```
#svl
//      vfl.svl  Virtual Framework Library
//
//      30-may-2007 (ad)      Removed MOE source dependencies; updates
//      13-jan-2006 (ad)      Created
//
```

//
// DESCRIPTION
//
//   The goal of the algorithm is to fuse rings forming all non-redundant
//   1-fusion, 1,2-fusion, and 1,2,3-fusion compounds. For the purpose of
//   this description, 1-fusions, 1,2-fusions, and 1,2,3-fusions are
```
```

```
//      defined as two rings that share one, two, or three carbon atoms,
//      respectively. A 1-fusion is only possible with secondary carbons on the
//      base structure. A 1,2-fusion is possible when each of carbon atoms in
//      the base structure is either secondary or tertiary whereas a
//      1,2,3-fusion is only possible for secondary or tertiary carbons at the
//      1, and 3 positions on the base structure. The algorithm starts with a
//      set of base structures that are to be built upon by fusion with a set
//      of ring fragments. All the base structures are modified by 1-, 1,2-,
//      and 1,2,3-fusions to produce a new set of base structures for input
//      into subsequent cycles of the algorithm.
//
//      Structures are stored using MOE's canonical SMILES string
//      representation which allows for the easy removal of duplicates.
//
//
// USAGE
//
//      1. Load this file:
//              (a) Manually load the file with the File | Open panel
//              (b) Save this file in your ~/svl directory
//
//      2. Launch the application
//              (a) In an SVL function, in a MOE menu file or at the SVL
//                  command line, type a command such as:
//
//              vfl no_layers
//              where no_layers is the number of rings to add before exiting
//
//      3. The results are in various files:
//              (a) Rings are in layer_i.txt where i-1 is the number of rings
//              (b) Double bond combinations are added in layer_i_double.txt
//              (c) Methyl groups are added to layer_i.txt and
//                  layer_i_double.txt to form layer_i_methyl.txt and
//                  layer_i_double_methyl.txt
//              (d) layer_i_temp.txt files can be safely ignored
//

const RINGS = [
   'C1CC1',          // cyclopropane addition
   'C1CCC1',         // cyclobutane addition
   'C1CCCC1',        // cyclopentane addition
   'C1CCCCC1',             // cyclohexane addition
   'C1CCCCCC1',      // cycloheptane addition
   'C1CCCCCCC1'      // cyclooctane addition
];
```

```
// generate all pairs between in a vector 1, 2, ..., n
local function pairs n
   local idx = igen n;
   local i2 = cat apt rep [ idx, dec idx ];
   local i1 = cat app igen dec idx;
   return [i1,i2];
endfunction

// find all the possible 1,2,3-addition on a structure
local function generate_triplets_idx atoms
   local akeys, bonds, blen, a1, a2, a3;
   akeys = atoms;
   bonds = aBonds akeys;
   blen = app length bonds;
   a1 = cat apt stretch [bonds, blen];
   a2 = cat apt stretch [akeys, sqr blen];
   a3 = cat apt resize [bonds, sqr blen];
   [a1, a2, a3] = [a1, a2, a3] || [a1 < a3];
   [a1, a2, a3] = [a1, a2, a3] ||
         [andE apt indexof [[a1, a2, a3], [akeys]]];
   return tr apt indexof [[a1, a2, a3], [akeys]];
endfunction

// finds the nearest atoms between two sets of atoms
local function nearest_atoms [atoms1, atoms2]
   local x2 = aPos cat atoms2;
   local x1;
   local dist;

     // generate a distance matrix between each pair of atoms
     // between atoms1 and atoms2
   local i;
   for i in x_id atoms1 loop
         x1 = aPos atoms1(i);

           // find the distance between atom x1 and all atoms2
         dist(i) = add sqr (x1-x2); // (x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2
   endloop

     // generate the indices of the minimum distance entry for each row
     // [[[1, col1_idx], [2, col2_idx], ..., [n, colm_idx]]
   local min_indices = tr [igen length dist, app x_min dist];

     // get the index of the minimum entry [row,col]
   local min_idx = cat get [min_indices,
     // for each of the matrix indices generated earlier
```

```
    // find the overall minimum entry index
        x_min apt get [dist, app second min_indices]];

    // extract the mininum distance from the distance matrix
    // In C, this would be the same as dist[i][j]
  local min_dist = peek[dist, min_idx];

    // get the atom keys corresponding to the minimum distance
  local min_atom1 = atoms1[min_idx(1)];
  local min_atom2 = atoms2[min_idx(2)];

    // return the two atom keys and the distance that separate them
  return [min_atom1, min_atom2, sqrt min_dist];
endfunction

// mutates each hydrogen atoms to methyl group
local function add_methyl chain

  Add_H cat cAtoms chain;
  local atoms = cat cAtoms chain;

    // find all hydrogen atoms
  local h_atoms = atoms | aElement atoms == 'H';

    // mutate, set name, geometry, formal charge, and add hydrogen atoms
  aSetElement [h_atoms, 'C'];
  aSetName [h_atoms, 'C'];
  aSetGeometry [h_atoms, 'sp3'];
  aSetHintLP [h_atoms, 0];
  aSetIon [h_atoms, 0];
  Add_H atoms;

endfunction

// performs 1,1, 1,2, and 1,2,3-additions
local function connect_ring [atoms1, atoms2]

    // check to make sure it is two singles or two couples
  if [[allfalse ((app length [atoms1, atoms2]) == 1) and
        allfalse ((app length [atoms1, atoms2]) == 2) and
        allfalse ((app length [atoms1, atoms2]) == 3)]] then
        exit 'Must pass one, two or three atoms';
  endif

    // get the neighbor lists
  local con_atms1 = aBonds atoms1;
```

```
    local con_atms2 = aBonds atoms2;

      // get the neighbor types
    local con_atms1_e = aElement con_atms1;
    local con_atms2_e = aElement con_atms2;

      // get the connected carbons
    local con_c1 = apt mget[con_atms1, apt eqE[con_atms1_e, 'C']];
    local con_c2 = apt mget[con_atms2, apt eqE[con_atms2_e, 'C']];

      // get the connected hydrogens
    local con_h1 = apt mget[con_atms1, apt eqE[con_atms1_e, 'H']];
    local con_h2 = apt mget[con_atms2, apt eqE[con_atms2_e, 'H']];

      // remove query atoms
    con_c1 = apt diff [con_c1, [atoms1]];
    con_c2 = apt diff [con_c2, [atoms2]];

    oDestroy con_h2;
    oDestroy app first con_h1;
    oDestroy atoms2;
    apt Bond [atoms1, con_c2];

endfunction

// writes out all 1,2,3-additions between a core and a ring to a SMILES file
local function add_layer_triple [out_file, dbmol_core, dbmol_add]
    local smiles;

      // append to output file
    if type out_file === 'tok' then
          out_file = fopen out_file;
          fseek [fsize out_file];
    endif

      // if [] or C1CC1
      // keep original molecule
    if mol_aCount dbmol_add <= 9 then
          //smiles = sm_ExtractUnique cat cAtoms mol_Create dbmol_core;
          //fwrite [out_file, '{},{},{}\n', smiles, smiles, []];
          oDestroy Chains[];
          return;
    endif

      // perturb the original coordinates
      // to avoid being one on top of another
```

```
local dbmol_add_pos = mol_aPos dbmol_add - 11;
dbmol_add(4)(MOL_ATOM_X) = dbmol_add_pos(1);

local chain_core = mol_Create dbmol_core;
local chain_add = mol_Create dbmol_add;

local atoms_core = cat cAtoms chain_core;
local atoms_add = cat cAtoms chain_add;

local smiles_core = sm_ExtractUnique atoms_core;
local smiles_add = sm_ExtractUnique atoms_add;

  // get the bonds between atoms
local nbrlist_core = BondList atoms_core;
local nbrlist_add = BondList atoms_add;

  // get only the carbons that have a hydrogen
local atoms_core_c = uniq (
      (second nbrlist_core) | first aElement nbrlist_core == 'H'
);
local atoms_add_c = uniq (
      (second nbrlist_add) | first aElement nbrlist_add == 'H'
);

  // find all those that has at least one hydrogen atoms
local atoms_core_c_no_h = aElement aBonds atoms_core_c == 'H';
local atoms_add_c_no_h = aElement aBonds atoms_add_c == 'H';

  // make sure that they are carbons and not other hetero atoms
atoms_core_c = atoms_core_c | aElement atoms_core_c == 'C';
atoms_add_c = atoms_add_c | aElement atoms_add_c == 'C';

  // generate all triplets of carbons
local idx_triplets_core = generate_triplets_idx atoms_core_c;
local idx_triplets_add = first generate_triplets_idx atoms_add_c;

local atoms_triplets_core = apt get [[atoms_core_c], idx_triplets_core];
local atoms_triplets_add = get [atoms_add_c, idx_triplets_add];

  // nest for apt down the road
if length atoms_triplets_core === 1 then
      atoms_triplets_core = nest atoms_triplets_core;
endif


  // for each triplet of the core mol, add each triplet of the add mol
```

```
local i_core;
local i_add = 1; // assumption that the added part is symmetrical

  // for each 1,2,3-fusion
for i_core in x_id atoms_triplets_core loop

        // current triplet to add onto the core
    local current_triplet_core = atoms_triplets_core (i_core);
    local current_triplet_add = atoms_triplets_add; // only one triplet

        // create temp chains at the same positions
      local chain_core_temp = mol_Create dbmol_core;
      local chain_add_temp = mol_Create dbmol_add;

        // get the atoms of the core
      local atoms_core_temp = cat cAtoms chain_core_temp;
      local atoms_add_temp = cat cAtoms chain_add_temp;

      local atoms_core_temp_c = atoms_core_temp |
         aElement atoms_core_temp == 'C';
      local atoms_add_temp_c = atoms_add_temp |
         aElement atoms_add_temp == 'C';


        // find closest atoms on temp chain
      local a, d; // temp vars not used
      local atom_core_temp_1;
      local atom_core_temp_2;
      local atom_core_temp_3;
      local atom_add_temp_1;
      local atom_add_temp_2;
      local atom_add_temp_3;

      if length current_triplet_core === 1 then
         current_triplet_core = cat current_triplet_core;
      endif
      [a, atom_core_temp_1, d] = nearest_atoms [
         first current_triplet_core, atoms_core_temp_c
      ];
      [a, atom_core_temp_2, d] = nearest_atoms [
         second current_triplet_core, atoms_core_temp_c
      ];
      [a, atom_core_temp_3, d] = nearest_atoms [
         third current_triplet_core, atoms_core_temp_c
      ];
```

```
        [a, atom_add_temp_1, d] = nearest_atoms [
           first current_triplet_add, atoms_add_temp_c
        ];
        [a, atom_add_temp_2, d] = nearest_atoms [
           second current_triplet_add, atoms_add_temp_c
        ];
        [a, atom_add_temp_3, d] = nearest_atoms [
           third current_triplet_add, atoms_add_temp_c
        ];

         // perform the fusion for a pair of structures
        connect_ring [
           [atom_core_temp_1, atom_core_temp_2, atom_core_temp_3],
           [atom_add_temp_1, atom_add_temp_2, atom_add_temp_3]
        ];

         // get the atoms of the temporary chains, and create a new molecule
         // with only those
        local atoms_new = diff [Atoms[], cat [atoms_core, atoms_add]];
        local chain_new = oCreate 0;
        local residue_new = oCreate chain_new;
        oReparent [atoms_new, residue_new];

         // make a dbmol for the new molecule and capture the smiles string
        local dbmol_new = mol_Extract chain_new;
        smiles = sm_ExtractUnique cat cAtoms chain_new;

         // write result of 1,2,3-fusion along with original structures
        fwrite [out_file, '{},{},{}\n', smiles, smiles_core, smiles_add];

         // remove the temporary stuff
        oDestroy chain_new;
        oDestroy chain_core_temp;
        oDestroy chain_add_temp;

    endloop

     // done with these two
    oDestroy chain_core;
    oDestroy chain_add;
endfunction

// writes out all 1,2-additions between a core and a ring to a SMILES file
local function add_layer_double [out_file, dbmol_core, dbmol_add]

    local smiles;
```

```
  // append to output file
if type out_file === 'tok' then
        out_file = fopen out_file;
        fseek [fsize out_file];
endif

        // keep original structure
if isnull dbmol_add then
        //smiles = sm_ExtractUnique cat cAtoms mol_Create dbmol_core;
        //fwrite [out_file, '{},{},{}\n', smiles, smiles, []];
        oDestroy Chains[];
        return;
endif

  // perturb the original coordinates
  // to avoid being one on top of another
local dbmol_add_pos = mol_aPos dbmol_add - 11;
dbmol_add(4)(MOL_ATOM_X) = dbmol_add_pos(1);

local chain_core = mol_Create dbmol_core;
local chain_add = mol_Create dbmol_add;

local atoms_core = cat cAtoms chain_core;
local atoms_add = cat cAtoms chain_add;

local smiles_core = sm_ExtractUnique atoms_core;
local smiles_add = sm_ExtractUnique atoms_add;


  // get the bonds between atoms
local nbrlist_core = BondList atoms_core;
local nbrlist_add = BondList atoms_add;

  // get only the carbons that have a hydrogen
local atoms_core_c = uniq (
        (second nbrlist_core) | first aElement nbrlist_core == 'H'
);
local atoms_add_c = uniq (
        (second nbrlist_add) | first aElement nbrlist_add == 'H'
);

  // make sure that they are carbons and not other hetero atoms
atoms_core_c = atoms_core_c | aElement atoms_core_c == 'C';
atoms_add_c = atoms_add_c | aElement atoms_add_c == 'C';
```

```
  // generate all pairs of carbons
local idx_pairs_core = pairs length atoms_core_c;
local idx_pairs_add = pairs length atoms_add_c;

  // keep only the pairs that are connected
local atoms_pairs_core = apt get[[atoms_core_c], idx_pairs_core];
atoms_pairs_core = apt mget [
   atoms_pairs_core,
   [
         app add (aBonds (first atoms_pairs_core) ==
         (second atoms_pairs_core))
      ]
];
local atoms_pairs_add = apt get[[atoms_add_c], idx_pairs_add];
atoms_pairs_add = apt mget [
      atoms_pairs_add,
   [
         app add (aBonds (first atoms_pairs_add) == (second atoms_pairs_add))
      ]
];


      // for each pair of the core mol, add each pair of the add mol
local i_core;
local i_add = 1; // assumption that the added part is symmetrical
for i_core in x_id first atoms_pairs_core loop

        // current pair to add onto the core
      local current_pair_core = apt get[atoms_pairs_core, i_core];
      local current_pair_add = apt get [atoms_pairs_add, i_add];

        // create temp chains at the same positions
      local chain_core_temp = mol_Create dbmol_core;
      local chain_add_temp = mol_Create dbmol_add;

        // get the atoms of the core
      local atoms_core_temp = cat cAtoms chain_core_temp;
      local atoms_add_temp = cat cAtoms chain_add_temp;

      local atoms_core_temp_c = atoms_core_temp |
         aElement atoms_core_temp == 'C';
      local atoms_add_temp_c = atoms_add_temp |
         aElement atoms_add_temp == 'C';
```

```
        // find closest atoms on temp chain
      local a, d; // temp vars not used
      local atom_core_temp_1;
      local atom_core_temp_2;
      local atom_add_temp_1;
      local atom_add_temp_2;

      [a, atom_core_temp_1, d] = nearest_atoms [
         first current_pair_core,
         atoms_core_temp_c
      ];
      [a, atom_core_temp_2, d] = nearest_atoms [
         second current_pair_core,
         atoms_core_temp_c
      ];
      [a, atom_add_temp_1, d] = nearest_atoms [
         first current_pair_add,
         atoms_add_temp_c
      ];
      [a, atom_add_temp_2, d] = nearest_atoms[
         second current_pair_add,
         atoms_add_temp_c
      ];

#if 0
        // set label to keep track of connection point
      aSetName [atom_core_temp_1, 'A0'];
      aSetName [atom_core_temp_2, 'A0'];
      aSetName [atom_add_temp_1, 'A1'];
      aSetName [atom_add_temp_2, 'A1'];
#endif

        // perform the 1,2-fusion
      connect_ring [
         [atom_core_temp_1, atom_core_temp_2],
         [atom_add_temp_1, atom_add_temp_2]
      ];

        // get the atoms of the temporary chains, and create new molecule
        // with only those
      local atoms_new = diff [Atoms[], cat [atoms_core, atoms_add]];
      local chain_new = oCreate 0;
      local residue_new = oCreate chain_new;
      oReparent [atoms_new, residue_new];

        // make a dbmol for the new molecule and capture the smiles string
```

```
        local dbmol_new = mol_Extract chain_new;
        smiles = sm_ExtractUnique cat cAtoms chain_new;
        fwrite [out_file, '{},{},{}\n', smiles, smiles_core, smiles_add];

          // remove the temporary stuff
        oDestroy chain_new;
        oDestroy chain_core_temp;
        oDestroy chain_add_temp;

    endloop

        // done with these two
    oDestroy chain_core;
    oDestroy chain_add;
endfunction

// writes out all 1,1-additions between a core and a ring to a SMILES file
local function add_layer_single [out_file, dbmol_core, dbmol_add]

    local smiles;

        // append to end of the file
    if type out_file === 'tok' then
        out_file = fopen out_file;
        fseek [fsize out_file];
    endif

        // keep original structure
    if isnull dbmol_add then
        //smiles = sm_ExtractUnique cat cAtoms mol_Create dbmol_core;
        //fwrite [out_file, '{},{},{}\n', smiles, smiles, []];
        oDestroy Chains[];
        return;
    endif

      // perturb the original coordinates
      // to avoid being one on top of another
    local dbmol_add_pos = mol_aPos dbmol_add - 11;
    dbmol_add(4)(MOL_ATOM_X) = dbmol_add_pos(1);

      // create the two dbmols
    local chain_core = mol_Create dbmol_core;
    local chain_add = mol_Create dbmol_add;

      // get the atoms of the core chain
    local atoms_core = cat cAtoms chain_core;
```

```
  // get the atoms of the add chain
local atoms_add = cat cAtoms chain_add;

local smiles_core = sm_ExtractUnique atoms_core;
local smiles_add = sm_ExtractUnique atoms_add;

  // find all carbon atoms
local atoms_core_c = atoms_core | aElement atoms_core == 'C';
local atoms_add_c = atoms_add | aElement atoms_add == 'C';

  // find all those that has at least one hydrogen atoms
local atoms_core_c_no_h = aElement aBonds atoms_core_c == 'H';
local atoms_add_c_no_h = aElement aBonds atoms_add_c == 'H';

  // keep only the carbons that are bound to two hydrogen atoms
atoms_core_c = atoms_core_c | app add atoms_core_c_no_h == 2;
atoms_add_c = atoms_add_c | app add atoms_add_c_no_h == 2;

local atom_core;
local atom_add = atoms_add_c(1); // assume that added part is symmetrical
for atom_core in atoms_core_c loop

        // create copies of the atoms
      local chain_core_temp = mol_Create dbmol_core;
      local chain_add_temp = mol_Create dbmol_add;

        // get the atom keys of the temporary core chain
        // and move it out of the way
      local atoms_core_temp = cat cAtoms chain_core_temp;

        // get the atom keys of the temporary add chain
      local atoms_add_temp = cat cAtoms chain_add_temp;

        // get the carbon atoms from both chains
      local atoms_core_temp_c = atoms_core_temp |
         aElement atoms_core_temp== 'C';
      local atoms_add_temp_c = atoms_add_temp |
         aElement atoms_add_temp == 'C';


        // find closest atoms on temp chain
      local a, d; // temp vars not used
      local atom_core_temp;
      local atom_add_temp;
      [a, atom_core_temp, d] = nearest_atoms [atom_core, atoms_core_temp_c];
```

```
        [a, atom_add_temp, d] = nearest_atoms [atom_add, atoms_add_temp_c];

#if 0
          // set label to keep track of connection point
        aSetName[[atom_core_temp, atom_add_temp], 'A0'];
#endif

          // perform the 1,1-fusion
        connect_ring [atom_core_temp, atom_add_temp];

          // get the atoms of the temporary chains, and create a new molecule
          // with only those
        local atoms_new = diff [Atoms[], cat [atoms_core, atoms_add]];
        local chain_new = oCreate 0;
        local residue_new = oCreate chain_new;

        oReparent [atoms_new, residue_new];
        aSetName [atoms_new, aElement atoms_new];

        local dbmol_new = mol_Extract chain_new;
        smiles = sm_ExtractUnique cat cAtoms chain_new;

        fwrite [out_file, '{},{},{}\n', smiles, smiles_core, smiles_add];

          // remove the temporary stuff
        oDestroy chain_new;
        oDestroy chain_core_temp;
        oDestroy chain_add_temp;

    endloop
    oDestroy chain_core;
    oDestroy chain_add;
endfunction

// read structures from input file and perform 1,1, 1,2, and 1,2,3-additions
// write results to a new file
local function add_layer [in_file, layer_smiles, out_file]

    oDestroy Chains[];

    if isnull in_file then
        exit 'Must specify input file';
    else
        in_file = fopenr in_file;
    endif
```

```
    if isnull out_file then
        exit 'Must specify output file';
    else
        local out_fkey = fopenw out_file;
    endif

    local atoms = app sm_Build layer_smiles;
    local chains = aChain app first atoms;
    local layer_dbmols = app mol_Extract atoms;
    oDestroy chains;
    local core_smiles;

      // for each SMILES string in input file
    while not isnull (core_smiles = freadb [in_file, 'line', 1]) loop
        local core_atoms = sm_Build token cat core_smiles;
        local core_chains = aChain first core_atoms;
        local core_dbmol = mol_Extract core_chains;
        oDestroy core_chains;
        local layer_dbmol = [];

          // for each ring perform 1,1, 1,2, and 1,2,3-fusions on input
          // structure
        for layer_dbmol in layer_dbmols loop
            add_layer_single[out_fkey, core_dbmol, layer_dbmol];
            add_layer_double[out_fkey, core_dbmol, layer_dbmol];
            add_layer_triple[out_fkey, core_dbmol, layer_dbmol];
      endloop
    endloop
    fclose out_fkey;
    fclose in_file;
endfunction

// read input SMILES and mutate each hydrogen to a methyl group
global function smi_add_methyl [smi_in, smi_out]
    oDestroy Chains[];

    local in_fn, out_fn;
    in_fn = fopenr smi_in;
    out_fn = fopenw smi_out;

    local in_smiles;
    local chain = [];
    local atoms = [];

      // for each SMILES structure, mutate hydrogen atoms to methyl groups
    while not isnull (in_smiles = cat freadb[in_fn, 'line', 1]) loop
```

```
            oDestroy chain;

            atoms = sm_Build token in_smiles;
            chain = aChain first atoms;
            add_methyl chain;
            fwrite [out_fn, '{}\n', string sm_ExtractUnique atoms];

    endloop
    oDestroy chain;

    fclose in_fn;
    fclose out_fn;
endfunction

// verify is bonds would form an cyclic allene by checking for a carbon
// participating in two double bonds
local function hasAllene bonds
    local b, rb;
    b = rb = cat bonds;
    for dec length b loop
            rb = rotl rb;
            if anytrue (b == rb) then
                return 1;
            endif
    endloop
    return 0;
endfunction

// write SMILES for all possible double-bonded configuration of a structure
local function smi_desaturate_rings [dbmol, sizes]

    local smiles = [];

      // hydrogens don't matter
    local mol = mol_Heavy dbmol;

            // copy the query molecule
    local chains = mol_Create mol;
    local atoms = cat cAtoms chains;
local orig_smile = sm_ExtractUnique atoms;

    local rings, ring_sizes;
    rings = SmallestRings atoms;

            // only keep the rings specified in 'sizes'
    ring_sizes = app length rings;
```

```
rings = rings | m_join [ring_sizes, sizes];

  // nothing to do, return empty list
if isnull rings then
   oDestroy chains;
      return;
endif

  // find the indices for valid rings
local ring_atoms = uniq cat rings;
local idx = x_join [atoms, ring_atoms];

oDestroy chains; // don't need these anymore

  // remove all bonds that originate from quaternary carbons
local bonds = mol(4)(MOL_ATOM_BONDS)[idx];
local qc = (idx | (app length bonds == 4));
if length qc then
   [idx, bonds] = [idx, bonds] || [not m_join [idx, qc]];
   bonds = bonds || not orE apt eqE [[bonds], qc];
endif

  // generate an index pair for each potential bond
bonds = uniq app sort cat app tr tag cat [
      [idx],
      [bonds]
];

  // remove unwanted ring connections (outside range of 'sizes')
bonds = bonds | app alltrue apt m_join [bonds, [idx]];

local i;
  // for all combinations of double bond configuration
  // skip first because it will give a mask of 0's
for i=2, pow [2, length bonds] loop
      local mol1 = mol;
      local m = grid_s2m [app length bonds, i] - 1;

        // don't duplicate original structure
      if allfalse m then continue; endif

      local geom = mol1(4)(MOL_ATOM_GEOM);
      local dbonds = bonds | m;
      if hasAllene dbonds then continue; endif;
        // create double bonds
      geom[uniq cat dbonds] = 'sp2';
```

```
        mol1(4)(MOL_ATOM_GEOM) = geom;
        chains = mol_Create mol1;
        smiles = cat [smiles, sm_ExtractUnique cat cAtoms chains];
        oDestroy chains;
    endloop

    return uniq smiles;

endfunction

// write out all possible double bonded configurations for each SMILES in
// an input file
global function smi_add_double_bonds [smi_in, smi_out]

   Close[];

   local in_fn, out_fn;
   in_fn = fopenr smi_in;
   out_fn = fopenw smi_out;

   local in_smiles;
   local chain = [];
   local atoms = [];
   local dbmol = [];
   local out_smiles = [];
     // for each SMILES in input file, add double bonds; write out SMILES
   while not isnull (in_smiles = cat freadb[in_fn, 'line', 1]) loop

        atoms = sm_Build token in_smiles;
        chain = aChain first atoms;

        dbmol = mol_Extract chain;
        oDestroy chain;
          // only for cyclopentane, cyclohexane, cycloheptane
        out_smiles = smi_desaturate_rings [dbmol, [5,6,7]];
        apt fwrite [out_fn, '{}\n', out_smiles];

   endloop
   oDestroy chain;

   fclose in_fn;
   fclose out_fn;

endfunction

// main entry point for generating the VFL
```

```
// This function will take
global function vfl no_layers

   Close[];

        // requires GNU sort and cut (Cygwin under Windows)
   local cygwin_path;
   if MOE_ARCH === 'i4w9' then
        if not tok_length (cygwin_path = getenv 'CYGWIN') then
           exit 'CYGWIN environment variable not set';
        endif

        if not (ftype cygwin_path === 'dir') then
           exit twrite ['File not found: {}', cygwin_path];
        endif
   endif

   local i_counter;
   for i_counter = 0, no_layers loop
        local current_layer_out = tok_cat [
           'layer_',
           totok i_counter,
           '_temp.txt'
        ];
        local in_file = tok_cat ['layer_', totok i_counter, '.txt'];
        if i_counter === 0 then
           local ring = [];
           local current_layer_out_fkey = fopenw current_layer_out;

            // copy original rings to output
           for ring in RINGS loop
                fwrite [current_layer_out_fkey, '{},{},{}\n', ring, ring, ''];
           endloop
           fclose current_layer_out_fkey;
        else
            // add one layer
           write twrite ['Adding layer: {}\n', i_counter];
           add_layer [
                tok_cat ['layer_', totok dec i_counter, '.txt'],
                RINGS,
                current_layer_out
           ];

        endif

           // extract first column and save for next layer addition
```

```
        local command;
        if MOE_ARCH === 'i4w9' then
           command = twrite [
                 '{}\\bin\\cut.exe -d\',\' -f1 {} | {}\\bin\\sort.exe -u > {}',
                 cygwin_path,
                 current_layer_out,
                 cygwin_path,
                 in_file
           ];
        else
           command = twrite [
                 'cut -d\',\' -f1 {} | sort -u > {}',
                 current_layer_out,
                 in_file
           ];
        endif
        write twrite ['Sorting layer: {}\n', i_counter];
        local pkey = exe_open_shell [command, '', [], 1];
        while exe_status pkey loop sleep 0.05; endloop

        write twrite ['Adding double bonds layer: {}\n', i_counter];
        local double_name = tok_cat [fbase in_file, '_double.txt'];
        smi_add_double_bonds [in_file, double_name];

        write twrite ['Adding methyl groups: {}\n', i_counter];
        write '\tSingle bonded structures\n';
        smi_add_methyl [in_file, tok_cat [fbase in_file, '_methyl.txt']];
        write '\tdone\n';
        write '\tDouble bonded structures\n';
        smi_add_methyl [
           double_name,
           tok_cat [fbase double_name, '_methyl.txt']
        ];
        write '\tdone\n';
        write 'done\n';
        write twrite ['done layer: {}\n\n', i_counter];

   endloop

endfunction

#eof
```