Business rules for Tautomer Standardization

Richard Lewis, Paolo Tosco
GDC CADD Novartis, Basel

Introduction

Tautomerisation occurs when a proton migrates from one site to another. The classic example is between 2-hydroxypyridine and pyridone. Tautomerism is an equilibrium between the different states, and depends on factors such as solvent, temperature and overall structure. Ideally, this would be represented as a Boltzmann population, but for 2D databases this is not generally possible. For model-building exercises using 2D descriptors, business rules are needed to specify the most likely state. Baker and coworkers at Syngenta published a series of 473 rules[1] based on QM calculations, to determine the most stable state of two tautomeric forms. The rules were published as RXN files in v2000 and v3000 formats for use in Pipeline Pilot. This works describes the conversion and curation of these rules into Reaction SMARTS for use in RDKit.

Methods

The v2000 format RXN files were converted to SMARTS format using standard tools. For the v3000 format files, a new parser was required, which has since been pushed into the RDKit repo. Each SMARTS component of a rule was curated manually using the SMARTS Viewer from U. Hamburg (SMARTS.plus) as an invaluable aid. Hand-crafted examples were used to test the matches. The main issues came in bond typing, aromaticity and H-counts. 723 rules and 410 exclusions were coded up.

Code for applying the rules was created. Each match needs to applied exhaustively before the next match can be applied, and exceptions need to be corrected for, so that there is a hierarchy of matches. If a rule leads to an error during sanitization, the rule is skipped.

Example rules and exceptions

```
[#6:1]1(=[#8:2])-[#7&H:3]-[#6:4](-[!#1:5]):[#6:6]:[#6:7](-
[!#1:8]):[#7&D2&H0:9]:1>>[#6:1]1(-[#8:2]):[#7&H0:3]:[#6:4](-
[!#1:5]):[#6:6]:[#6:7](-[!#1:8]):[#7:9]:1 CBR18_0a

[#6:1]1(=[#8:2])-[#7&H:3]-[#6:4](-[!#1:5]):[#7&D2:6]:[#6:7](-
[!#1:8]):[#7&D2:9]:1>>[#6:1]1(-[#8:2]):[#7&H0:3]:[#6:4](-
[!#1:5]):[#7:6]:[#6:7](-[!#1:8]):[#7:9]:1 CBR18_0b

[#7&+:1](-[#8,#16;H:2])(=[#8,#16:3])-[!#1:4]>>[#7&+:1](-[#8,#16;-
;D1:2])(=[#8,#16:3])-[!#1:4] CBR18_1

[#7:1](=[#5:2](-[!#1:3])(-[!#1:4])-[!#1:5])(=[!#1:6])-
[!#1:7]>>[#7+:1](-[#5-:2](-[!#1:3])(-[!#1:4])-[!#1:5])(=[!#1:6])-
[!#1:7] CBR18_2

CBR18_0a_excl_1        [#7H]1:[#6]:[#6]:[#6](:[#7]:[#6]:1=[#8,#16])-
[#8,#16;H]

CBR18_0a_excl_2        [#7H]1:[#6]:[#6]:[#6](:[#7]:[#6]:1=[#8,#16])-
[#8,#16;H]

CBR18_0a_excl_3        [#7]1:[#6]:[#6]:[#6](:[#7H]:[#6]:1=[#8,#16])-
[#8,#16;H]

CBR18_0a_excl_4         [#7]1=[#6]=[#6]=[#6](-[#7]-[#6]-1=[#8,#16])-
[#8,#16;H]
```

```
def process_tautomer(Name, MOL, EX, verbose=False):
    '''take a molecule, and as appropriate apply rxns to it, subject to
exclusions
    rxns is an ordered array of reactions, exclusions is a dict indexed by the
reaction name
    return possible altered molecule, Change flag, rxns with updated counts.
Name is just for debugging
    Richard Lewis Oct 2020'''
    Changed = False # record if a mol is changed at all by this routine
    GeneratedSmiles = [] # this is used to check for cycling
    mol = RemoveIndices(copy.deepcopy(MOL)) # as the indices affect the smiles
strings
    Canon = Chem.MolToSmiles(mol) # for testing uniqueness
    GeneratedSmiles.append(Canon)
    molH = Chem.AddHs(mol) # need this for some of the substructure smarts with
explicit H's in them
    mol = mol_with_crank_index(mol) # add the rank property
    for i,rxn in enumerate(rxns):
        ''' Apply rules in order as above
        if match to a reagent, try further
        if matches exclusion for the reaction, continue
        for the number of matches to the reagent query
           find the order of matches using CanonicalRank of the matched atoms
           perform the reaction to create a new mol
           If after all the changes, the molecule cannot be aromatised, revert
to the last valid molecule'''
        Matches = mol.GetSubstructMatches(rxn[0])
        if Matches:
Order = OrderMatch(mol, Matches) #find the canonical order of matches
            for Ind in Order:
                Match = Matches[Ind]
                skip = False
                if rxn[2] in exclusions: # does a reaction have associated
exclusions?
                    for j, esmarts in enumerate(exclusions[rxn[2]]):
                      #print('excl: ', esmarts[2], esmarts[1])
                      if molH.HasSubstructMatch(esmarts[0]):
                        ematches = molH.GetSubstructMatches(esmarts[0])
                        if verbose: print('excl2: ', rxn[2], esmarts[2],
ematches)
                        skip = CheckOverlap(Match, ematches) #is the exclusion
for the same part of the molecule?
                        if skip:
                            exclusions[rxn[2]][j][3] = exclusions[rxn[2]][j][3]
+ 1
                            EX = EX + 1
                            break #out of the exclusions loop
                if skip:
                    Matches = None
                    break # to the next reaction as the match has overlap with
an exclusion
#                print('Past excl')
                ps = rxn[1].RunReactant(mol,0) #or RunReactants([mol])
                if verbose: print('h2 ', len(ps), rxn[2], Name, Canon)
                if len(ps) == 0:
                    if verbose: print('no reaction')
                    continue
                #mol = mol
                else: # a reaction has occurred
                    for j,p in enumerate(ps):
                        if p[0] is None: continue
                        if verbose:
                            print ('product ', Name, rxn[2],
Chem.MolToSmiles(mol), Chem.MolToSmiles(p[0]))
                            Chem.SanitizeMol(p[0]) # hard crash
                        try:
                            Chem.SanitizeMol(p[0])
                            nmol = RemoveIndices(p[0])
                            smi = Chem.MolToSmiles(nmol)
                        except ValueError as e:
                            print ('product rejected due to %s'%e, Name, rxn[2],
Chem.MolToSmiles(mol), Chem.MolToSmiles(p[0]))
                            msg = str(e)
                            if 'Explicit valence' in msg:
                                nmol = FixValence(p[0], msg)
                                if nmol is not None: smi =
Chem.MolToSmiles(nmol)
                            else:
                                continue # try the next product
                        if nmol is None: continue
                        if smi in GeneratedSmiles:
                            continue
                        # nmol is now a new distinct valid molecule
                        NMatches = nmol.GetSubstructMatches(rxn[0]) # check that
the reaction was applied to the right match
                        if NMatches and CheckOverlap(Match, NMatches):
                            # wrong reaction site
                            continue # and get the next product
                        mol = copy.deepcopy(nmol)
                        Canon = Chem.MolToSmiles(mol) # for testing uniqueness
                        GeneratedSmiles.append(Canon)
                        molH = Chem.AddHs(mol) # need this for some of the
substructure smarts with explicit H's in them
                        rxns[i][3] = rxns[i][3]+1 # increment the reaction count
                        if verbose: print('CH: ', Name, rxn[2], Canon )
                        Changed = True
                        break # out of ps loop
            #now back to the next match in the structure.
        #now back to the next reaction
    if Changed: #don't bother if it is the same molecule
        Canon = GeneratedSmiles.pop() # get last valid molecule
        mol = Chem.MolFromSmiles(Canon)
        if verbose: print ('changed structure ', Name, Canon)
return RemoveIndices(mol), rxns, Changed, exclusions, EX
```

Results and Discussion

The application of the rules to the ChemBl database (v30) of 3645829 structures resulted in 107072 changes to structures (2.9%). No prior cleaning or filtering was done. Other data sets showed similar levels of structures with tautomeric potential. 31 rules are triggered >50 times. The effect of changing tautomer on an inhouse cLogP model gave Deltas of 0.1 to 0.7 with a MAE of 0.3, demonstrating that consistency of representation does have a real effect

The exact rules for which tautomeric form should be stored in a database are debatable as the exact equilibrium is a function of solvent and environment. In model building, the form of the 2D structure used to generate descriptors should be consistent, even if it is not 'right'. These 473 tautomer business rules at least try to enforce that, and are more sophisticated with respect to exceptions. There are other commercial programs that can create tautomer populations (Unicon from Rarey group, U. Hamburg, Tauthor from Molecular Discovery), but they handle only the more obvious cases. There is also some existing code (https://rdkit.blogspot.com/2020/01/trying-out-new-tautomer.html). This approach offers that possibility to identify those compounds that can exist as tautomers, using the more sophisticated set of rules from Syngenta. If desired, the program could be used to create input to a more rigorous scoring approach such as QM, using fast methods like xtb with solvent corrections.

Conclusions

The Syngenta rules for assigning tautomer structures have been implemented in RDKit and are available for general use.

References

1. **Tautomer Standardization in Chemical Databases: Deriving Business Rules from Quantum Chemistry** Christopher M. Baker **et al.** *J. Chem. Inf. Model.* 2020, 60, 8, 3781−3791