The Reaction System framework

A python implementation

Introduction

This is a brief explanation of what my approach was to implement in python the formal framework of the Reaction Systems.
This framework is much used for investigating processes carried out by biochemical reactions in living cells.

In order to understand the framework, we need to clarify some concepts about the theoretical formulation of the representation.

What is a reaction? - 1

- A biochemical reaction can take take place if all of its reactants are present in a given and none of its inhibitors is present.
- When a reaction takes place it creates its products.

For the sake of simplicity, to understand the functioning of a reaction system, we can think to the entities of a reaction (atoms, ions, molecules) as numbers.

So for example: the reaction ({1,2},{3},{1,3,4}) takes place when in the state there are all of its reactants {1,2} and no one of its inhibitors {3}; when it takes place it produces the new entities {1,3,4}.

What is a reaction? - 2

def. A reaction is a triplet $a = (R,I,P)$, where R, I, P are finite nonempty sets with $R \cap I = \varnothing$.

If S is a set such that $R,I,P \subseteq S$, than a is a reaction in S.

def. Let T be a finite set.

1) The reaction a is enabled by T : $en_a(T)$ if $R \subseteq T$ and $I \cap T = \varnothing$.

   The result of a on T is $res_a(T) = \begin{cases} Pa & \text{if} \quad en_a(T) \\ \varnothing & \text{otherwise} \end{cases}$

2) Let A be a finite set of reactions.

   The result of A on T : $res_A(T) = \cup_{a \in A} res_a(T)$

What is a reaction? - 3

Notes:

- The intuition behind *T* is that of a state of a biochemical system, i.e., a set of biochemical entities present in the current biochemical environment. Thus *a* is enabled by *T* if *T* separates *R* from *I*.
- The result of all individual reactions *A* on *T* is cumulative
- There is no conflict of resources, i.e., if reactions *a* and *b* need some common reactants to be enabled, we suppose to have them in a sufficient quantity to enabled both reactions.

What is a reaction system?

def. A reaction system, abbreviated *rs*, is an ordered pair $\mathcal{A} = (S,A)$ such that S is a finite set, and $A \subseteq rac(S)$.

note: $rac(S)$ is the set of all possible reactions in S, so A is a set of reactions.

- S is called *background set* of $\mathcal{A}$, and its elements are called *entities*. They represent molecular entities that may be present in the states of a biochemical system modeled by $\mathcal{A}$.
- A is called the *set of reactions* of $\mathcal{A}$.

Dynamic behavior of a *rs*

The dynamic behavior of a reaction system is formalized through the notion of an interactive process.

def. Let $\mathcal{A} = (S,A)$ be a *rs* and let $n \geq 0$ be an integer.
An n-step interactive process in $\mathcal{A}$ is a pair $\pi = (\gamma,\delta)$ of a finite sequences such that $\gamma = C_0,\ldots,C_n$ and $\delta = D_0,\ldots,D_n$, where $C_0,\ldots,C_n,D_0,\ldots,D_n \subseteq S$, $D_0 = \varnothing$, and $D_i = res_{\mathcal{A}}(D_{i-1} \cup C_{i-1})$ for all $i \in \{1,\ldots,n\}$.

- γ is called *context sequence* of π.
- δ is called *result sequence* of π.

Then $\mathcal{T} = W_0,\ldots,W_n$ defined by $W_i = C_i \cup D_i$ for all $i \in \{0,\ldots,n\}$ is the *state sequence* of π.

The *context sequence* of an interactive process

The *context sequence* formalizes the intuition that, in general, a *rs* is not a closed system, and so its behavior is influenced by its environment.

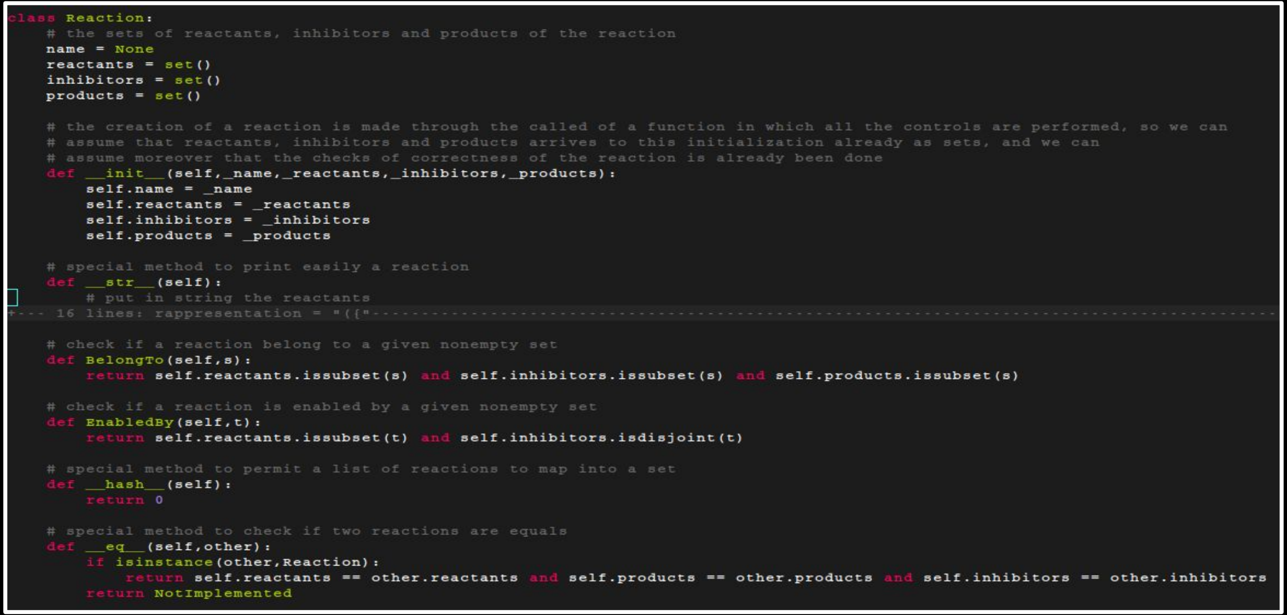
➡ If $C_i \subseteq D_i$ for all $i \in \{0,...,n\}$ than we say that π (and $\mathcal{T}$) are *context-independent*.

And this is due to the fact that, if for every *i*-th step we have not supplementary entities to those that are already present in the result set D_i, than it means that all the states W_i are equal to the set D_i, confirming that the *state sequence* do not depend from the *context sequence*.

In a *context-independent* interactive process the *state sequence* depends only on the initial state $W_o = C_0$ and its length n+1.

The python approach - Reaction class - 1

As first thing, I’ve implemented the class ‘Reaction’ to model the concept of reaction as explained before.



```
class Reaction:
    # the sets of reactants, inhibitors and products of the reaction
    name = None
    reactants = set()
    inhibitors = set()
    products = set()

    # the creation of a reaction is made through the called of a function in which all the controls are performed, so we can
    # assume that reactants, inhibitors and products arrives to this initialization already as sets, and we can
    # assume moreover that the checks of correctness of the reaction is already been done
    def __init__(self,_name,_reactants,_inhibitors,_products):
        self.name = _name
        self.reactants = _reactants
        self.inhibitors = _inhibitors
        self.products = _products

    # special method to print easily a reaction
    def __str__(self):
        # put in string the reactants
+--- 16 lines: rappresentation = "({"------------------------------------------------------------------------------------

    # check if a reaction belong to a given nonempty set
    def BelongTo(self,s):
        return self.reactants.issubset(s) and self.inhibitors.issubset(s) and self.products.issubset(s)

    # check if a reaction is enabled by a given nonempty set
    def EnabledBy(self,t):
        return self.reactants.issubset(t) and self.inhibitors.isdisjoint(t)

    # special method to permit a list of reactions to map into a set
    def __hash__(self):
        return 0

    # special method to check if two reactions are equals
    def __eq__(self,other):
        if isinstance(other,Reaction):
            return self.reactants == other.reactants and self.products == other.products and self.inhibitors == other.inhibitors
        return NotImplemented
```

The python approach - Reaction class - 2

A Reaction object has the three sets of reactants, inhibitors and products.

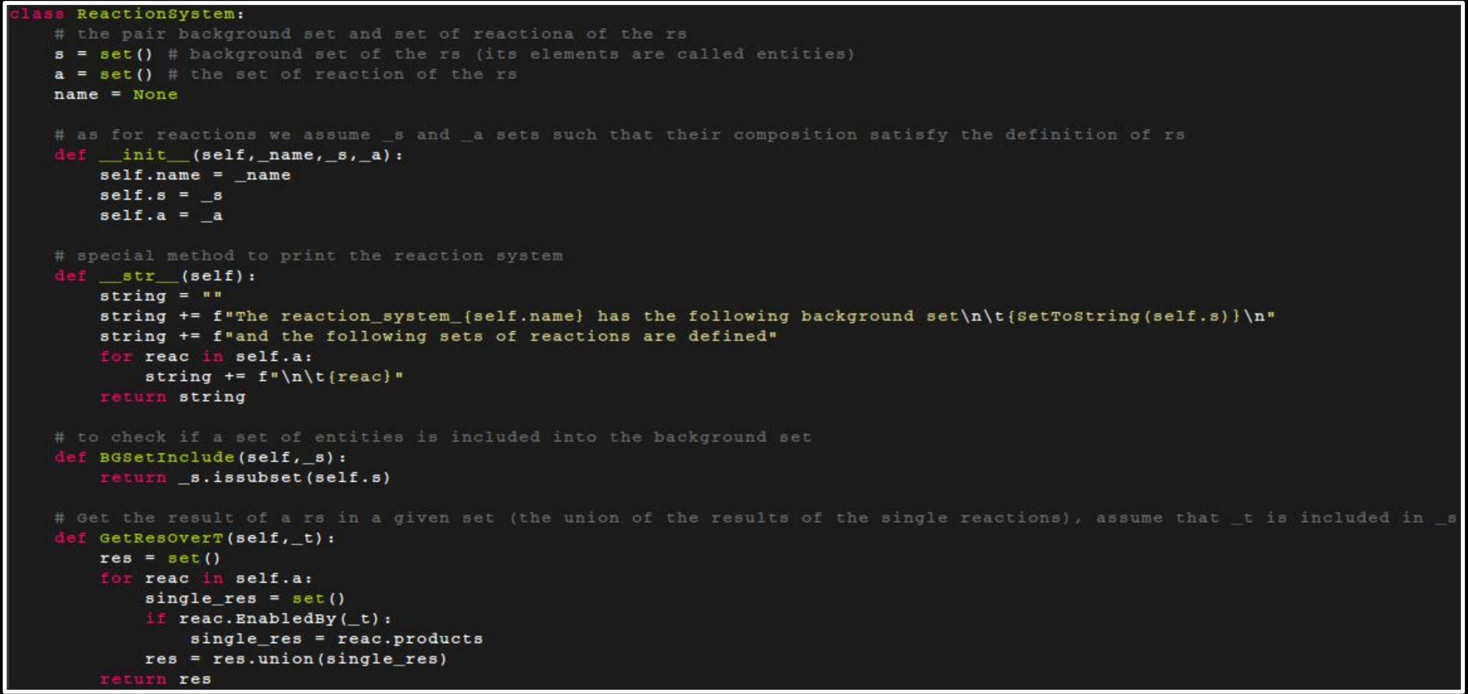
It has a name to identificate and distinguish it among the other reactions.

With the method ‘BelongTo’ is possible to check if the reaction belong to a given set, i.e., all the reaction entities are included into that set.

Finally, the method ‘EnabledBy’ return True if the reaction is enabled by a given set, i.e., if $R \subseteq T$ and $I \cap T = \varnothing$.

The python approach - ReactionSystem class - 1

Once I have a Reaction object I can construct a Reaction System.



```python
class ReactionSystem:
    # the pair background set and set of reactiona of the rs
    s = set() # background set of the rs (its elements are called entities)
    a = set() # the set of reaction of the rs
    name = None

    # as for reactions we assume _s and _a sets such that their composition satisfy the definition of rs
    def __init__(self,_name,_s,_a):
        self.name = _name
        self.s = _s
        self.a = _a

    # special method to print the reaction system
    def __str__(self):
        string = ""
        string += f"The reaction_system_{self.name} has the following background set\n\t{SetToString(self.s)}\n"
        string += f"and the following sets of reactions are defined"
        for reac in self.a:
            string += f"\n\t{reac}"
        return string

    # to check if a set of entities is included into the background set
    def BGSetInclude(self,_s):
        return _s.issubset(self.s)

    # Get the result of a rs in a given set (the union of the results of the single reactions), assume that _t is included in _s
    def GetResoverT(self,_t):
        res = set()
        for reac in self.a:
            single_res = set()
            if reac.EnabledBy(_t):
                single_res = reac.products
            res = res.union(single_res)
        return res
```

The python approach - ReactionSystem class - 2

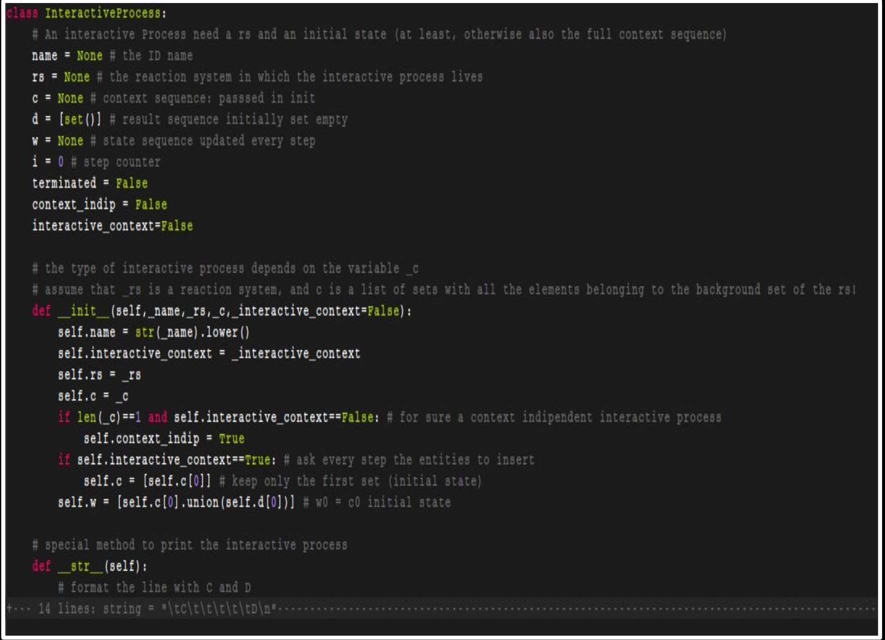
Here also there is a name identificator and the two set that compose the reaction system: *background set* and the *set of reactions*.

Essentially, we have two main methods.

- ‘BGSetInclude’ is used in the object creation phase to check if a set to add to the set of reactions belong to the background set, moreover, it is used before to call the second ‘GetResOverT’ to be sure that the provided set over which we want the result belong to the background set.
- ‘GetResOverT’ return the set of the union of the single result of all the reactions over the set T.

The python approach - InteractiveProcess class - 1

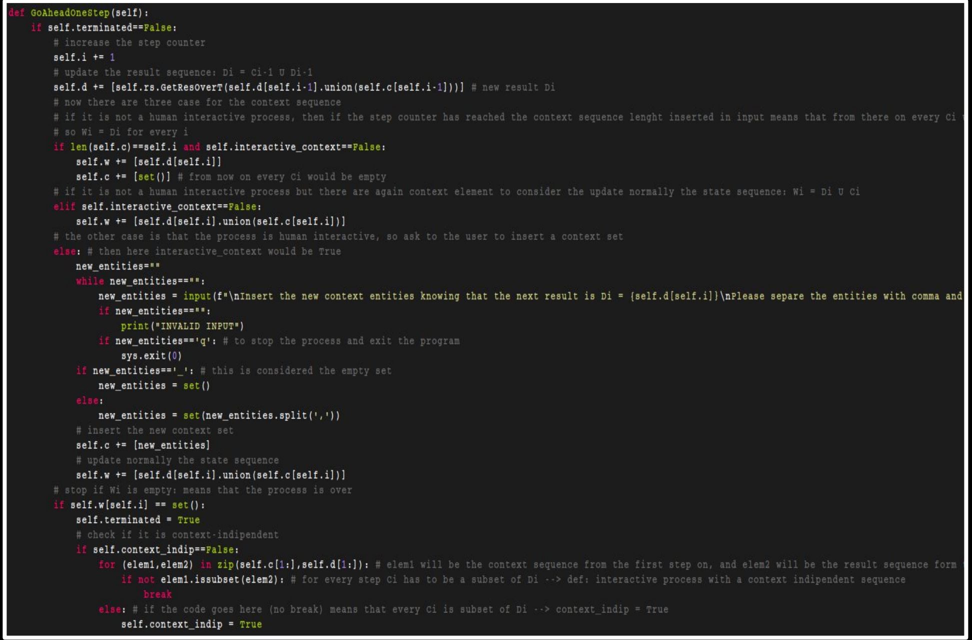
The last main class is the ‘InteractiveProcess’ class with the purpose to model the functioning of an interactive process in a certain reaction system.



```python
class InteractiveProcess:
    # An interactive Process need a rs and an initial state (at least, otherwise also the full context sequence)
    name = None # the ID name
    rs = None # the reaction system in which the interactive process lives
    c = None # context sequence: passed in init
    d = [set()] # result sequence initially set empty
    w = None # state sequence updated every step
    i = 0 # step counter
    terminated = False
    context_indip = False
    interactive_context=False

    # the type of interactive process depends on the variable _c
    # assume that _rs is a reaction system, and c is a list of sets with all the elements belonging to the background set of the rs!
    def __init__(self,_name,_rs,_c,_interactive_context=False):
        self.name = str(_name).lower()
        self.interactive_context = _interactive_context
        self.rs = _rs
        self.c = _c
        if len(_c)==1 and self.interactive_context==False: # for sure a context indipendent interactive process
            self.context_indip = True
        if self.interactive_context==True: # ask every step the entities to insert
            self.c = [self.c[0]] # keep only the first set (initial state)
        self.w = [self.c[0].union(self.d[0])] # w0 = c0 initial state

    # special method to print the interactive process
    def __str__(self):
        # format the line with C and D
+··· 14 lines: string = "\tC\t\t\t\t\tD\n"·······································································
```



```python
def GoAheadOneStep(self):
    if self.terminated==False:
        # increase the step counter
        self.i += 1
        # update the result sequence: Di = Ci-1 U Di-1
        self.d += [self.rs.GetResOverT(self.d[self.i-1].union(self.c[self.i-1]))] # new result Di
        # now there are three case for the context sequence
        # if it is not a human interactive process, then if the step counter has reached the context sequence lenght inserted in input means that from there on every Ci
        # so Wi = Di for every i
        if len(self.c)==self.i and self.interactive_context==False:
            self.w += [self.d[self.i]]
            self.c += [set()] # from now on every Ci would be empty
        # if it is not a human interactive process but there are again context element to consider the update normally the state sequence: Wi = Di U Ci
        elif self.interactive_context==False:
            self.w += [self.d[self.i].union(self.c[self.i])]
        # the other case is that the process is human interactive, so ask to the user to insert a context set
        else: # then here interactive_context would be True
            new_entities=""
            while new_entities=="":
                new_entities = input(f"\nInsert the new context entities knowing that the next result is Di = {self.d[self.i]}\nPlease separe the entities with comma and
                if new_entities=="":
                    print("INVALID INPUT")
                if new_entities=='q': # to stop the process and exit the program
                    sys.exit(0)
            if new_entities=='_': # this is considered the empty set
                new_entities = set()
            else:
                new_entities = set(new_entities.split(','))
            # insert the new context set
            self.c += [new_entities]
            # update normally the state sequence
            self.w += [self.d[self.i].union(self.c[self.i])]
        # stop if Wi is empty: means that the process is over
        if self.w[self.i] == set():
            self.terminated = True
            # check if it is context-indipendent
            if self.context_indip==False:
                for (elem1,elem2) in zip(self.c[1:],self.d[1:]): # elem1 will be the context sequence from the first step on, and elem2 will be the result sequence form
                    if not elem1.issubset(elem2): # for every step Ci has to be a subset of Di --> def: interactive process with a context indipendent sequence
                        break
                else: # if the code goes here (no break) means that every Ci is subset of Di --> context_indip = True
                    self.context_indip = True
```

The python approach - InteractiveProcess class - 2

The attributes are those necessary to define an interactive process.

- ‘name’ : as an identifier.
- ‘rs’ : the reaction system to consider.
- ‘c’, ‘d’, and ‘w’ are the *context*, *result* and *state* sequences respectively.
- i : the step counter of the process

There are the additional attributes to take into account the termination of an interactive process, if either the state sequence is context-independent or not, and if the user has requested to make an interactive process with his human interaction, i.e., it is requested to him the context set at each step.

The python approach - InteractiveProcess class - 3

In the ‘__init__’ method are set the attributes, it is set the initial context sequence (or just the initial context set eventually) checking if eventually is possible to conclude that the process has a context independent state sequence. Further, it is checked if the process has a human interactive progress.

The ‘GoAheadOneStep’ method is responsible update properly all the sequences. It makes use of the ‘GetResOverT’ method of the ReactionSystem class to obtain the new result set D_i , then if the process is human interactive it ask for the new context set. At the end, if the obtained state set W_i is empty, we have have the termination of the process, and we can check for the context-independent property.

The python code - Demo program

To show the functioning of the framework I’ve implemented a small main program in which to the user is asked to provide a file with the definition of the reaction system.

Once the reaction system is given, it is asked to him to choose between a human interactive interactive process and a simple interactive process on which the entire context sequence is specified in advance.

Please, see the README.txt file for much detail on how to test the code.