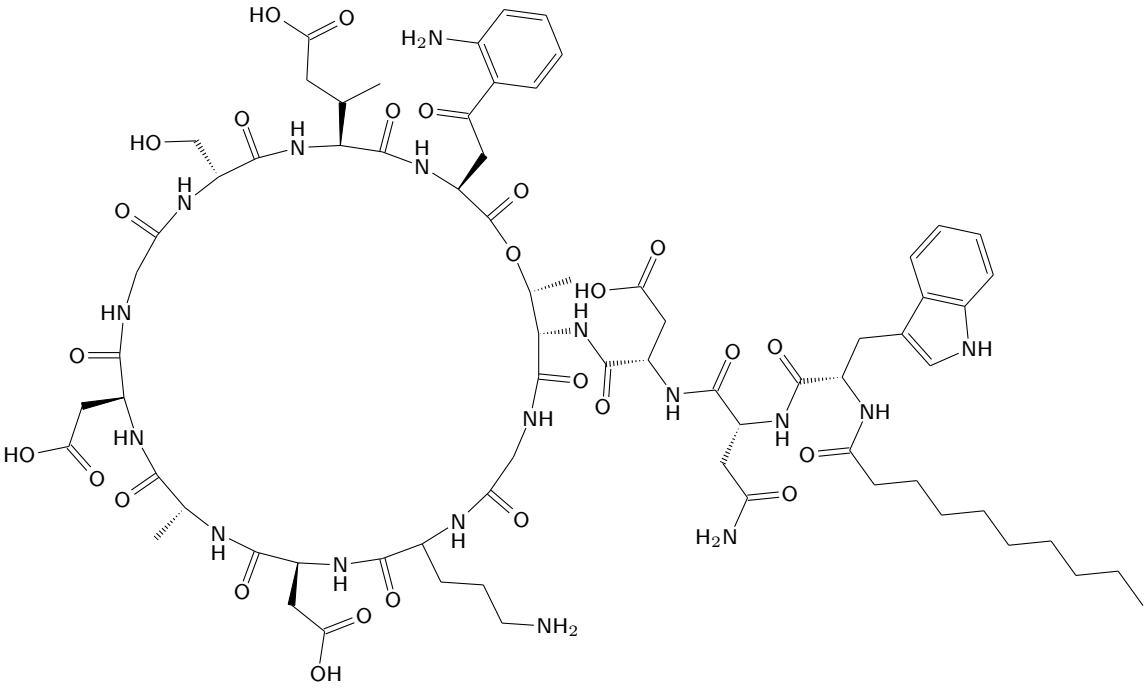
To draw crossing bonds, atom numbers and proportioned double or triple bonds, `mol2chemfig` relies on several custom macros and `TiKZ` styles defined in the `mol2chemfig.sty` package. Accordingly, the generated code becomes a bit more verbose and less `chemfig`-like.

4.6 Recalculating coordinates

Molecules specified in `SMILES` format don't have any coordinates attached to them, so `mol2chemfig` needs to calculate them; this is performed automatically. Input in `molfile` format comes with coordinates attached, and `mol2chemfig` uses these by default. However, we can explicitly request `mol2chemfig` to discard these coordinates and calculate new ones with the `--recalculate-coordinates` or `-u` option. When applied to the daptomycin example from section 1,

```
mol2chemfig -u -wf daptomycin.mol > daptomycin-u.tex
```

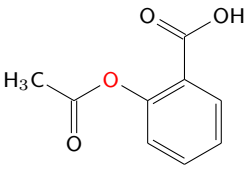
this gives us



4.7 Working with sub-molecules

The `chemfig` package supplies a nifty mechanism to assemble larger molecules from predefined fragments, or sub-molecules. The following (hand-coded) example builds aspirin from two sub-molecules:

```
\definesubmol{acetyl}{(=[::60]O)-[::-60]H_3C}
\definesubmol{benzoate}{*6(-=-=-(-(=[::60]O)-[::-60]OH)=)}%

\chemfig{
    {\color{red}O}              % the oxygen in the middle
         (-[:210]!{acetyl})     % treat one submol as a branch,
    -[:-30]!{benzoate}          % the other one as the main chain
}
```



As a more advanced example, let us piece together the structure of Sodium Green, a fluorescent sodium indicator dye (the figure below was ripped from an information sheet by the supplier, Molecular Probes).

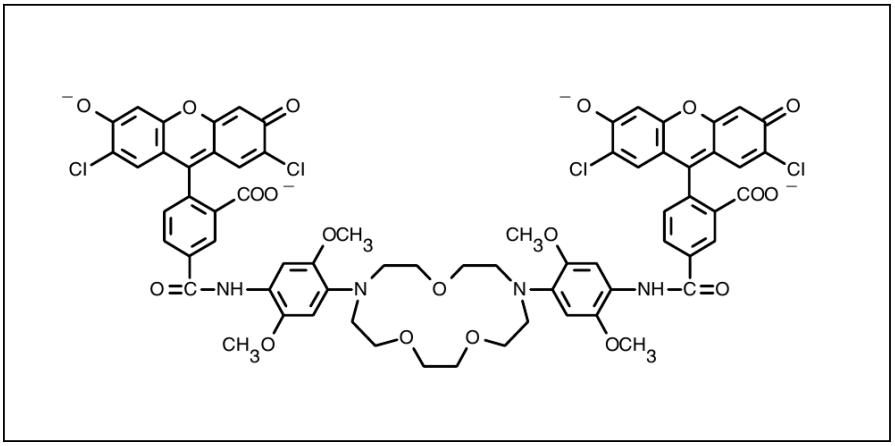


Figure 1. *Chemical structure of Sodium Green indicator.*

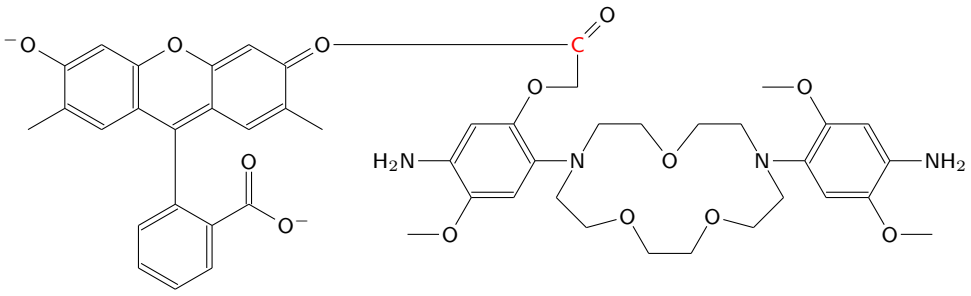
The molecule contains two moieties of dichlorofluorescein, attached to a crown ether via a linker. We start with dichlorofluorescein as one submol and the crown ether-cum-linkers as the other.

```
mol2chemfig -l dcf1 dichlorofluorescein.mol > dcf-submol1.tex
mol2chemfig -l ce1 crown-ether.mol > ce-submol1.tex
```

Now, we put the two pieces together, starting with the bridging carbonyl group:

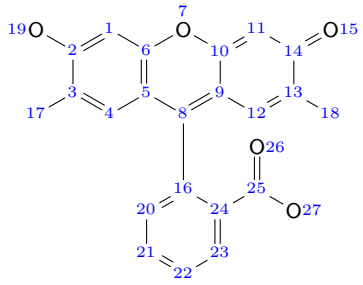
```
\input{dcf-submol1}     % load generated submolecules ...
\input{ce-submol1}

\chemfig{               % and stitch them together
          {\color{red}C}
           (=[:45]O)
           (-[:180,6]!{dcf1})
   -[:-90]!{ce1}
}
```

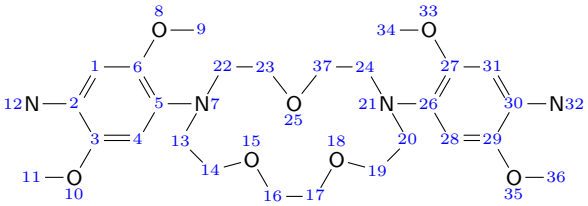
which gives us the following fabulous result:



What went wrong here? Submol expansion is essentially string substitution. When the submol is filled in, the connection to the preceding part of the molecule is made by whatever atom happens to have been rendered first, and the last rendered atom connects to the subsequent parts of the structure. So, we need to tell `mol2chemfig` the order in which to render the atoms. To identify the atoms that we want to join, we display all atom numbers in the structure.

```
mol2chemfig -n -l dcf2 dichlorofluorescein.mol > dcf-submol2.tex
mol2chemfig -n -l ce2 crown-ether.mol > ce-submol2.tex
```





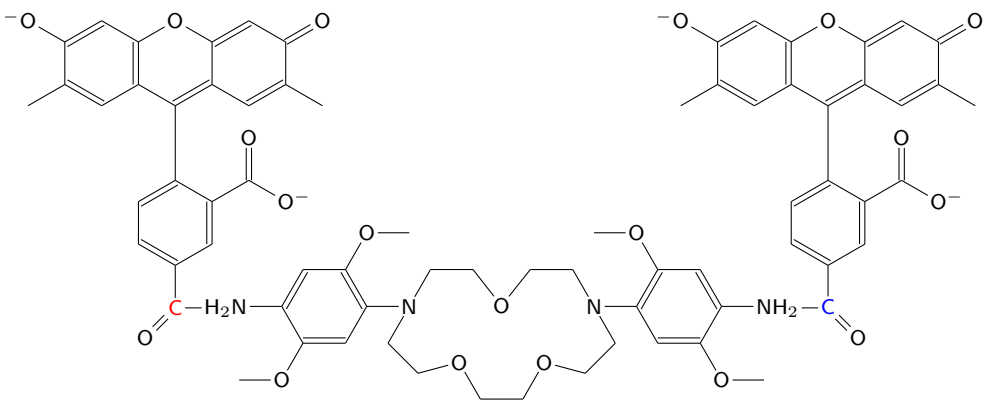
For dichlorofluorescein, we need to connect to atom 22. For the crown ether, we want to enter at atom 12 coming from the left, and leave at atom 32 on the right. To render the sub-molecules accordingly, we use the `--entry-atom` or `-e` as well as the `--exit-atom` or `-x` options:

```
mol2chemfig -e 22 -l dcf3 dichlorofluorescein.mol > dcf-submol3.tex
mol2chemfig -e 12 -x 32 -l ce3 crown-ether.mol > ce-submol3.tex
```

We put the whole molecule together:

```
\input{dcf-submol3}
\input{ce-submol3}

\chemfig{
          {\color{red}C}
           (=[:225]O)
           (-[:90]!{dcf3})
        -!{ce3}
        -{\color{blue}C}
           (=[:-45]O)
        -[:90]!{dcf3}
}
```

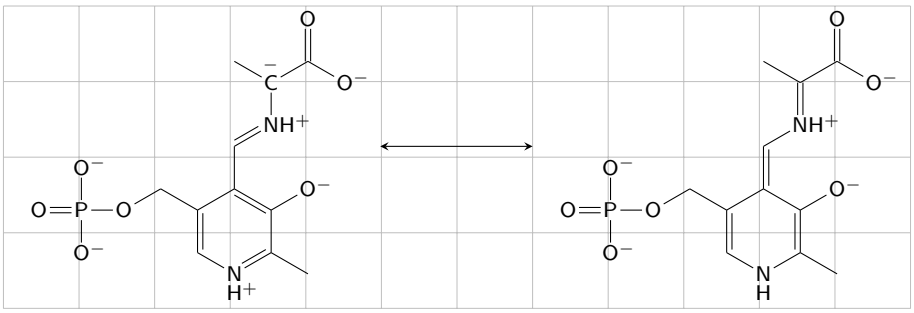
This produces



which is what we want ... well, almost; the entry and exit amine groups of the crown ether submol now each have one surplus hydrogen. This problem cannot be fixed using `mol2chemfig` but requires editing of the generated code by hand.

4.8 Using generated code in composite figures

The `mol2chemfig` package loads `chemfig`, which in turn loads the general purpose graphics package `TikZ`. Through the latter package, we have access to the `tikzpicture` environment. Here is a `tikzpicture` that depicts two resonance structures of alanine bound to pyridoxal phosphate:



The code for this graphics is

```
\input{plp}
\input{plp2}

\begin{center}
\begin{tikzpicture}%
[>=stealth, help lines/.style={very thin,draw=black!25}, x=1cm, y=1cm]

% draw grid of help lines
\draw[help lines] (0,0) grid (12,4);

% place  both structures
\node[anchor=south west, inner sep=10pt] at (0,0) (plp)  {\chemfig{!{plp}}};
\node[anchor=south west, inner sep=10pt] at (7,0) (plp2) {\chemfig{!{plp2}}};

\draw[<->] (plp) -- (plp2);

\end{tikzpicture}
\end{center}
```

The two structures were placed on the canvas using `TiKZ` nodes. The `TiKZ` nodes cannot contain `\input` macros. Therefore, as a workaround, we render the structures as submol definitions with

```
mol2chemfig -f --submol-name plp  plp.mol  > plp.tex
mol2chemfig -f --submol-name plp2 plp2.mol > plp2.tex
```

and then reference those definitions from within the `\chemfig` macros inside the nodes.

Reactions such as the mesomeric transitions within pyridoxal phosphate should also show some electron pushing. The `chemfig` package offers a mechanism for naming atoms and bonds in molecules; these

names can then be used to attach push arrows. We can ask `mol2chemfig` to generate atom and bond names for us, using the `--markers` or `-g` option:

```
mol2chemfig -f -l mp -g mp -a 270 mp.mol > mp.tex
```

This option adds a unique identifier to each atom and to each bond. The option value (`mp` in our example) is used as a prefix; this allows to unambiguously reference atoms in multiple molecules in the same drawing. In our example, atom 3 will be given the marker `@{mp3}` for atom 3. The bond between atoms 3 and 4 will be labeled with `@{mp3-4}`; in bond markers, the smaller atom number always comes first. The generated code now starts to get a wee bit tough on the eyeballs:

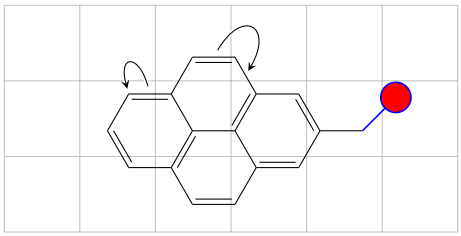
```
\definesubmol{mp}{
                              @{mp17}%
            -[@{mp14-17}:180]@{mp14}%
    -[@{mp14-15}:120,,,,dlh]@{mp15}%
            -[@{mp11-15}:180]@{mp11}%
```

...

In order to attach drawing elements to the nodes defined in the `mol2chemfig`-rendered structures, we need to use a separate `tikzpicture` environment, with the optional arguments `[remember picture, overlay]`. Inside this environment, we can use arbitrary `TiKZ` commands to decorate our rendered structures. Here is an example:

```
\input{mp}

\begin{center}
\begin{tikzpicture}% first picture contains mol2chemfig structure
[>=stealth, help lines/.style={very thin,draw=black!25}, x=1cm, y=1cm]

\draw[help lines] (0,0) grid (6,3);
\node[anchor=south west, inner sep=10pt] at (1,0) {\chemfig{!{mp}}};
\end{tikzpicture}

\begin{tikzpicture}[remember picture,overlay] % overlaid picture
% places draws relative to nodes defined in the mol2chemfig structure

% The mcfpusharrow tikz style is defined in the mol2chemfig package.
\draw[mcfpusharrow](mp12-13) to [out=60,in=60,looseness=4] (mp11-12);
\draw[mcfpusharrow](mp2-3) to [out=105,in=105,looseness=5] (mp3);

% attach some arbitrary shapes
\draw[semithick,blue,fill=red](mp17) -- ++(0.3,0.3) arc (-135:225:0.2);
\end{tikzpicture}
\end{center}
```

which gives



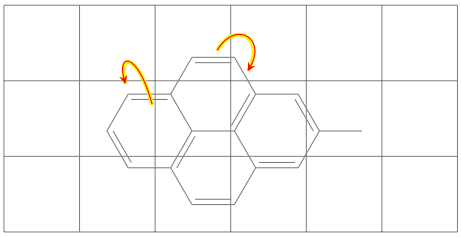
One important thing to note is that, when the overlay mechanism is used, the document has to be processed *twice* by pdftex—otherwise, the overlaid elements tend to get misaligned.

The `\draw` commands used in the example for the electron push arrows employed the `mcfpusharrow TiKZ` style that is defined by the `mol2chemfig` package. You can adjust this style to your own tastes with the `\tikzset` macro.

If the push-arrows are the *only* drawing element you need, you can avoid the need for explicitly creating an overlaid second `tikzpicture` environment by using macros predefined by either `chemfig` and `mol2chemfig`. The next example illustrates the use of the `\mcfpush` macro defined by `mol2chemfig`.

```
\input{mp}

\colorlet{mcfpusharrowcolor}{red}
\colorlet{mcfbgcolor}{yellow}
\begin{center}
\begin{tikzpicture}[help lines, x=1cm, y=1cm]

\draw[help lines] (0,0) grid (6,3);
\node[anchor=south west, inner sep=10pt] at (1,0) {\chemfig{!{mp}}};

\mcfpush{mp12-13}{60:1.5em}{mp11-12}{60:1.5em}
\mcfpush[-4pt][4pt]{mp2-3}{105:1.5em}{mp3}{105:1.5em}

\end{tikzpicture}
\end{center}
```

which gives



The arguments of the `\mcfpush` macro are, in order, the named anchor of the origin, the parameters of the first control point (departure angle:distance), and the second anchor and second control point (arrival angle:distance). Two optional arguments set the `shorten <=` and `shorten >=` lengths for the arrow in question, which override the global values which can be adjusted using the `\tikzset` mechanism. The `\mcfpush` macro uses the `mcfpusharrow TiKZ` style (see above). This style references two color definitions, which were redefined using `colorlet` (supplied by package `xcolor`) in this example. The back-

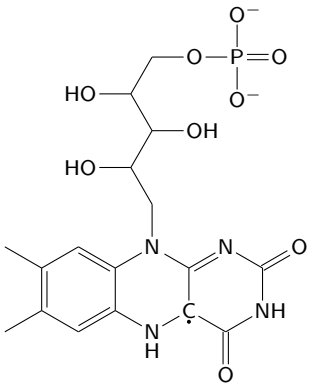
ground color (`mcfbgcolor`) defaults to white; you will only want to change it when drawing on a non-white canvas. This color definition is also used when drawing crossing bonds in the foreground (see section 4.5).

Note that internally the picture overlay mechanism is still used, so the need for processing the file through pdftex twice remains.

5 Invoking `mol2chemfig` from within LaTeX

Using the `--shell-escape` option on Linux or its equivalents on other systems, LaTeX can execute shell commands, capture the output and insert it directly into the document. We can use this with `mol2chemfig`. If you have `mol2chemfig` working and LaTeX properly configured, the following command will insert the structure of FMNH directly into your document, without creating a separate file:

```
\mcfinput{-fw examples/fmnh.mol}
```



Note, however, that with large documents and numerous formulas the overhead of running `mol2chemfig` on each formula during every compilation will add up.

6 `chemfig` settings used in this document

Several settings are offered by `chemfig` to control the appearance of structures in your documents. Below are the settings that were used in this document.

```
% reduce font size and use sans-serif
\renewcommand{\printatom}[1]{%
\fontsize{8pt}{10pt}\selectfont{\ensuremath{\mathsf{#1}}}}

% reduce bond dimensions to match smaller fonts
\setcrambond{2.5pt}{0.4pt}{1.0pt}
\setbondoffset{1pt}
\setdoublesep{2pt}
\setatomsep{16pt}
```

7 Conclusion

This tutorial has covered most capabilities of `mol2chemfig`. There are a few more options that influence the appearance of the output; these should be pretty much self-explanatory.

I hope `mol2chemfig` will be useful to you. If you come across any bugs or issues, please send email to mpalmer@uwaterloo.ca.

8 Acknowledgments

Christian Tellechea wrote the excellent `chemfig` package, upon which `mol2chemfig` is based. He also gave valuable advice and suggestions concerning `mol2chemfig` itself. To the extent that `mol2chemfig` understands chemistry, it owes this to the creators of `indigo`. An earlier version of `mol2chemfig` used `rdkit`; however, after some experimentation, I found that `indigo` was better suited to my purpose. Nevertheless, I thank `rdkit`'s creator, Greg Landrum, for promptly and thoroughly answering all my questions. My student Eric Brefo-Mensah tested the code extensively and uncovered numerous bugs (or rather, a whole plague of locusts). Further bugs were reported by Benjamin Abel, Philipp Bisson, and Vincent Liegeois.