Rule-based stochastic Batch-Reactor Simulator

Clemens Heiderer

Supervisor Univ.-Prof. Mag. Dr. Christoph Flamm

March 23, 2021

Contents

Abstract.

The task was to implement a simulation to show how the mixture of molecules changes over time upon an action of tranformation rules [1]. For that molecules were labeled with an id number. Gillespie was used as stochastic method for the simulation. Code is written in Python 3. Rdkit [2] was used for reading, writing, and transforming molecules.

1 Master Equation

[3]

We define a discrete state space and transitions rates W between states, where W_{ji} is a transition rate from state i to state j.
The state of the system is described by a state vector p of species counts S.
P = (S_1, S_2,.., S_m), e.g. p = (S_1, S_2).
Here we have for 2 species (S_1, S_2) with 3 reactions following operations on a state vetor.

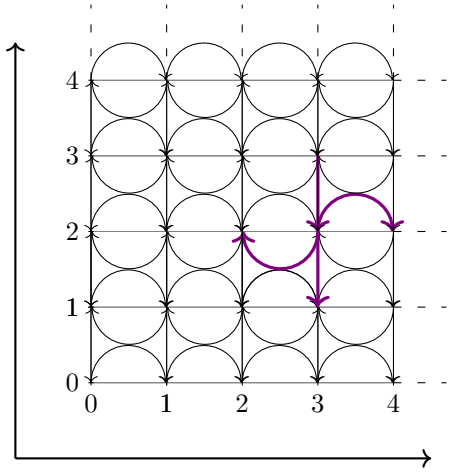


Figure 1: state space of P(3, 2)

Reaction 1: $\phi \rightarrow S_1$–$p(S_1, S_2) \rightarrow p(S_1 + 1, S_2)$

Reaction 2: S_1+$S_2 \rightarrow S_1$–$p(S_1, S_2) \rightarrow p(S_1, S_2$-1)

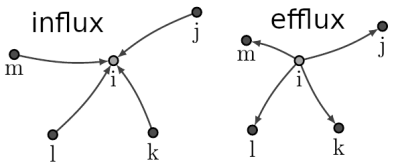
Reaction 3: $S_1 \rightarrow \phi$–$p(S_1, S_2) \rightarrow p(S_1 - 1, S_1)$

A trajectory is as a walk in this state space. States reachable from Point (3, 2) in state space are P (3, 1) by Reaction 2, P (2, 2) by Reaction 3 and P (4, 2) by Reaction 1.
The walk is memory less, the decision to which state to go next, is only based on the current state and not on the information how the state was reached.
We are interested in the probability that one of the set of states {i, j, k, l, m} occupies the system in regard to a continuous time variable t. Describing the state over time with a matrix of transitionrates W depending on time in the state vector is called the master equation.

$$\frac{d\vec{P}}{dt} = \mathbf{W}\vec{P}$$

$$\frac{dp_i(t)}{dt} = \sum_{j \neq i} \left[\underbrace{W_{ij}(t)p_t}_{\text{influx}} - \underbrace{W_{ji}(t)p_i(t)}_{\text{efflux}} \right]$$

p_i..probability that the system is found at state i at time t.



[4]

Figure 2: influx from other states into state i
efflux from state i to other states

$$P(t+1) = WP(t)$$

$$P(t+2) = WP(t+1) \leftarrow\!\!\triangleq Wp(t) \text{..current population vector in matrix of transition rates}$$

$$P(t+\tau) = WP(t+\tau\text{–}1) \leftarrow\!\!\triangleq Wp(t+\tau\text{–}2)$$

$$P(t+\tau) = \underbrace{WW..W}_{\tau \text{ - times}} P(t)$$

$$\boxed{P(t+\tau) = W^{\tau}P(t)}$$

This approach works only for short times and discrete integer time steps.

$$P(t) = e^{tW}P(0)$$

For longer times, to calculate CME on P(t + τ) which requires solving of matrix exponential via talyor expansion.

2 Gillespie

In [5] general for large systems with an unbound state space the CME cannot be solved.
In these cases one uses the Gillespie algorithm which simulates the CME statistically correct.
Gillespie generates a statistically correct [6]trajectory (possible solution) of a stochastic equation system for which the reaction rates are known.
It follows each iteration a classic monte carlo step. As input it needs all reaction rates k, every amount of every chemical species S, and a choosen end time.

- When does my reaction occur
  - First random number p1 in the interval [0-1] gets choosen.

    $$random.uniform(0,1)$$

  - Calculation of rates for each iteration: $W_{pe} = \mathrm{erp}_{\mathrm{product} \leftarrow \mathrm{educt}}\ k$

    $$rates = [erp_1 k_1, erp_2 k_2, .., erp_n k_n]$$

    erp .. embedded reaction possibilities, how many possibilities there are that educts form its product
    k..reaction rate W_tot = sum of my rates
  - tau..time step to next reaction

    $$\tau = \frac{1}{\mathrm{W\_tot} * p_1} \Rightarrow newtime = time + \tau$$

- Which reaction occurs
  - Selecting of second random number p_2 in the interval [0-1].

    if: $\mathrm{p}_2 \cdot$ sum of rates $\leqq$ partial sum of W_i

```
sum = 0
for r in rates:
   sum += r
   if sum > p2 * sum of my rates:
      break
```

    The waiting time in a particular state until the state is left follows an exponential distribution. The steeepness of the curve is increases with a higher W_{tot}. In a deep local minima, the total outflow rate W_{tot} is small, hence the waiting time is large. On a mountain top the W_{tot} is larger, hence the waiting time is short.

    Gillespie algorithm stops with reaching the end time or when the total propensity equals 0, that means if there are no more possibilities available to form a new reaction.

3 Rdkit

is [7] a collection of cheminformatics and machine-learning software written in C++ and Python with an open source BSD license by main author and moderator Gregory Landrum. Core Algorithms and data structures are written in C++. Their application programming interfaces are in Python, C++, Java, KNIME.

3.1 Functionality

- Input/Output: SMILES/SMARTS
- Reactions
- Descriptors
- MCS
- Enhanced stereochemestry
- Molecular standardization
- Depiction
- Fingerprinting:
  · topological torsions
  · Morgan Algorithm
- Thight integration with Jupyter and Pandas
- Clustering (hierachical
- Open3D implementation
- Integration with PyMOL for 3D visualization
- UFF and MMFF94/MMFF94S
- Open 3D Align
- Feature map vectors
- Pharmacore embedding
- Gasteiger-Marsali Charges
- Machine Learning:
  · Clustering
  · Information Theory (Shannon entropy,Information gain..)
- Gasteiger-Marsali Charges
- Molecular standardization

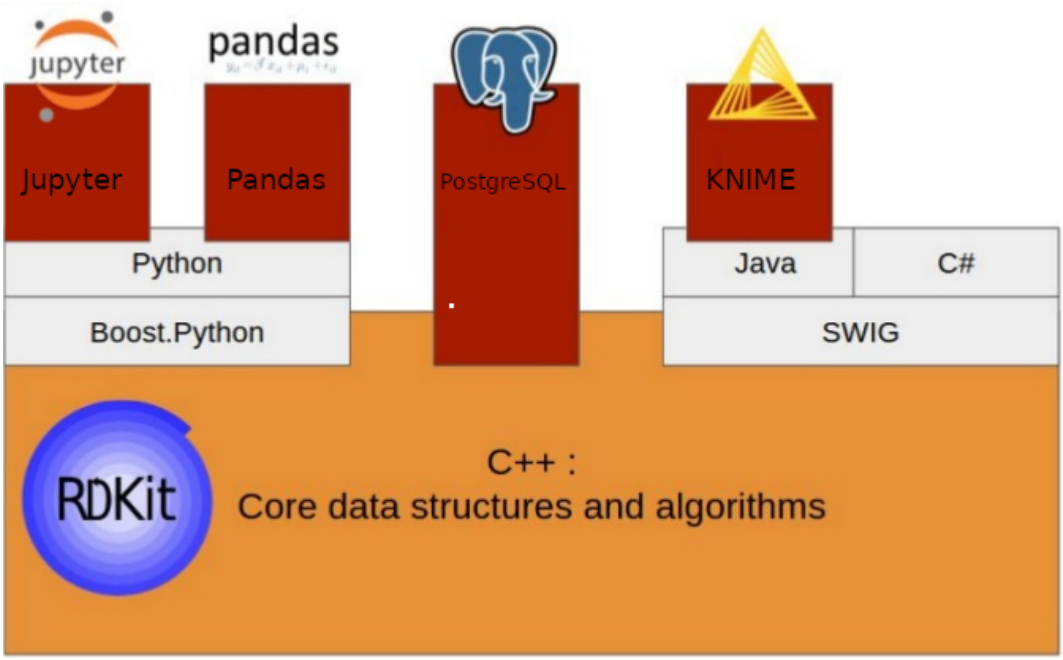


Figure 3: for full list and draw ilustration see G. Landrum presentationn here [8] and there [9].

SMILES/SMARTS

SMILES [10] stands for a Simplified Molecular Input Entry System. Its a line notation for entering and representing molecules and reactions using ASCII strings.
It contains the same information as in an connection table, it takes half the space. Language is controlled by enterprise Daylight. Unique SMILES are called canonical SMILES.

SMILES originate from a graph representation, their string is obtained by printing encountered symbol nodes in a depth-first tree traversal. Symbols are used for atoms and bonds, with them one can specify their molecule's graph (nodes and edges). 11

- atoms are represented by their atomic symbols. Each atom specified independently is enclosed in square brackets ,[ ] by its atomic symbol. Elements in the organic subset B, C, N, O, P, S, F, Cl, Br, and I may be written without brackets.

- bonds single, double, triple and aromatic bonds are represented by the symbol -, =, # and :
  C#N hydrogen cyanid
- branches are enclosed in parenthesis. The implicit connection to a parenthesized expression is to the left.

- cyclic structures are written as connected non-cyclic graph, where bond is broken, the same atom is then represented at the start and end of a SMILE, like: C1CCCCC1.

  There are specific rules for isotopic specifications, like local chirality.
  For adding atomic numbers and hydrons, SMILES have to be specified in the like:
  C(=N)C#N to $'[H:7][N:45] = [C-:45]([C:7]\#[N:7])[H+:28]']$
- for reactions SMILES uses reactant $\gg$ product specification.

SMARTS 12 stands for SMILES arbitrary target specification. It specifies substructural patterns (subgraph) in molecules (graph).
It is used for searching databases for similar structures, highlighting functional groups, specifiying substructures, comparing characterized structeres. Almost all SMILES specifications are valid SMARTS targets.
IN SMARTS the labels for graph's nodes and edges are extended to include "logical operators" and special atomic and bond symbols, it allows them to be more general. [C, N] stands for either of both atoms, ˜ matches any bond, [*H2] means any atom with exactly two hydrogens attached. Daylight describes a full list of supported atomic properties. In addition SMARTS knows logical operators.

Main difference between SMILES and SMARTS is, that SMILES describes molecules, whereas SMARTS describe patterns, which are subjected to searching.

SMIRKS [13] is a line notation, they represent types of generic reactions.

Monthly contributions to RDKit are posted by Gregory Landrum on his blogspot. [14]

For installation it is advised to use miniconda and combine it with jupyter notebook or google colab.
An overview of how to use RDKit functionality for Python offeres Getting Started with RDKit in Python. [15]
MDL Blocks are available, which allows to add 3D coordinates. Section Atom Map Indices in SMARTS, describes how to attach indices to the atoms in SMARTS pattern. GetAtomMapNum() library reads atoms from a molecule. From stored atom string indices position, where all relevant atoms are assigned an atom map number property can be obtained with atom.GetIdx().
When using the query on a molecule with GetSubstructureMatches() you can get the indices af all matching atoms.

4 Implementation

At beginning a list of tuples, consisting of my wished molecule SMILES, and their IDs, begining with 1 in integer are created.

My Gillespie starts with reading all my flasks into a while loop, it breaks off if the next timestep tau + last time is greater then set end time is reached, or the total propensity reaches 0, means if there are no more possibilities to form from my educt flask a product. Weighting is done For test pattern search I enumerate through all my flasks, translate all flask strings with the library Chem.MolTolToSmiles() to SMILES. Weighting will be simple determined by my reaction konstants. For my gillespie functions are avoided and flask strings are used directly, all translating to SMILEs is done after termination of the gillespie algorithm.

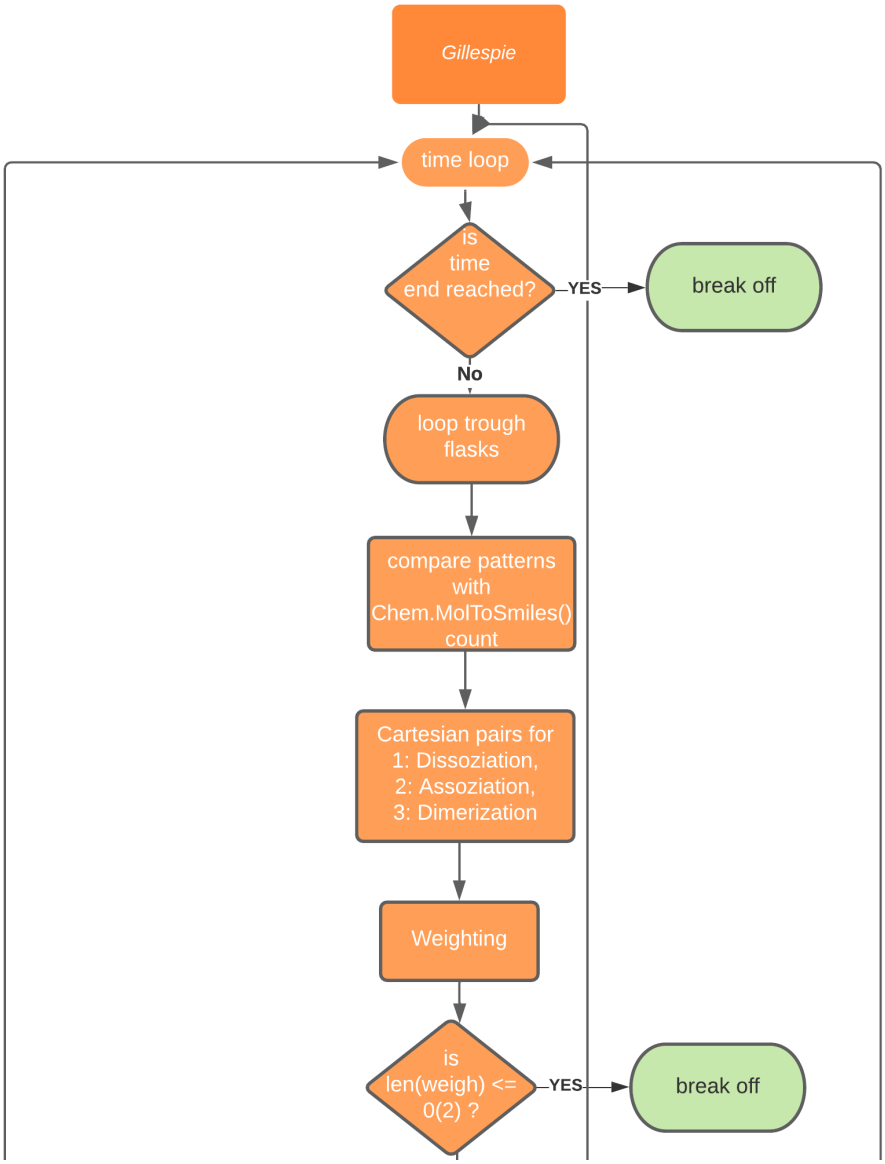


Figure 4

rdkit.Chem package
rdkit.Chem.AllChem module: Imports all RDKit chemistry modules

Following Rdkit libraries16 were used:

- rdkit.Chem.rdmolops.AddHs: Adds hydrogens to the graph of a molecule.

- SetAtomMapNum((Atom)self, (int)mapno[, (bool)strict=False]): Sets the atoms map number, a value of 0 clears the atom map.

- rdkit.Chem.rdmolfiles.MolFromSmiles((AtomPairsParameters)SMILES, (SmilesParserParams)params) → Mol
  Constructs a molecule from a SMILES string. Uses a smile string as argument, returns a mol object.

- $GetAtoms((Mol)arg1) \rightarrow_R OAtomSeq$: Returns a read-only sequence containing all of the molecule's Atoms

- $rdkit.Chem.rdChemReactions.ReactionFromSmarts((str)SMARTS[, (dict)replacements = [, (bool)useSmiles = False]])$: constructs a ChemicalReaction from a reaction SMARTS string.

- $rdkit.Chem.rdmolfiles.MolFromSmarts((AtomPairsParameters)SMARTS[, (bool)mergeHs = False [, (dict)replacements =]]) \rightarrow Mol$:
  Constructs a molecule from a SMARTS string

- $GetSubstructMatches((Mol)self, (Mol)query[, (bool)uniquify = True[, (bool)useChirality = False [, (bool)useQueryQueryMatches = False[, (int)maxMatches = 1000]]]]) \rightarrow object$:
  Returns tuples of the indices of the molecule's atoms that match a substructure query. Allows the use of Chirality

- $GetAtomMapNum((Atom)arg1) \rightarrow int$:
  Gets the atoms map number, returns 0 if not set.

- $RunReactant((ChemicalReaction)arg1, (AtomPairsParameters)arg2, (int)arg3) \rightarrow object$:
  applies the reaction to a single reactant

- $GetAtoms((Mol)arg1) \rightarrow_R OAtomSeq$:
  Returns a read-only sequence containing all of the molecule's Atoms.

- $GetAtomWithIdx((Mol)arg1, (int)arg2) \rightarrow Atom$:
  Returns a particular Atom.

4.1 Reactiongenerator

Dissoziationreaction, Assoziationreaction, Dimerizationreaction in Rdkit

$dis = AllChem.ReactionFromSmarts($
$'[H:1][C:2]\#[N:3] >> [H+:1].[C-:2]\#[N:3]')$

$ass = AllChem.ReactionFromSmarts($
$'[H+:1].[C-:2]\#[N:3] >> [H:1][C:2]\#[N:3]')$

$dim = AllChem.ReactionFromSmarts('$
$[H:1][C:2]\#[N:3].[H+:4].[C-:5]\#[N:6] >> [H:4][C:5](=[N:6][H:1])[C:2]\#[N:3]')$

Figure 5 A reaction generator starts with defining templates of all reactions with the library AllChem.ReactionFromSmarts(). Function inputmol adds first with library Chem.AddHs(Chem.MolFromSmiles(SMILE)) hydrons, then with method GetAtoms() atom IDs are added.
As Input flask IDs are given to the method dis.RunReactants([flask[i]]) which produces product output strings. Pattern SMILES are written down, library Chem.MolFromSmarts(pattern_with_atom_numbers) construct therefrom a molecule.

Structure is used by educt.GetSubstructMatches(e0_patt) to assign tuples of indices of constructed pattern molecules with flask(i) string (*here educt*), which is either my choosen flask(i) educt string or my converted product string (*here each output product id string from dis.RunReactants([flask[i]]).*

Get.ATomMapNum() is used to create a list of atom IDs of structered pattern (*here [1, 2, 3]*) and another which creates a list of atom IDs of assigned educt (*here [2, 2, 2]*).
Both get merged to a dictionary *1: 2, 2: 2, 3: 2.* They are so correctly ordered embedded, they will be used to assign IDs to products.

mol.GetSubstructMatches(pattern_structuere) assigns tuples of indices product pattern structeres to the product output strings (*mol is a variable assigned from any Chem.AddHs(Chem.MolFromSmiles(SMILE)) string outputs*). Index 0 (*like matches[0]*) gives me the tupel of their atom indices.

GetAtomMapNum() transforms constructed pattern in a list of atom IDs (here [1], [2, 3]).

Both lists get zipped together to a dictionary (*x from matches, y from GetAtomMapNum()*). Position on x is, to get my atoms from mol (*all product strings*) with mol.GetAtomWithIdx(x).
y is used to embedd my atom ID integers (*all atoms of one molecule got the same integer at start*) with SetAtomMapNum().
Translation of product string in SMILES with Chem.MolToSmiles.

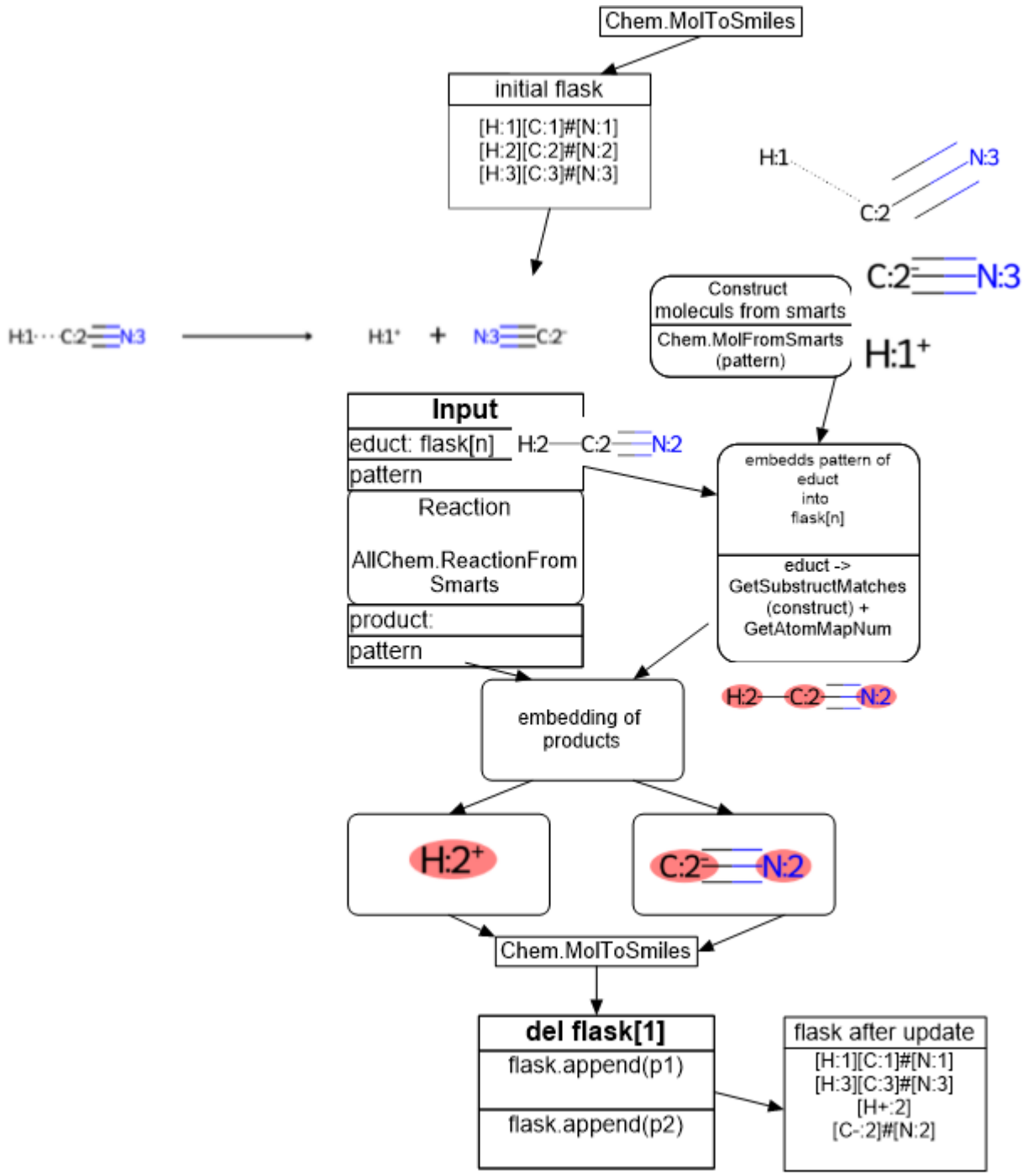
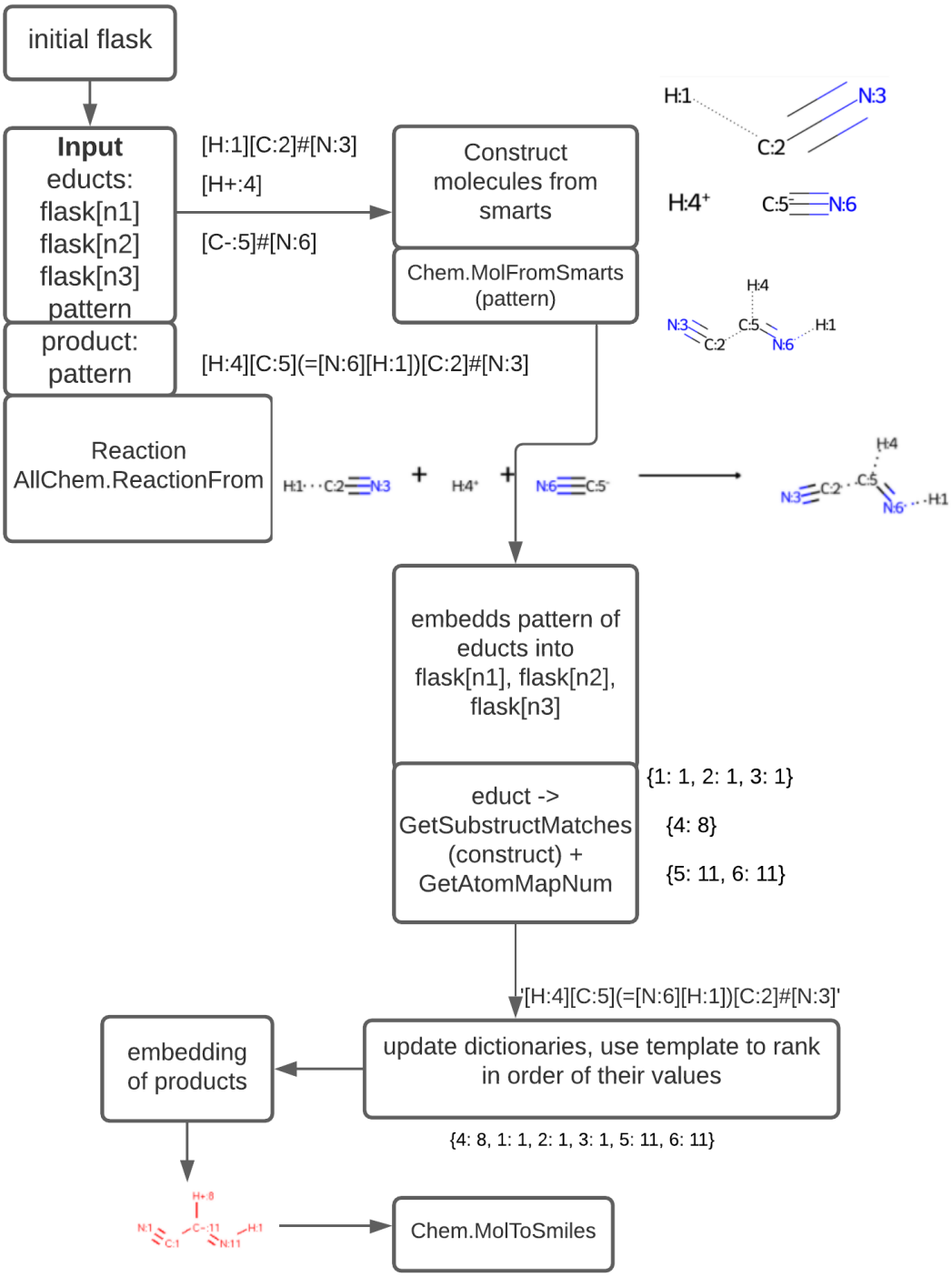


Figure 5: legend for Dissoziationreaction [17]
program was written with python 3.6, RDKit version: '2017.09.1'
pictures were taken from RDKit version 2020_03, python 3.8



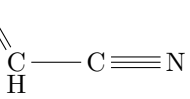
[18]

Figure 6: legend for Dimerizationreaction

4.2 Building Cartesian Product in current flask

Dissoziation:19 $HC{\equiv}N \longrightarrow H^+ + {}^-C{\equiv}N$

Assoziation: $H^+ + {}^-C{\equiv}N \longrightarrow HC{\equiv}N$

Dimerization: $HC{\equiv}N + {}^-C{\equiv}N + H^+ \longrightarrow$ 

[H+:7]
[H+:6]
[H:5][C:5]#[N:5]
[C-:4]#[N:4]
[H:3][C:3]#[N:3]
[C-:2]#[N:2]
[H+:1]
[H:0][C:0]#[N:0]

Figure 7: current flask

Lets assume we run in the current flask our 3 reactions.
To know which kind auf reaction may occur we simple consider all possible molecule pairing who could form a reaction. In our case either one of both hydrogen cyanides with the IDs [5] and [3] could transform to a hydron and a cyanide ion.
For the assoziation reaction we have 6 possible ways which of our educts may react to forminitrile.
So we simple build the cartesian products of our 2 sets, denoted20 $[C-]\#N \times H+$which are all sets of all ordered pairs where flask ID of any cyanide molecule is in set $[C-]\#N$ and flask ID of any hydron is in set $H+$.

$$[C-]\#N \times H+ := \{(c,h) \mid c \in [C-]\#N, h \in H+\}$$

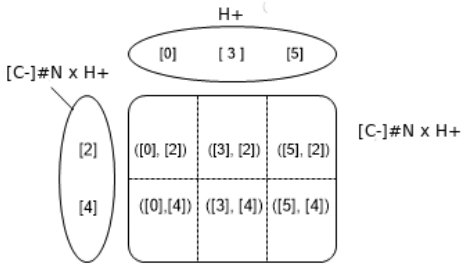


Figure 8: assoziation table of cartesian pair tupels

Our molecule pairs for our assoziation reaction look like this: [(0, 2), (0, 4), (3, 2), (3, 4), (5, 2), (5, 4)].

On Figure 8 you can see both tables for building the cartesian product with cells of all ordered pairs, on top for the assoziationreaction and below for the dimerization reaction we get: [(0, 2), (0, 4), (3, 2), (3, 4), (5, 2), (5, 4)] Figure 9: Our 18 pairs for dimerization reaction are: [(0, 2, 1), (0, 2, 6), (0, 2, 7), (0, 4, 1), (0, 4, 6), (0, 4, 7), (3, 2, 1), (3, 2, 6), (3, 2, 7), (3, 4, 1), (3, 4, 6), (3, 4, 7), (5, 2, 1), (5, 2, 6), (5, 2, 7), (5, 4, 1), (5, 4, 6), (5, 4, 7)].

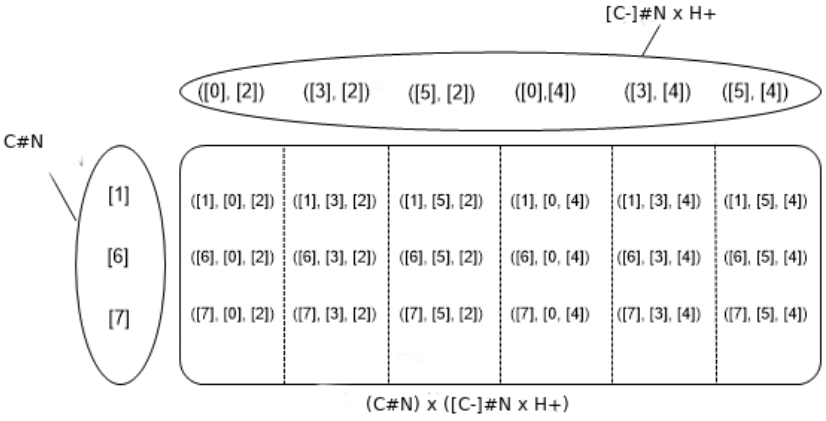
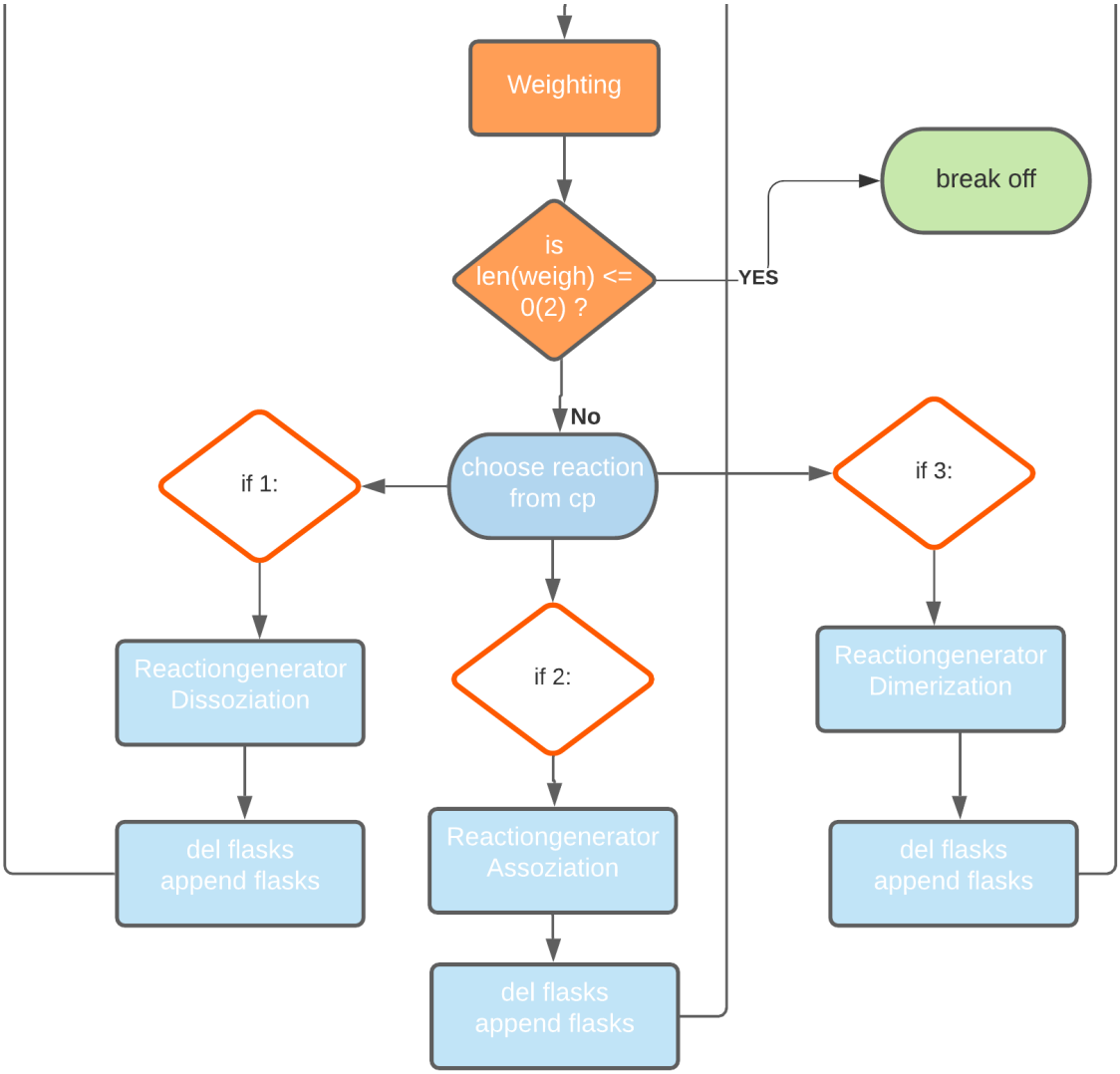


Figure 9: table of cartesian pairs for dimerization

Together our educts from all reactions have 26 possibilities to embed as products in the flask listed as rea_rule.
Each pair-tupel gets assigned the number of its reaction rate. For k_diss: 0.7, k_ass: 0.6, and k_dim weights_rea = [0.7, 0.7, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3].
Our random number for choosing our tuple in the correct proportion do we get from random.choices with all cartesian tuples and weights_rea as input.



4.3 Choosing time interval tau for next step

For identifying which cartesian tuple we got, we just used length calculation of the cartesian tuples, for a more common method a pattern search would be appropriate.
As time pases with each cycle the amount of possible cartesian pairs reduces. If length of rea_rule == 0, we break off the gillespie algorithm. Otherwise length of cartesian tuples determines the reaction generator we choose.
Length of the next time step tau gets selected from an exponential distribution.

$$tau = \textbf{numpy.random.exponential(1/rate\_sum)}$$

21
Rate_sum is just the sum of list weights_rea.
After choosing length of rea_rule our listed patterns are readed in, educts are stored in directly as flask[real_rule[i]], where i is the position in the kartesian tuple, which are directly translated by library RunReactant into product strings like: $((< rdkit.Chem.rdchem.Molat0x284f7919c90 >, < rdkit.Chem.rdchem.Molat0x284f8fa4fa8 >),)$. Template pattern of the particular, reaction is used by method AllChem.ReactionFromSmarts. After running a function from reaction generator flasks[rea_rule[i]] are appended. For deleting all used flask IDs simultaneously we enumerate through a frozenset.

5 Data

Output of Gillespie.txt consists of a timeline list, furthermore a list of cycle number, with its corresponding transformer(flasks).
List IDs of molecule names per cycle gets printed out, likewise choosen cartesian and name of operating reaction.
For molecule presentation all strings were translated into SMILES with library Chem.MolToSmiles after run of Gillespie algorithm.

```
cycle: 12
transformer(flask): ['[H:1][C:1]#[N:1]', '[H:2][C:2]#[N:2]', '[H:3][C:3]#[N:3]', '[H:4][C:4]#[N:4]', '[H:5][C:5]#[N:5]', '[H:6][C:6]#[N:6]', '[H:7][C:7]#[N:7]', '
39]', '[C-:41]#[N:41]', '[C-:43]#[N:43]', '[C-:46]#[N:46]', '[C-:47]#[N:47]', '[H:8][N:35]=[C-:35]([C:8]#[N:8])[H+:27]', '[H:13][N:31]=[C-:31]([C:13]#[N:13])[H+:2
[C-:38]#[N:38]', '[H+:24][C-:42]#[N:42]', '[H:10][N:40]=[C-:40]([C:10]#[N:10])[H+:25]', '[H:15][N:44]=[C-:44]([C:15]#[N:15])[H+:16]', '[H+:20][C-:36](=[N:36][H+:2
list_id_hcn: [0, 1, 2, 3, 4, 5, 6, 7, 8, 24]
10
list_id_cn: [12, 13, 14, 15, 16, 17, 18]
7
list_id_h: [9, 10, 11]
list_id_imino: [19, 20, 21, 22, 25, 26]
6
amount_imino [0, 1, 2, 2, 2, 3, 4, 4, 4, 5, 6, 6]
cartesian: [[0], [1], [2], [3], [4], [5], [6], [7], [8], [24], (9, 12), (9, 13), (9, 14), (9, 15), (9, 16), (9, 17), (9, 18), (10, 12), (10, 13), (10, 14), (10, 1
0), (0, 12, 11), (0, 13, 9), (0, 13, 10), (0, 13, 11), (0, 14, 9), (0, 14, 10), (0, 14, 11), (0, 15, 9), (0, 15, 10), (0, 15, 11), (0, 16, 9), (0, 16, 10), (0, 16
3, 9), (1, 13, 10), (1, 13, 11), (1, 14, 9), (1, 14, 10), (1, 14, 11), (1, 15, 9), (1, 15, 10), (1, 15, 11), (1, 16, 9), (1, 16, 10), (1, 16, 11), (1, 17, 9), (1,
(2, 13, 11), (2, 14, 9), (2, 14, 10), (2, 14, 11), (2, 15, 9), (2, 15, 10), (2, 15, 11), (2, 16, 9), (2, 16, 10), (2, 16, 11), (2, 17, 9), (2, 17, 10), (2, 17, 11
9), (3, 14, 10), (3, 14, 11), (3, 15, 9), (3, 15, 10), (3, 15, 11), (3, 16, 9), (3, 16, 10), (3, 16, 11), (3, 17, 9), (3, 17, 10), (3, 17, 11), (3, 18, 9), (3, 18
4, 11), (4, 15, 9), (4, 15, 10), (4, 15, 11), (4, 16, 9), (4, 16, 10), (4, 16, 11), (4, 17, 9), (4, 17, 10), (4, 17, 11), (4, 18, 9), (4, 18, 10), (4, 18, 11), (5
(5, 15, 10), (5, 15, 11), (5, 16, 9), (5, 16, 10), (5, 16, 11), (5, 17, 9), (5, 17, 10), (5, 17, 11), (5, 18, 9), (5, 18, 10), (5, 18, 11), (6, 12, 9), (6, 12, 10
1), (6, 16, 9), (6, 16, 10), (6, 16, 11), (6, 17, 9), (6, 17, 10), (6, 17, 11), (6, 18, 9), (6, 18, 10), (6, 18, 11), (7, 12, 9), (7, 12, 10), (7, 12, 11), (7, 13
6, 10), (7, 16, 11), (7, 17, 9), (7, 17, 10), (7, 17, 11), (7, 18, 9), (7, 18, 10), (7, 18, 11), (8, 12, 9), (8, 12, 10), (8, 12, 11), (8, 13, 9), (8, 13, 10), (8
(8, 17, 9), (8, 17, 10), (8, 17, 11), (8, 18, 9), (8, 18, 10), (8, 18, 11), (24, 12, 9), (24, 12, 10), (24, 12, 11), (24, 13, 9), (24, 13, 10), (24, 13, 11), (24,
7, 9), (24, 17, 10), (24, 17, 11), (24, 18, 9), (24, 18, 10), (24, 18, 11)]
[0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.3, 0
3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3,
0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.
3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3,
0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3]
r is: [(24, 15, 9)]
choosen cartesian of this reaction: [24, 15, 9]
length of rea_rule: 3
rates: [4.0, 16.8, 63.0]
83.8
rea_rule == 3 - dimerization


cycle: 13
transformer(flask): ['[H:1][C:1]#[N:1]', '[H:2][C:2]#[N:2]', '[H:3][C:3]#[N:3]', '[H:4][C:4]#[N:4]', '[H:5][C:5]#[N:5]', '[H:6][C:6]#[N:6]', '[H:7][C:7]#[N:7]', '
43]#[N:43]', '[C-:46]#[N:46]', '[C-:47]#[N:47]', '[H:8][N:35]=[C-:35]([C:8]#[N:8])[H+:27]', '[H:13][N:31]=[C-:31]([C:13]#[N:13])[H+:26]', '[H:12][N:34]=[C-:34]([C
=[C-:40]([C:10]#[N:10])[H+:25]', '[H:15][N:44]=[C-:44]([C:15]#[N:15])[H+:16]', '[H+:20][C-:36](=[N:36][H+:23])[C-:45]#[N:45]', '[H+:17][C-:41](=[N:41][H+:24])[C-:
list_id_hcn: [0, 1, 2, 3, 4, 5, 6, 7, 8]
9
list_id_cn: [11, 12, 13, 14, 15, 16]
6
list_id_h: [9, 10]
list_id_imino: [17, 18, 19, 20, 22, 23]
6
amount_imino [0, 1, 2, 2, 2, 3, 4, 4, 4, 5, 6, 6, 6]
cartesian: [[0], [1], [2], [3], [4], [5], [6], [7], [8], (9, 11), (9, 12), (9, 13), (9, 14), (9, 15), (9, 16), (10, 11), (10, 12), (10, 13), (10, 14), (10, 15), (
5, 10), (0, 16, 9), (0, 16, 10), (1, 11, 9), (1, 11, 10), (1, 12, 9), (1, 12, 10), (1, 13, 9), (1, 13, 10), (1, 14, 9), (1, 14, 10), (1, 15, 9), (1, 15, 10), (1,
5, 9), (2, 15, 10), (2, 16, 9), (2, 16, 10), (3, 11, 9), (3, 11, 10), (3, 12, 9), (3, 12, 10), (3, 13, 9), (3, 13, 10), (3, 14, 9), (3, 14, 10), (3, 15, 9), (3, 1
4, 10), (4, 15, 9), (4, 15, 10), (4, 16, 9), (4, 16, 10), (5, 11, 9), (5, 11, 10), (5, 12, 9), (5, 12, 10), (5, 13, 9), (5, 13, 10), (5, 14, 9), (5, 14, 10), (5,
4, 9), (6, 14, 10), (6, 15, 9), (6, 15, 10), (6, 16, 9), (6, 16, 10), (7, 11, 9), (7, 11, 10), (7, 12, 9), (7, 12, 10), (7, 13, 9), (7, 13, 10), (7, 14, 9), (7, 1
3, 10), (8, 14, 9), (8, 14, 10), (8, 15, 9), (8, 15, 10), (8, 16, 9), (8, 16, 10)]
[0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0
3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3,
0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3]
r is: [(10, 11)]
choosen cartesian of this reaction: [10, 11]
length of rea_rule: 2
rates: [3.6, 9.600000000000001, 32.4]
45.6
rea_rule == 2 - assoziation
```

Figure 10

$\text{amount}_{cn} = [18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 5, 4, 5, 4, 5, 4]$

$\text{amount}_{hydrons} = [14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 1, 0, 1, 0, 1, 0]$

$\text{amount}_{imino} = [0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 9, 10, 11, 11, 12, 12, 12, 12, 12]$

$\text{amount}_{hcn} = [15, 14, 15, 14, 13, 12, 11, 10, 9, 8, 7, 8, 7, 6, 5, 4, 3, 2, 3, 2, 1]$


```
all_cycles = ['[H:7][C:7]#[N:7]', '[H+:28]', '[C−:45]#[N:45]', '[H:7][N:45]=[C−:45]([C:7]#[N:7])[H+:28]', '[H+:19]', '[C−:33]#[N:33]',
    '[H+:19][C−:33]#[N:33]', '[H:9][C:9]#[N:9]', '[H+:25]', '[C−:46]#[N:46]', '[H:9][N:46]=[C−:46]([C:9]#[N:9])[H+:25]', '[H:2][C:2]#[N:2]',
    '[H+:24]', '[C−:43]#[N:43]', '[H:2][N:43]=[C−:43]([C:2]#[N:2])[H+:24]', '[H:4][C:4]#[N:4]', '[H+:16]', '[C−:44]#[N:44]',
    '[H:4][N:44]=[C−:44]([C:4]#[N:4])[H+:16]', '[H:8][C:8]#[N:8]', '[H+:18]', '[C−:37]#[N:37]', '[H:8][N:37]=[C−:37]([C:8]#[N:8])[H+:18]',
    '[H:1][C:1]#[N:1]', '[H+:20]', '[C−:35]#[N:35]', '[H:1][N:35]=[C−:35]([C:1]#[N:1])[H+:20]', '[H:5][C:5]#[N:5]', '[H+:26]', '[C−:40]#[N:40]',
    '[H:5][N:40]=[C−:40]([C:5]#[N:5])[H+:26]', '[H:12][C:12]#[N:12]', '[H+:23]', '[C−:47]#[N:47]', '[H:12][N:47]=[C−:47]([C:12]#[N:12])[H+:23]',
    '[H:3][C:3]#[N:3]', '[H+:22]', '[C−:31]#[N:31]', '[H:3][N:31]=[C−:31]([C:3]#[N:3])[H+:22]', '[H+:21]', '[C−:38]#[N:38]', '[H+:21][C−:38]#[N:38]',
    '[H+:19][C−:33]#[N:33]', '[H+:27]', '[C−:41]#[N:41]', '[H+:19][N:41]=[C−:41]([H+:27])[C−:33]#[N:33]', '[H:10][C:10]#[N:10]', '[H+:17]',
    '[C−:36]#[N:36]', '[H:10][N:36]=[C−:36]([C:10]#[N:10])[H+:17]', '[H:6][C:6]#[N:6]', '[H+:29]', '[C−:42]#[N:42]',
    '[H:6][N:42]=[C−:42]([C:6]#[N:6])[H+:29]', '[H:11][C:11]#[N:11]', '[H+:11]', '[C−:11]#[N:11]', '[H:13][C:13]#[N:13]', '[H+:11]', '[C−:32]#[N:32]',
    '[H+:11][C−:32]([C:13]#[N:13])=[N:32][H:13]', '[H:15][C:15]#[N:15]', '[H+:15]', '[C−:15]#[N:15]', '[H+:15]', '[C−:39]#[N:39]',
    '[H+:15][C−:39]#[N:39]', '[H+:15][C−:39]#[N:39]', '[H+:15]', '[C−:39]#[N:39]', '[H+:21][C−:38]#[N:38]', '[H+:15]', '[C−:30]#[N:30]',
    '[H+:15][C−:30](=[N:30][H+:21])[C−:38]#[N:38]']
```


5.1 Gillespie

Start with 15 forminitrile, 14 hydrons, 18 cyanide ions and 0 methanimidoyl cyanide. 21 iterations.

Figure 11: k_diss: 1, k_ass: 1, k_dim: 1, graphic generated with plotly, allows hoovering over molecule smiles on each cycle

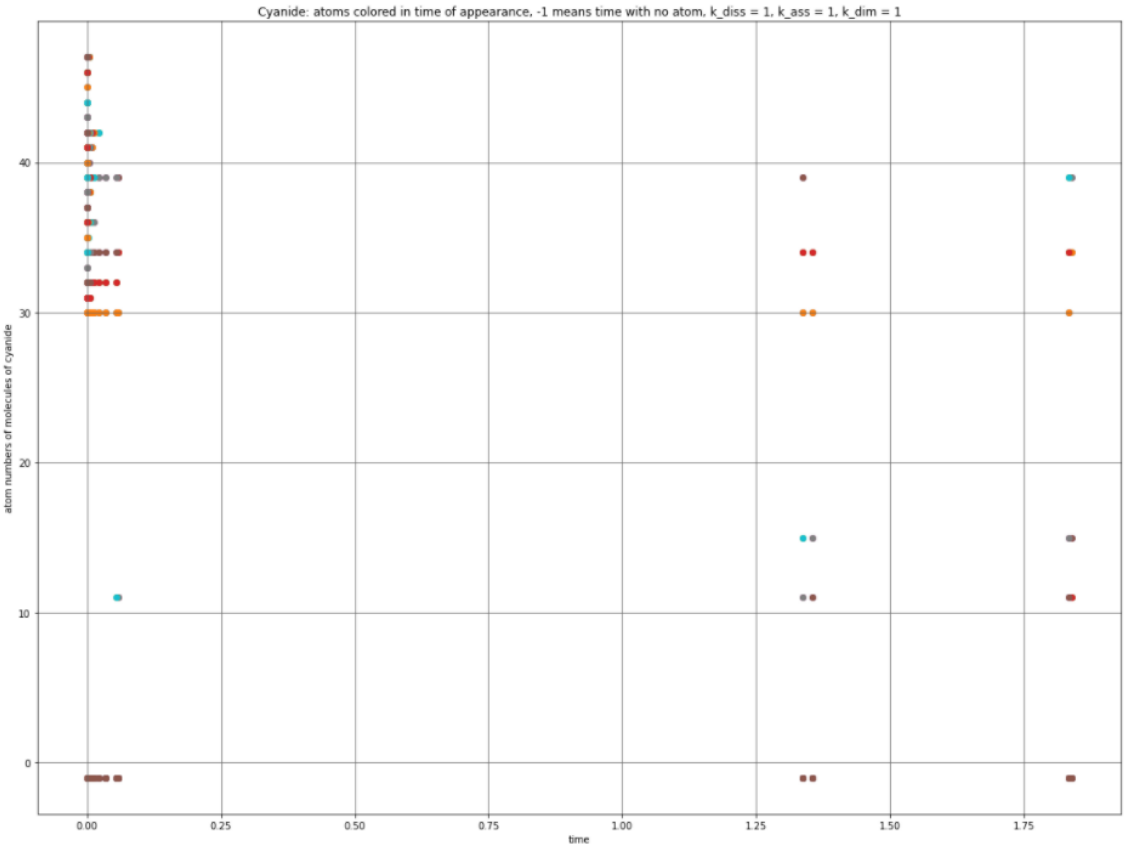


Figure 12: cyanide over time

Figure 13: formonitrile over time

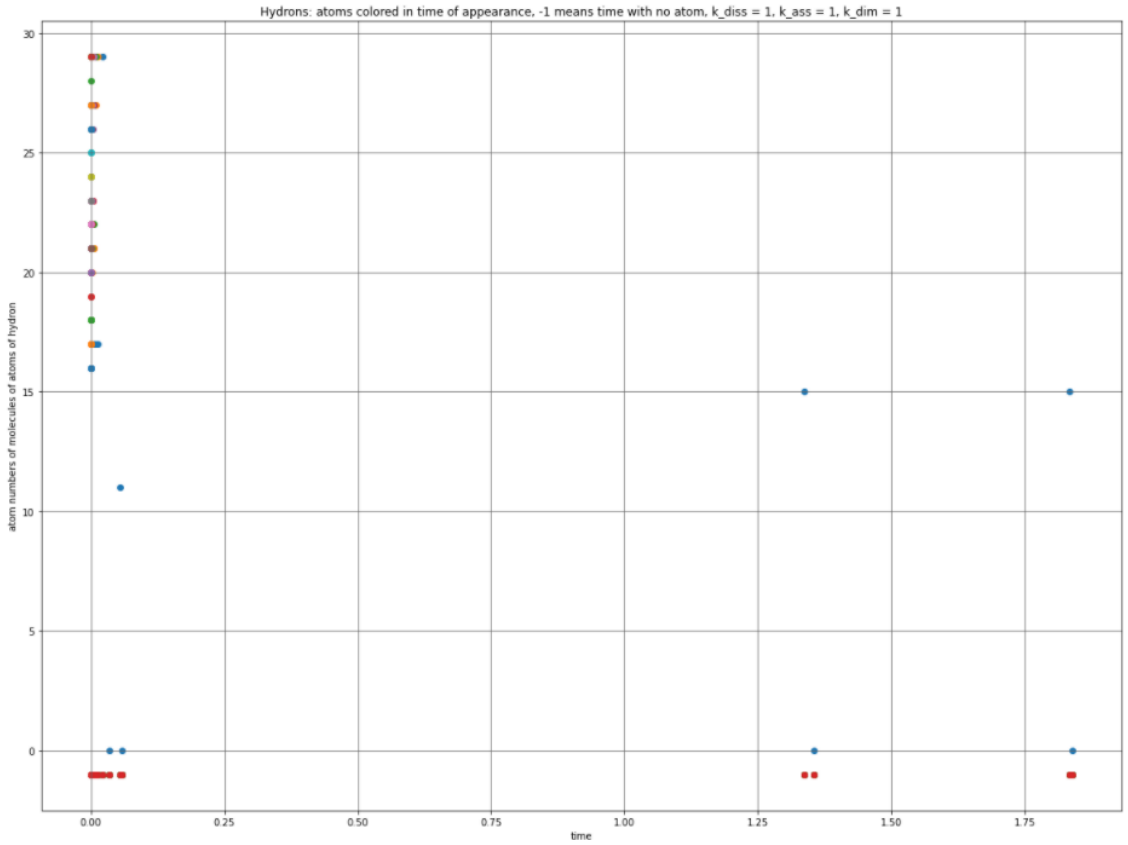


Figure 14

Figure 15: iminoacetonitrile over time

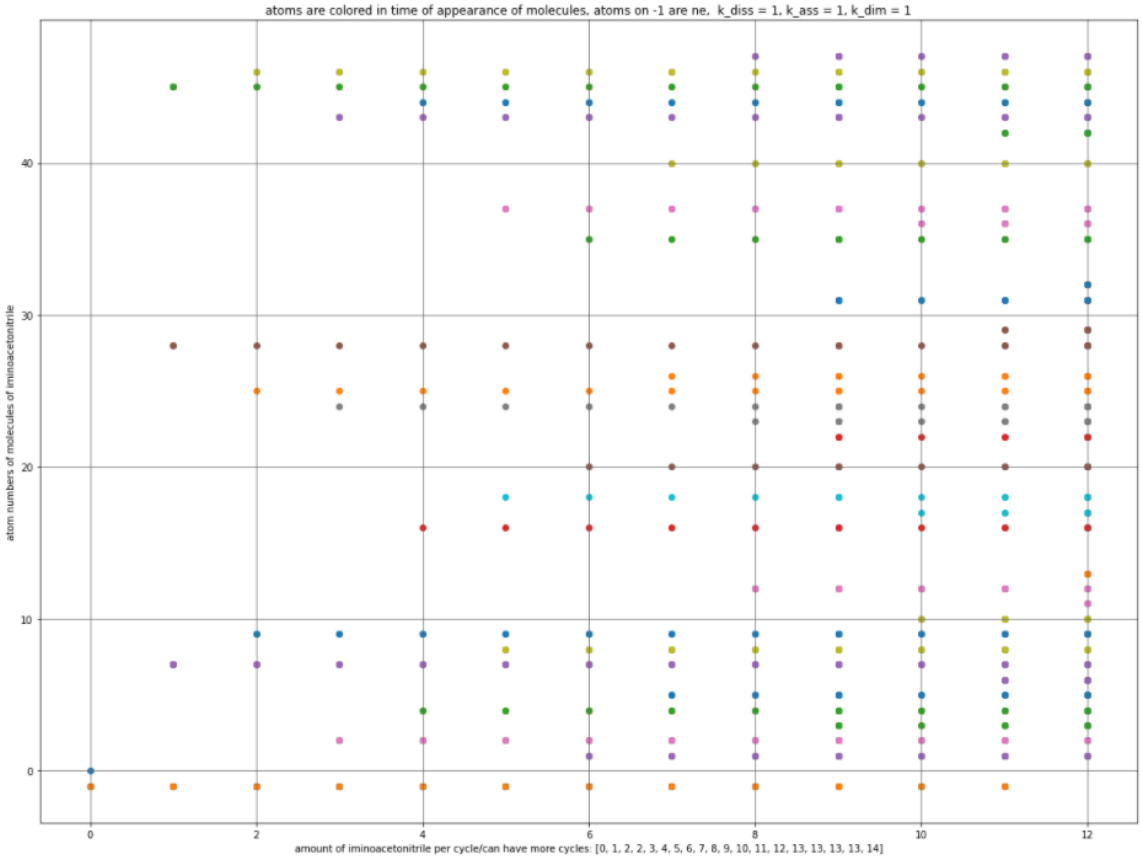

atoms are colored in time of appearance of molecules, atoms on -1 are ne, k_diss = 1, k_ass = 1, k_dim = 1
atom numbers of molecules of iminoacetonitrile
amount of iminoacetonitrile per cycle/can have more cycles: [0, 1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 13, 13, 14]
0
10
20
30
40
0
2
4
6
8
10
12

Figure 16

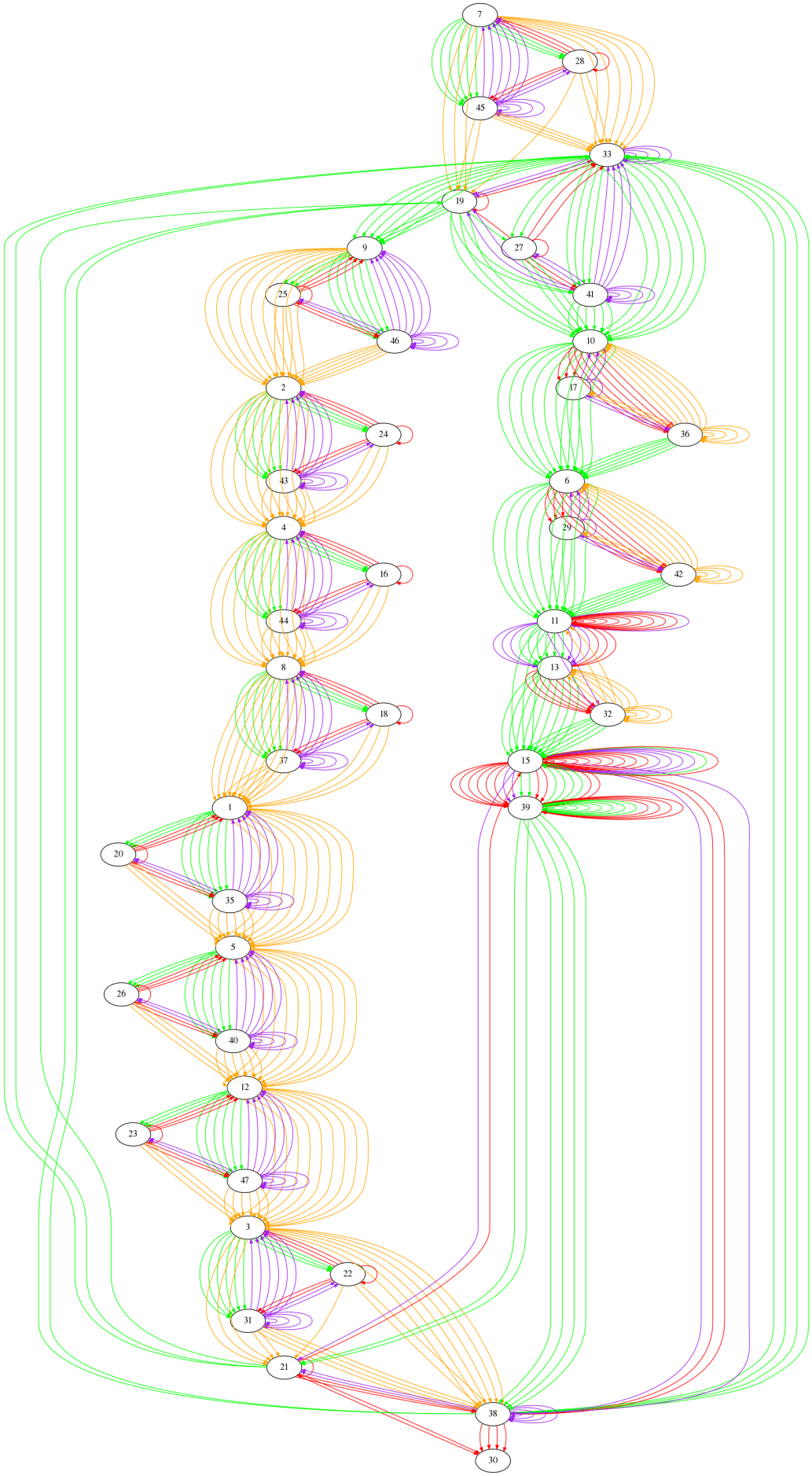


Figure 17: Iminoacetonitrile: orange, formonitrile: green, cyanide ion: purple, hydron: red

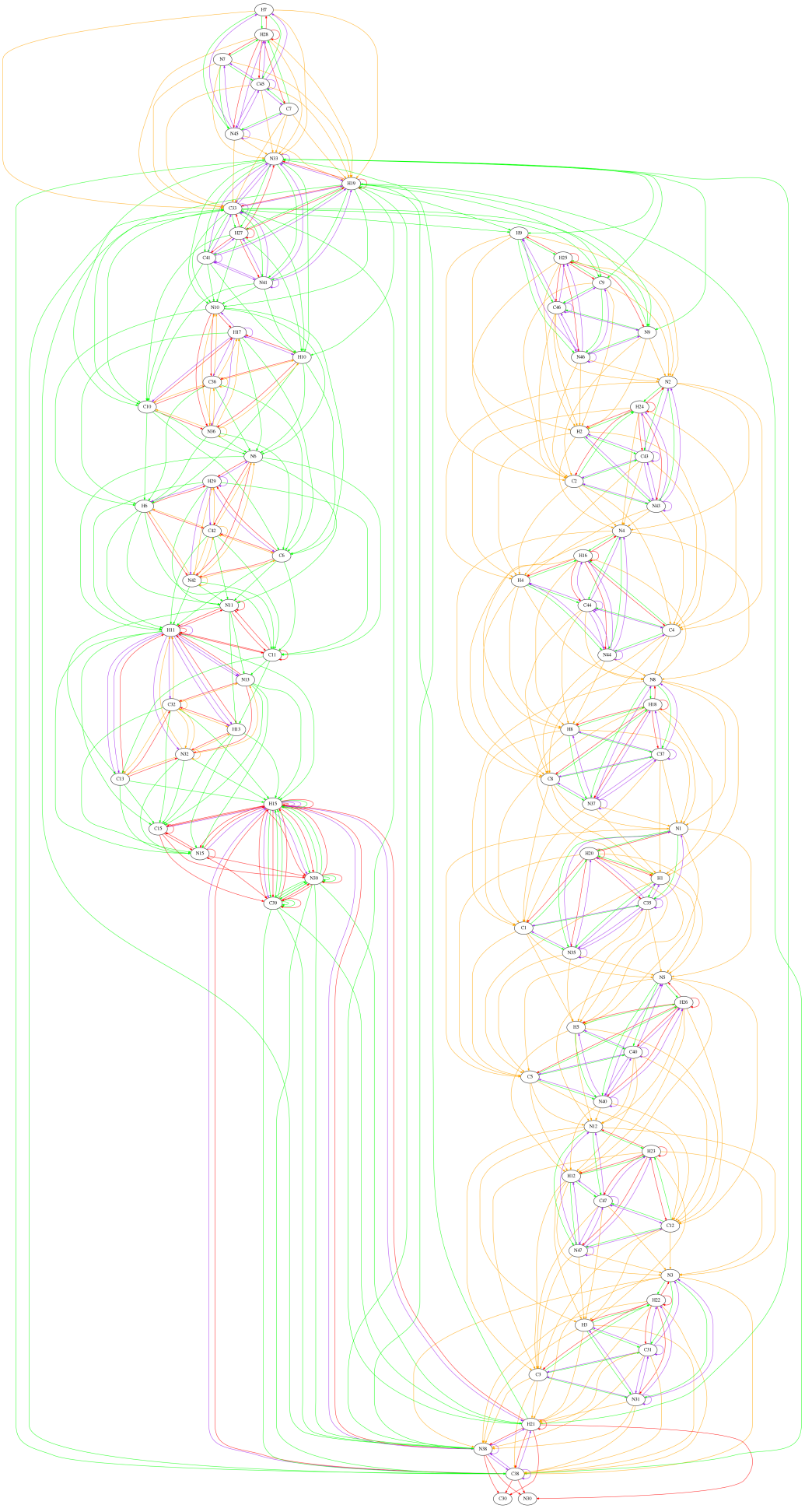


Figure 18

Bibliography

[1] C. Flamm. (2020) Softwareentwicklungsprojekt p.11. [Online]. Available: https://moodle.univie.ac.at/pluginfile.php/13413611/mod_resource/content/1/project2.pdf

[2] G. Landrum. (2020) Rdkit: Open-source cheminformatics software. [Online]. Available: https://www.rdkit.org/

[3] C. Flamm, "lecture." [Online]. Available: https://ufind.univie.ac.at/de/person.html?id=17324

[4] "fig1 fig2 drawn with tikzpicture."

[5] D. T. GILLESPIE. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. [Online]. Available: https://www.math.kit.edu/ianm3/lehre/semjahnke2011w/media/gillespie_general_method.pdf

[6] gillespie introduction. [Online]. Available: https://www.chemeurope.com/en/encyclopedia/Gillespie_algorithm.html

[7] sourceforge. [Online]. Available: https://sourceforge.net/projects/rdkit/

[8] powerpoint. [Online]. Available: https://de.slideshare.net/GregLandrum1/rd-kitopen-sourceintheenterprise

[9] Rdkit history and status. [Online]. Available: https://www.rdkit.org/UGM/2012/Landrum_RDKit_UGM.History%20and%20Status.Final.pptx.pdf

[10] Smiles - a simplified chemical language. [Online]. Available: https://www.daylight.com/dayhtml/doc/theory/theory.smiles.html

[11] taken from wikipedia - graph-based definition. [Online]. Available: https://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system

[12] Smarts - a language for describing molecular patterns. [Online]. Available: https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html

[13] Smirks - a reaction transform language. [Online]. Available: https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html

[14] Rdkit experiments, tips, and tutorials. [Online]. Available: https://rdkit.blogspot.com/

[15] Getting started with the rdkit in python. [Online]. Available: https://www.rdkit.org/docs/GettingStartedInPython.html

[16] library. [Online]. Available: https://rdkit.org/docs/genindex.html

[17] reaction sample rea.py written by christoph flamm. [Online]. Available: https://krios.tbi.univie.ac.at/Teaching/softwarepraktikum/rearulesim/-/tree/master

[18] legends were made with lucid. [Online]. Available: https://lucid.app/lucidchart/

[19] mol2chemfig allows to convert chemical structures from mdl molfile format to chemfig. [Online]. Available: https://ctan.org/pkg/mol2chemfig?lang=de

[20] wikipedia uses mathvault as source. [Online]. Available: https://mathvault.ca/hub/higher-math/math-symbols/set-theory-symbols/

[21] compare pseudo code of gillespie algorithm /direct method. [Online]. Available: https://mathvault.ca/hub/higher-math/math-symbols/set-theory-symbols/