Equipe Lifeware, INRIA - Juin 2022 – Tuteur : François Fages.

COMPILING MATHEMATICAL FUNCTIONS IN BIOCHEMICAL REACTION NETWORKS

Rapport de stage INF591 – Julien Bienvenu

PREAMBULE

Dans ce rapport de stage, je vais vous présenter l'état d'avancée actuel des travaux réalisés jusqu'à présent au cours de mon stage (débuté le 1er avril et s'achevant fin août), en tenant des propos les plus synthétiques possibles et sans nécessairement détailler toutes les étapes de conception et de réflexion eues pendant cette période afin de ne conserver que l'essentiel dans cet écrit.

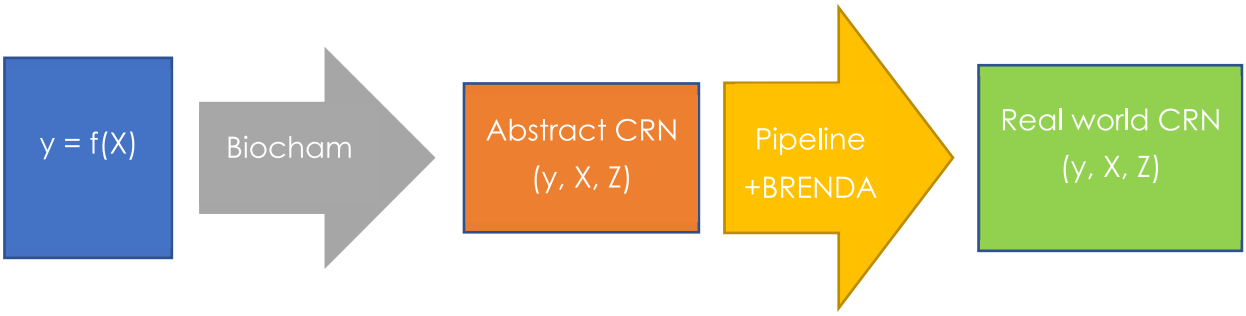
Mon sujet de stage s'inscrit dans le domaine de la bio-informatique, et plus particulièrement de l'informatique analogique cellulaire, il a été proposé par François Fages au sein de l'équipe projet Lifeware de l'INRIA Saclay.

SUJET

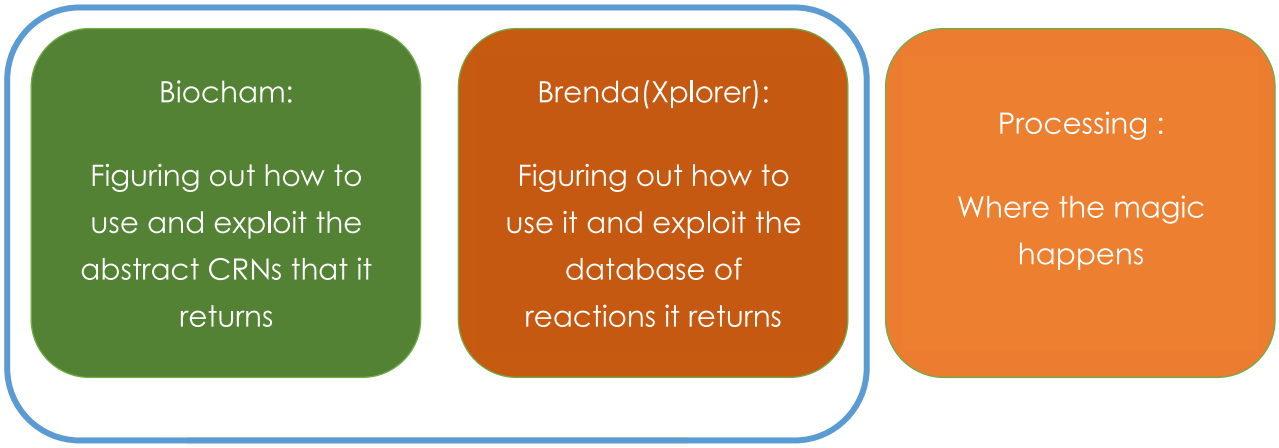
Les Réseaux de Réactions Chimiques (Chemical Reaction Networks, CRN) sont un formalisme standard utilisé en chimie et biologie pour décrire, analyser et designer des réseaux d'interactions moléculaires complexes. Dans le cadre de la biologie des systèmes, les CRN sont un outil central pour analyser les fonctions haut-niveau des cellules en terme d'interactions moléculaires bas-niveau. Dans le cadre de la biologie synthétique, les CRN constituent un langage de programmation cible pour implémenter en chimie de nouvelles fonctions, dans des cellules vivantes ou des dispositifs artificiels.

En se basant sur un résultat précédemment obtenu de Turing-complétude sur les CRN finis interprétés par des ODE, l'équipe Lifeware a implémenté dans son software BIOCHAM (Biochemical Abstract Machine) une pipeline originale permettant de computer n'importe quelle fonction mathématique élémentaire (du temps ou d'une concentration d'espèce chimiques en input) en un CRN abstrait qui implémente cette fonction. Le résultat obtenu est donné selon la concentration d'une espèce chimique output, soit par sa trace pour une fonction du temps, soit par sa stabilisation pour une fonction input/output.

Il existe actuellement seulement deux systèmes performant des tâches de synthétisation de CRN. Les deux fonctionnent dans la perspective de concevoir des concrétisation de CRN avec des systèmes de déplacement de brins d'ADN, là où l'équipe Lifeware vise des réactions enzymatiques avec des protéines aux cinétiques plus rapides de plusieurs ordres de grandeur, du même type que des CRN naturels, mais également limitées en variétés et cinétiques, avec la présence de réactions adverses non négligeables.



L'objectif de mon stage est de réaliser une pipeline en backend de BIOCHAM, permettant de concrétiser un CRN abstrait obtenu par ce dernier, compte tenu d'une base de donnée de réactions enzymatiques. Il n'existe en l'état pas de programme réalisant cette tâche.



Pre-Processing Bricks

ORGANISATION ET FONCTIONNEMENT AU SEIN DE L'EQUIPE

J'ai tenu dans un premier temps à résoudre le problème donné uniquement d'un point de vue conceptuel, sans programmer immédiatement, ce qui m'a permis de développer les outils et objets abstraits nécessaires rapidement tout en garantissant des résultats sensés et performants pour mon éventuel code. Cela ne m'a pas évité des étapes de débogage une fois le code réalisé mais m'a permis de me concentrer sur le sens de ce que je faisais.

Mon sujet de stage m'a donné une grande autonomie au sein de l'équipe Lifeware, et m'a notamment permis de naviguer et d'échanger avec les différents membres de l'équipe pour résoudre des problèmes de natures variés, qu'ils soient mathématiques, biochimiques ou plutôt informatiques. J'ai également beaucoup communiqué avec l'équipe Sys2Diag du CNRS qui a développé un outil qui nous a servi de référence pour mon projet, Brenda Xplorer.

Avec mon tuteur, et d'autres membres du laboratoire, nous avons pour ambition de rédiger plusieurs articles pendant mon stage. Le premier a été soumis a la conférence BioCAS 2022 (Biomedical Circuits and Systems Conference) et se trouve en annexe. Le second portera a priori d'avantage sur des aspects cinétiques.

Au cours de cette première partie de stage j'ai pu tenir plusieurs interventions au sein de séminaires internes à l'équipe, qui m'ont permis de présenter mes avancées, mes idées et d'échanger avec tous les membres du laboratoires sur les questions ou problèmes rencontrés.

VOCABULAIRE ET ACRONYMES

CRN : Chemical Reaction Network (Réseau de Réaction Chimique). On en distinguera deux types, les abstraits (aCRN) et les concrets (cCRN).

Métabolite : Substance organique formée au cours du métabolisme ou qui y participe. Pour cet écrit, métabolite désignera donc plus particulièrement toute espèce chimique impliquée (en tant que réactif, produit ou inhibiteur) dans une réaction enzymatique.

Enzyme : Une enzyme est une protéine dotée de propriétés catalytiques. Une enzyme est ainsi impliquée dans une ou plusieurs réactions qu'elle catalyse.

aCRN : Peut-être décrit comme un graphe bipartite orienté, possédant deux types de sommets : des métabolites et des enzymes abstraites (que l'on désignera toutes deux avec des lettres majuscules). Ce CRN abstrait peut être accompagné de contraintes cinétiques pour les différentes réactions qui y sont impliquées. Chaque métabolite (et enzyme) de ce graphe représente une espèce abstraite distincte.

cCRN : De la même façon peut-être décrit comme un graphe bipartite orienté. Les espèces chimiques contenues sont réelles, et deux sommets distincts se doivent d'être deux espèces chimiques distinctes.

BIOCHAM (également Biocham ou BioChAM) : Biochemical Abstract Machine, est un environnement de modélisation pour la biologie des systèmes et la biologie synthétique développé par l'équipe Lifeware, et basé sur la théorie computationnelle des CRN.

SEPI et SISO : subgraph epimorphism et subgraph isomorphism (ie epimorphisme de sous-graphe et isomorphisme de sous-graphe).

TRAVAUX REALISES

BRENDA

BRENDA est l'une des principales collections de données sur les enzymes disponible à la communauté scientifique. Elle est accessible via son site internet (www.brenda-enzymes.org). Le développement de cette base a débuté en 1987 au German National Research Centre for Biotechnology in Braunschweig (GBF), et continué par l'Université de Cologne. Actuellement elle est entretenue et hébergée à la Technical University of Braunschweig, Institute of Biochemistry and Bioinformatics.

Les enzymes y sont classifiés selon la list d'enzymes de l'Enzyme Commission, les données qui y sont liées sont extraites de la littérature et évaluées critiquement par des scientifiques compétents. La nomenclature originellement utilisée par les auteurs pour designer l'enzyme et ses sous-unités est conservée, tout comme celle liée aux organismes et souches (même s'il y a reclassification en cours de route).

Les données que j'ai utilisées pour mes démarches de concrétisation proviennent d'une extraction de BRENDA réalisée grâce à l'outil BrendaXplorer (https://brendaxplorer.lisn.upsaclay.fr/). Cette base contient 304 848 réactions, parmi lesquelles on dénombre 116 215 noms de métabolites distincts et 6502 EC distincts. Une des premières difficultés rencontrées face cette base est que différents noms de

métabolites peuvent faire référence au même composé chimique (par exemple acétone et 2-propanone sont la même espèce)

Ce constat soulève des inquiétudes naturelles quant à la suite de la démarche, en effet il est indispensable qu'une espèce chimique n'apparaisse pas plusieurs fois dans un cCRN pour garantir son fonctionnement. Une API développée par BRENDA permet de résoudre partiellement cet écueil, renvoyant un identifiant unique par molécule, pour un nom donné (reconnu). Par exemple « acétone » et « 2-propanone » obtiennent tous deux l'ID "286". Il doit cependant être noté que ce système n'est pas chimiquement parfait, en effet les espèces "H2O", "H3O+", "HO-" et "1alpha-hydroxy-vitamin D3" sont toutes désignées par l'ID "1". Parmi les 116 215 noms de métabolites, l'API n'en reconnait à l'état actuel pas 35 755 d'entre eux et distingue 53 162 molécules distinctes parmi les métabolites restants. Des 304 848 réactions initiales, il y a 129 482 réactions impliquant au moins un métabolite non reconnu, elles sont exclues de notre set de données et on conserve un total de 175 366 réactions intégralement reconnues.

Pipeline principale

Nous avons représenté les différents objets (enzymes, métabolites, réactions, bibliothèque de réactions..) avec des classes bien choisies et assez modulaires pour pouvoir être réexploités dans le futur. Je ne vais pas toutes les détailler, mais évoquer les aspects les plus saillants et nécessaires pour le détail de la pipeline.

Un CRN peut-être représenté par un graphe bipartite orienté avec deux types de sommets, les métabolites et les enzymes. Ces sommets sont étiquetés avec des lettres ou des chaines de caractères, et le graphe peut-être implémenté en utilisant une matrice M comme suit : $M_{(i,j)} = 'in'$ si le i-ème métabolite est réactif pour la j-ème enzyme, et il en va de même pour $'out'$ pour les produits et $'inh'$ pour les inhibiteurs. Cette matrice peut être combinée à une liste détaillant les différentes descripteur cinétiques associées à chaque enzyme. Lorsque l'on se réfère à des enzymes ou métabolites concrets, des lettres majuscules sont utilisées, des minuscules pour les concrètes. Pour un enzyme E, $E.in$ (et $E.out$) représentent l'ensembles des réactifs (et produits) de E. Un aCRN utilise des noms formels pour désigner ses espèces chimiques.

Lorsque l'on compare une réaction abstraite (par exemple comme partie d'aCRN) à une réaction concrète, il est nécessaire d'être capable de dire si ces deux réactions se correspondent (ou « match »), ie si la seconde peut-être utilisée pour concrétiser la première. L'un des critères évident est d'ordre topologique : assez d'entrées et de sorties. Cela donne lieu à deux types de matching, le "hard-

matching » qui requiert que les deux réactions aient exactement autant d'entrées et de sorties l'une que l'autre, et à un « soft-matching » qui requiert que la réaction concrète ait au moins autant d'entrées (et de sorties) que la réaction abstraite. Le « hard » donne lieu à une recherche de type SISO là où le « soft » est plus souple. En plus de ces contraintes topologiques à respecter, on pourrait à terme ajouter des contraintes cinétiques pour ne considérer que les réactions ayant un profil cinétique compatible.

La pipeline principale que j'ai développée possède deux étapes principales. La première est un pré-parcours astucieux du aCRN à concrétiser et la deuxième use de ce parcours pour chercher une concrétisation.

L'objectif du pré-parcours est de choisir un ordre selon lequel concrétiser (ou « remplir ») tous les enzymes et métabolites, linéariser la seconde étape de la pipeline, et mettre en exergue le plus tôt possible des incompatibilités potentielles (ce qui réduit grandement la durée de la seconde étape étant donné qu'elle consiste en exploration arborescente récursive). Le pré-parcours consiste en un coloriage de la matrice du graphe, rouge pour les métabolites non remplis et vert pour ceux déjà remplis. Pour entamer le processus, les métabolites prédéfinis sont verdis et les autres rougis. Ensuite l'enzyme non rempli avec les plus de vert dans sa colonne est recherché (en cas d'égalités, celui avec le moins de rouge est préféré). Plus de vert signifie plus de métabolites déjà remplis et donc moins de chance de trouver un enzyme correspondant, et moins de rouge implique qu'il y aura moins de nouvelles branches dans l'arbre récursif en devenir. On enregistre l'enzyme rempli à cette étape, tout comme les métabolites déjà remplis dans sa colonne, ainsi que ceux qui vont l'être (séparément). Tous les métabolites de cet enzyme sont verdis (pour toutes les colonnes où ils sont impliqués), et on réitère le processus pour trouver un nouvel enzyme à remplir. Le processus s'arrête quand il n'y a plus d'enzyme non-rempli possédant du vert dans sa colonne. Le graphe complet aura été rempli ssi il est connexe. La chaine des enregistrements constitue notre pré-parcours et sera le chemin suivi lors de la concrétisation du aCRN par le procédé suivant.

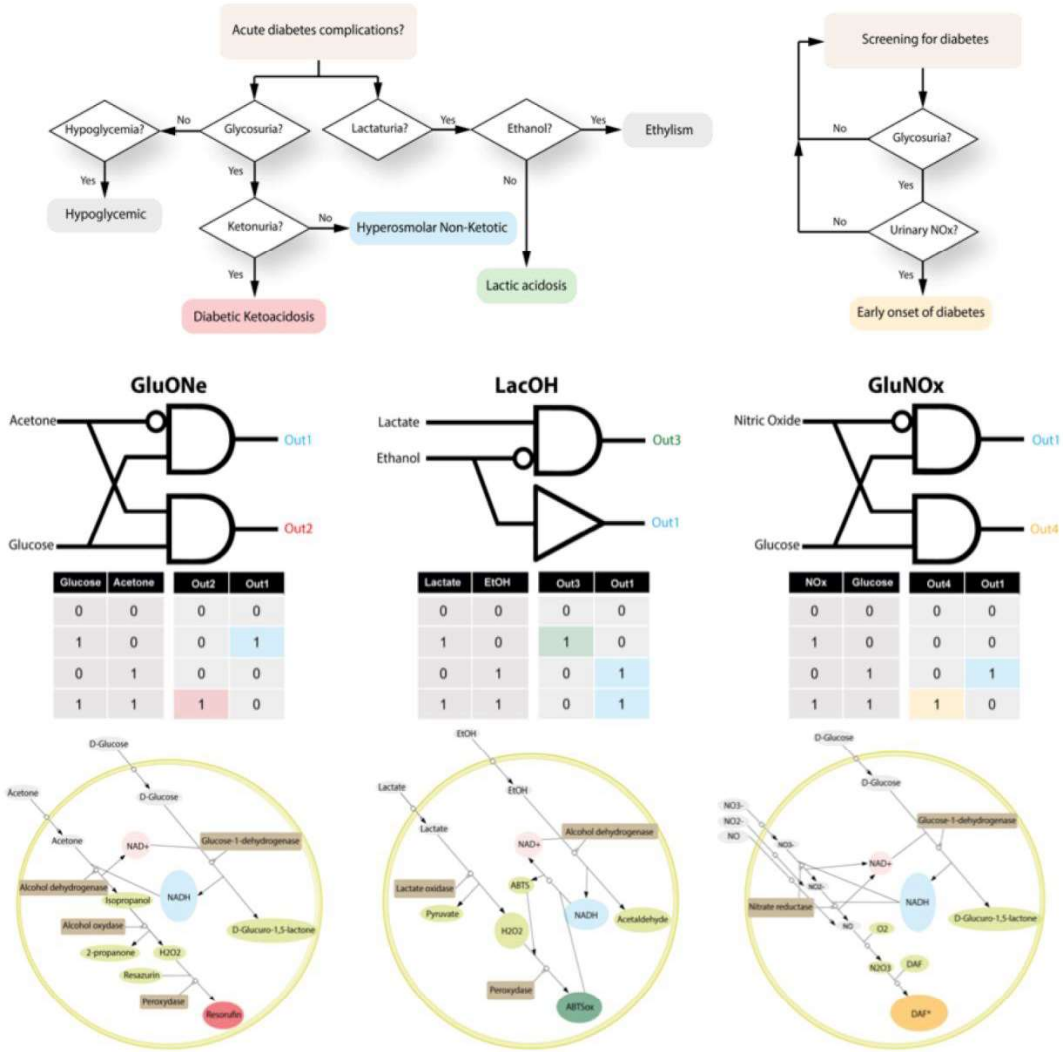
Ce second procédé est essentiellement récursif et je vais le détailler pour le cas « soft-match » (les deux autres étant soit similaires soit plus simples). Dans ce procédé, on va gérer deux blacklists, deux dictionnaires et une greylist. Les deux blacklists vont chacune servir à nous éviter de réutiliser des enzymes et métabolites dans le graphe, une fois qu'ils ont servi à remplir une espèce abstraite. On les notera B_E et B_M respectivement. La greylist, notée G_M compilera les espèce « environnantes », ie les composés impliqués dans une réaction mais n'apparaissant pas dans le CRN. Ces métabolites environnants peuvent être réutilisés en tant que tels dans des réactions

ultérieures mais pas pour remplir un métabolite abstrait. Les dictionnaires décriront la concrétisation, D_M pour les métabolites et D_E pour les enzymes. Avant de lancer la récursion, on compile une liste de sets de candidats matchant pour chaque enzyme de l'aCRN. Pour l'étape n on procède comme suit :

- Si n est plus grand que le nombre d'enzymes, (D_E, D_M, G_M) constitue une solution, que l'on enregistre.
- Sinon, une description E, A, X (avec E enzyme à remplir, A l'ensemble des métabolites à remplir et X l'ensemble des métabolites déjà remplis par x) de l'étape n est considérée, ainsi que l'ensemble K des candidats pour E
    - On rejette tout candidat $c \in K$ si $c \in B_E$ ou $x \not\subset c.in \cup c.out$
    - Pour tous les candidats c restants de K, on suit ces étapes :
        - Soit $e_{in} = c.in\ (x.in \cup G_M)$ et $e_{out} = c.out\ (x.out \cup G_M)$.
        - Si $|e_{in}| < |A.in|$ ou $|e_{out}| < |A.out|$, ce candidat est rejeté.
        - Sinon, $D_E[E] = c$ et l'on considère tous les arrangements ai de e_{in} à $|A.in|$ éléments et les arrangements ao de e_{out} à $|A.out|$ éléments
        - Pour tous les couples (ai, ao), $D_M[A.in] = ai$, $D_M[A.out] = ao$, et le processus avance à l'étape $n+1$ avec B_M remplacé par $B_M \cup ai \cup ao$, B_E par $B_E \cup c$, et G_M par $G_M \cup (e_{in} \setminus ai) \cup (e_{out} \setminus ao)$.

RESULTATS OBTENUS

Les enjeux de la concrétisation sont multiples, notamment dans le domaine du test médical. En effet un diagnostic peut se décrire parfois très aisément à l'aide de circuits logiques. Désigner une vésicule contenant les « bonnes » espèces chimiques pour réaliser cette fonction logique revient intégralement à concrétiser l'aCRN associé au circuit logique. Une fois conçues ces vésicules réaliseraient de véritables diagnostics fiables et peu couteux. Voici la série d'exemples sur lesquels se sont basés mes principaux test.



Glucose	Acetone	Out2	Out1
0	0	0	0
1	0	0	1
0	1	0	0
1	1	1	0

Lactate	EtOH	Out3	Out1
0	0	0	0
1	0	1	0
0	1	0	1
1	1	0	1

NOx	Glucose	Out4	Out1
0	0	0	0
1	0	0	0
0	1	0	1
1	1	1	0

Figure 1- Algorithme de diagnostic de différents types de comas (diabétique de deux types, éthylique, ou lactique), circuits logiques correspondants (testant la présence ou l'absence de glucose, acétone, éthanol, lactate), et graphes des CRNs correspondants (produisant en sortie une coloration ou fluorescence par accumulation de NADH, Résorufine, ABTS et DAF-2), implémentés dans des vésicules artificielles qui ont été testées dans des urines et du sang de patient (Courbet et al. 2018)

Ces cCRN ont été développés dans la thèse de pharmacologie d'A. Courbet, et être en mesure de les retrouver de façon automatisée constituait une étape clé du déroulement de mon stage. Nous détaillerons ici principalement les résultats obtenus sur le CRN de GluONe.

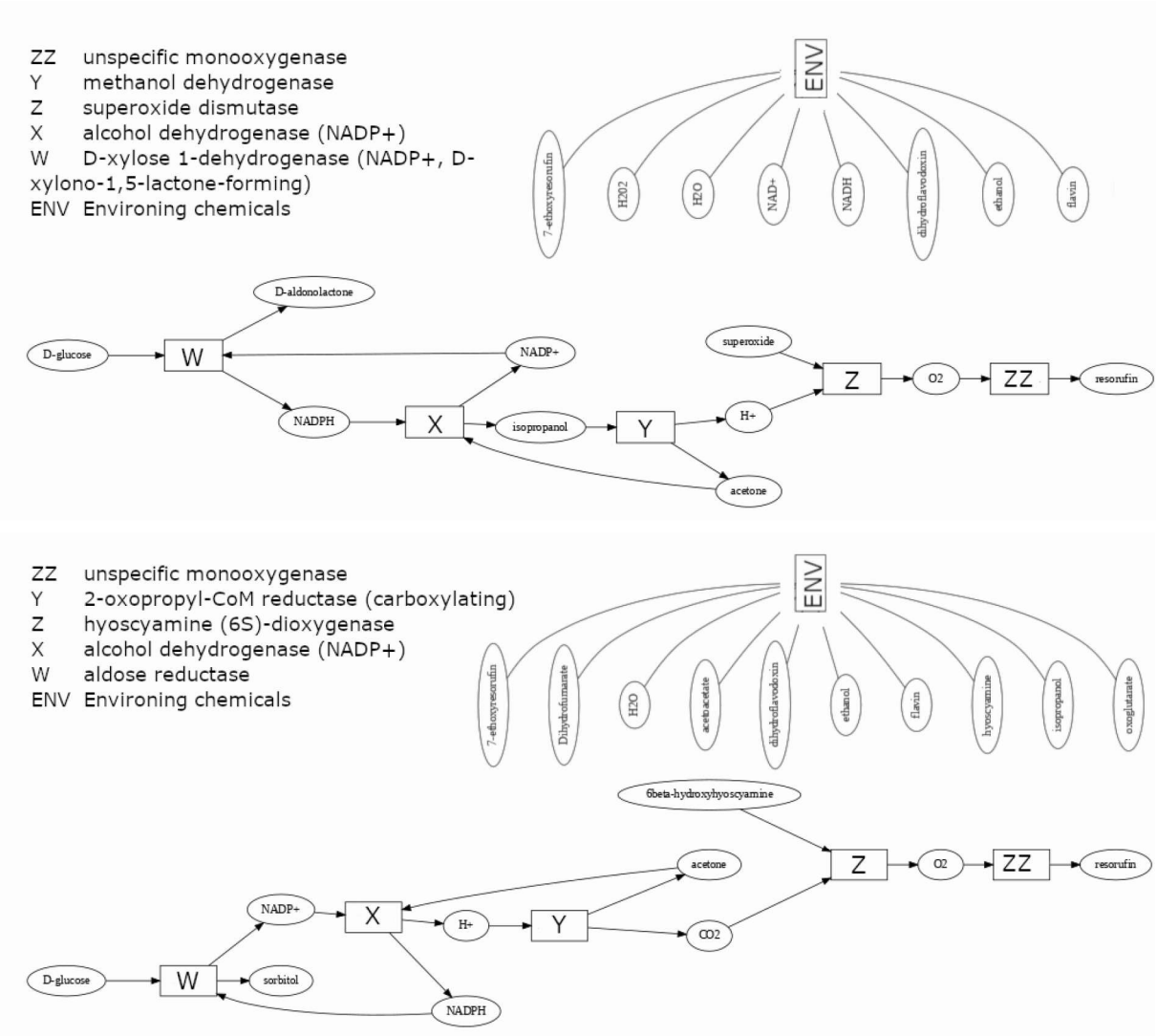


Figure 2- Deux exemples de concrétisation SISO du aCRN de GluONe*

En utilisant l'algorithme précédemment décrit, j'ai pu obtenir plusieurs milliers de solutions de concrétisation du circuit GluONe, incluant plus de six cent ensembles d'enzymes solutions distincts. Ces solutions ont été compilées en cherchant des cCRN SISO⋆ à celui développé lors de la thèse d'A. Courbet, mais en ajoutant un métabolite intermédiaire entre H_2O_2 et resorufin. En effet cette dernière réaction est absente de la base BRENDA. A cet exeption près nous avons été en mesure de retrouver exactement le cCRN d'Alexis. Deux autres solutions sont représentées ci-dessus.

SISO ET SEPI

Si mon code actuel, avec le « soft-matching » correspond à une version élargie de SISO (que j'ai tendance à noter SISO⋆) puisqu'il permet d'ajouter au cCRN un ensemble d'espèces environnantes, il n'en est pas moins très éloigné du concept de SEPI qui permet une grande souplesse structurelle. Biocham permet de réaliser de telles réductions à l'aide d'un solver SAT mais celui-ci n'est pas assez robuste pour en effectuer sur une base contenant plus d'une grosse centaine de réactions. Impossible donc de l'utiliser sur BRENDA. De plus il nous est récemment apparu que le support d'un tel SEPI n'étant pas nécessairement connexe, il ne peut a fortiori pas représenter un cCRN. Les délétions et fusions permises par le formalisme du SEPI sont trop permissives pour bien décrire une concrétisation. Nous allons donc devoir développer une notion, éventuellement nouvelle, plus restrictive pour fidèlement décrire le type de relation liant un aCRN et un cCRN. Cette question reste ouverte au moment où j'écris ces lignes.

Pour expérimenter les notions de SEPI et certaines fonctionnalités de Biocham j'ai développé un notebook à visée didactique ces dernières semaines, qui se trouve en annexe.

QUESTIONS CINETIQUES

Un problématique additionnelle sur laquelle je travaille dernièrement est celle des cinétiques qui résultent d'une concaténation de réactions. Je m'explique : lors d'une recherche de SEPI, il est courant d'effectuer des opérations de fusions de réactions pour en obtenir une plus petite. Mais comment caractériser sa cinétique compte tenu des cinétiques des réactions dont elle résulte ? Je me suis pour l'instant penché sur des chaines simples et des cinétiques de type loi d'action de masse.

Pour une chaine de réactions de la forme $\mathrm{a}_0 \xrightarrow{\mathrm{MA}(\mathrm{k}_0)} \mathrm{a}_1 \xrightarrow{\mathrm{MA}(\mathrm{k}_1)} \dots \xrightarrow{\mathrm{MA}(\mathrm{k}_{n-1})} \mathrm{a}_n \xrightarrow{\mathrm{MA}(\mathrm{k}_n)} \mathrm{b}$, on obtient via $D : u \mapsto \frac{du}{dt}$ et $Q_{(k_j)_{0\leq j\leq n}} = \frac{X}{\prod_{j=0}^{n} k_j} \prod_{j=1}^{n} (X + k_j)$ que $a_0 = Q_{(k_j)_{0\leq j\leq n}} \circ D(b)$.

Pour des chaines plus complexes, le problème reste ouvert. De plus si l'équation précédente lie bel et bien a_0 et b, il n'en reste pas moins peu clair quel type de cinétique on peut en déduire pour la réaction $\mathrm{a}_0 \xrightarrow{?} \mathrm{b}$. Ce sujet fera sans doute l'objet d'un second article si nous arrivons à faire progresser la question.

ANNEXES

1. Présentation du projet Lifeware

(source : https://www.inria.fr/fr/lifeware)

Le projet Lifeware vise à développer des méthodes formelles et des dispositifs expérimentaux pour comprendre la machinerie cellulaire et établir des paradigmes computationnels en biologie des systèmes. Lifeware se fonde sur la vision des cellules comme des machines, des réseaux de réactions biochimiques comme des programmes, et le développement de concepts et d'outils informatiques permettant de maîtriser la complexité des processus cellulaires. Le projet s'intéresse à des questions de recherche fondamentale sur le calcul analogique chimique, les analyses statiques des réseaux d'interactions moléculaires en théorie des graphes, les analyses dynamiques en logique temporelle quantitative, et les correspondances entre structure et dynamique de tels réseaux. En collaboration avec des biologistes, y compris de l'industrie pharmaceutique, nous étudions des questions biologiques et biomédicales concrètes, et développons des modèles et des méthodes computationnels pour y répondre. Nos développements logiciels reposent notamment sur des techniques originales de prototypage rapide en programmation logique avec contraintes.

2. Codes de la pipeline principale

3. Notebook SEPI

```python
# Lifeware.py

from abc import ABC, abstractmethod
import colorama
from colorama import Fore
import numpy as np
import os
import itertools


adress = "C:\\Users\\jlnbn\\Downloads\\Stage 3A - Lifeware"

os.chdir(adress)
### SOAP API
from zeep import Client
import hashlib

wsdl = "https://www.brenda-enzymes.org/soap/brenda_zeep.wsdl"
password = hashlib.sha256("brendabiocham".encode("utf-8")).hexdigest()
client = Client(wsdl)


def hashed_metabolite(name):
    parameters = ("julien.bienvenu@polytechnique.edu",password,name)
    resultString = client.service.getLigandStructureIdByCompoundName(*parameters)
    return resultString

### Operations on sets/listes


def findsets(s, n='full'):
    if n=='full':
        n=len(s)
    return list(itertools.permutations(s, n))

def obj_is_in_list(Obj,List):
    Listofcompare = [(Obj==Listj) for Listj in List]
    Bool = False
    for b in Listofcompare:
        Bool = Bool or b
    return Bool


def dictionnary_lengths(dic):
    Dic = dict()
    for key in dic:
        Dic[key] = len(dic[key])
    return Dic

def disjoint(a,b):
    return (len(set(a).intersection(b))==0)

def union(a,b):
    return list(set(a).union(set(b)))

def included(a,b):
    return set(a).issubset(set(b))

def diff(a,b):
    return list(set(a).difference(set(b)))

# Renvoie si deux listes a et b s'intersectent : any(i in a for i in b)
# Renvoie si a inclut dans b : set(a).issubset(set(b))


###

# def pop2str (S,t=-1):
#     l = list(S)
#     l.pop(t)
#     s = ""
#     for u in l:
#         s+=u
#    return (s)


crypte=open("brendaReaction.tsv",'r')
Texte=crypte.readlines()
#Texte = [pop2str(i) for i in Texte]
Lecture = [i.split("\t") for i in Texte]

for i in range(len(Lecture)) :
    #print(l)
    #print(l)
    l = Lecture[i]
    l[3],l[4] = l[3].split("-+-"),l[4].split("-+-")
    Lecture[i] = l[:12]
#Lecture = list(np.array(Lecture).transpose()[1])
crypte.close()
def comptability_lecture(Lecture):
    Metabolites = set()
    Enzymes = set()
    ECs = set()
    i = 0
    for l in Lecture :
```

```
            i+=1
            Metabolites = Metabolites.union(set(l[3]+l[4]))
            ECs = ECs.union({l[0]})
            Enzymes = Enzymes.union({l[1]})
            if i%50==0:
                s = "\r "+str(i)+" ECs : "+str(len(ECs))+", Enzymes : "+str(len(Enzymes))+", Metabolites : "+str(len(Metabolites))
                print(s, end="")
        s = "\r "+str(i)+" ECs : "+str(len(ECs))+", Enzymes : "+str(len(Enzymes))+", Metabolites : "+str(len(Metabolites))
        print(s, end="")

        crypte=open("Metabolites.tsv",'w')
        j=0
        for m in Metabolites :
            crypte.write(m+'\n')
            j+=1
            s = "\r Metabolites written : "+str(j)
            print(s, end="")
        crypte.close()

    def redundancy_lecture(Lecture):
        EC_Enzymes = set()
        i = 0
        for l in Lecture :
            i+=1
            #Metabolites = Metabolites.union(set(l[3]+l[4]))
            EC_Enzymes = EC_Enzymes.union({(l[0],l[1])})
            if i%50==0:
                s = "\r "+str(i)+" EC-Enzymes couples : "+str(len(EC_Enzymes))
                print(s, end="")
        s = "\r "+str(i)+" EC-Enzymes : "+str(len(EC_Enzymes))
        print(s, end="")
        L=[]
        for e in EC_Enzymes:
            L.append(e[1])
        values,counts = np.unique(L,return_counts=True)
        print('\n')
        i=0
        for c in counts :
            if c>1:
                v = values[i]
                for e in EC_Enzymes :
                    if e[1]==v:
                        print(e)
            i+=1

    def clean_lecture(Lecture,badfilename,aff=True):
        L = []
        crypte=open(badfilename,'r')
        Texte=crypte.readlines()
        BadLists = [int(i) for i in Texte]
        crypte.close()
        i,j = 0,0
        for l in Lecture:
            if not(i in BadLists):
                L.append(l)
            else :
                j+=1
            if aff :
                s = "\r Entries read "+str(i+1)+" Entries rejected : "+str(j)
                print(s, end="")
            i+=1
        return L

    def is_number(s):
        try:
            float(s)
            return True
        except ValueError:
            return False

    def with_KM(Lecture):
        i,j=0,0
        for l in Lecture:
            i+=1
            if is_number(l[6]):
                j+=1
            s = "\r Entries read "+str(i)+" Entries with Km : "+str(j)
            print(s, end="")

    Lecture = clean_lecture(Lecture,"brendaReactionBadEntries.tsv")
    #redundancy_lecture(Lecture)
    #with_KM(Lecture)
    ##
    Glossaire = []
    j=0
    for l in Lecture :
        g = '('+str(len(l[3]))+','+str(len(l[4]))+')'
        if g == '(2,2)':
            if (l[3][0] in l[4]) or (l[3][1] in l[4]):
                j+=1
        Glossaire.append(g)

    values,counts = np.unique(Glossaire,return_counts=True)
    print('(nbre entrées,nbre sorties) -> nbre de réaction de ce type dans Brenda nettoyé')
    for i in range(len(values)):
```

```
    v,c = values[i],counts[i]
    print(v+' -> '+str(c)+' '+str(100*c/len(Glossaire))+'%')

##
def lexicon_metabolites(Lecture):
    Metabolites = set()
    Dictionnary = dict()
    Bad_Entries = []
    i,j,k = 0,0,0
    for l in Lecture :
        i+=1
        L = [m for m in l[3]+l[4] if not(m in Metabolites)]
        for m in L :
            m_id = hashed_metabolite(m)
            if not(m_id==''):
                j+=1
                if m_id in Dictionnary :
                    Dictionnary[m_id].append(m)
                else :
                    Dictionnary[m_id] = [m]
            else :
                k+=1
                Bad_Entries.append(m)
            Metabolites.add(m)
        s = "\r Entries read "+str(i)+", Metabolites not recognized : "+str(k)+", Recognized : "+str(j)
        print(s, end="")
    return Dictionnary, Bad_Entries

def Write_Metabolites(Lecture):
    Metabolites = set()
    crypte=open("Metabolites.tsv",'w')
    i,j=0,0
    LL = Lecture.copy()
    LL.pop(0)
    for l in LL :
        i+=1
        L = [m for m in l[3]+l[4] if not(m in Metabolites)]
        for m in L :
            j+=1
            crypte.write(m+'\n')
            Metabolites.add(m)
        s = "\r Entries read "+str(i)+", Metabolites written : "+str(j)
        print(s, end="")
    crypte.close()

def Write_Metabolites_Id(filename):
    crypte=open("Metabolites.tsv",'r')
    Texte=crypte.read().split('\n')
    crypte.close()
    i=11396
    for m in Texte[i:] :
        #print(m+'\n')
        i+=1
        crypte=open(filename,'a')
        if m.isprintable():
            m_id = hashed_metabolite(m)
        else :
            m_id ='000 not unicode'
        if m_id=='':
            m_id='000'
        if m_id==None:
            m_id='000'
        crypte.write(m+'\t'+m_id+'\n')
        crypte.close()
        s = "\r Entries read and identified "+str(i)
        print(s, end="")

def Write_Metabolites_Id_with_prefilling(filename,prefilename):
    crypte=open(prefilename,'r')
    Texte=crypte.read().split('\n')
    Lecture = [l.split('\t') for l in Texte]
    crypte.close()
    i,j=4379,0
    for u in Lecture[i:] :
        i+=1
        m,m_i = u[0],u[1]
        #print('\n'+m+'\n')
        crypte=open(filename,'a')
        if m_i=='?':
            j+=1
            if m.isprintable():
                m_id = hashed_metabolite(m)
            else :
                m_id ='000 not unicode'
            if m_id=='':
                m_id='000'
            if m_id==None:
                m_id='000'

        else :
            m_id = m_i
        crypte.write(m+'\t'+m_id+'\n')
        crypte.close()
        s = "\r Entries read and identified "+str(i)+" previously unindentified : "+str(j)+" with id : "+m_i+'|'+m_id
        print(s, end="")
```

```
#Write_Metabolites(Lecture)
#Write_Metabolites_Id_with_prefilling("Lexique2.tsv","Lexique_test.tsv")


#comptability_lecture(Lecture)
#print(Lecture[4])
### Lexique exploitation

def Lexiques_building(filename):
    crypte=open(filename,'r')
    Texte=crypte.read().split('\n')
    crypte.close()
    meta_to_key = dict()
    key_to_metas = dict()
    Table = [l.split('\t') for l in Texte]
    Table.pop(-1)
    i=0
    for u in Table:
        i+=1
        if len(u)<2:
            print(i,u)
        meta,key = u[0],u[1]
        meta_to_key[meta]=key
        if key in key_to_metas :
            key_to_metas[key].append(meta)
        else :
            key_to_metas[key] = [meta]
    return meta_to_key,key_to_metas

Lex1,Lex2 = Lexiques_building("Lexique2.tsv")
#i,j=0,0
Unknowns = Lex2['000']
#Unknowns.pop(-1)

def Write_Metabolites_Id_with_archive(filename,archivename):
    crypte=open("Metabolites.tsv",'r')
    Texte=crypte.read().split('\n')
    crypte.close()
    Lex1,Lex2 = Lexiques_building(archivename)
    i,j = 0,0
    for m in Texte :
        i+=1
        crypte=open(filename,'a')
        if m in Lex1:
            m_id = Lex1[m]
        else :
            m_id = '?'
            j+=1
        crypte.write(m+'\t'+m_id+'\n')
        crypte.close()
        s = "\r Entries read : "+str(i)+", unidentified :"+str(j)
        print(s, end="")

#Write_Metabolites_Id_with_archive("Lexique_test.tsv","LexiqueArchive.csv")

#
# for meta in Unknowns :
#     print(meta,hashed_metabolite(meta))
### Numbering
NumberedLecture = []
i=0
for l in Lecture :
    n = l.copy()
    n[3]=[Lex1[s] for s in l[3]]
    n[4]=[Lex1[s] for s in l[4]]
    NumberedLecture.append(n)


###
i,j=0,0
for l in Lecture[1:] :
    i+=1
    metas = l[3]+l[4]
    if not(disjoint(metas,Unknowns)):
        j+=1
        crypte=open('brendaReactionBadEntries.tsv','a')
        crypte.write(str(i)+'\n')
        crypte.close()
    s = "\r Entries read "+str(i)+" Entries rejected : "+str(j)
    print(s, end="")


###

SetR = set()
for n in NumberedLecture :
    s = 'm'+n[3][0]
    for r in n[3][1:]:
        s+=' + m'+r
    s+=' => m'+n[4][0]
    for p in n[4][1:]:
        s+=' + m'+p
    s+='.'
```

```
    if not(s in SetR):
        #print(s)
        crypte = open("BrendaBiochamReactionsM.bc",'a')
        crypte.write(s+'\n')
        crypte.close
        SetR = SetR.union({s})


###
#print(Fore.RED + 'This text is rouge in color')


class Metabolite:
    def __init__(self,name):
        self.name = name
        self.synonyms = [name]

    def add_synonyms(self,liste):
        self.synonyms = self.synonyms + list(set(liste) - set(self.synonyms))

    def sameas(self,metabolite):
        return obj_is_in_list(self.name,metabolite.synonyms) or obj_is_in_list(metabolite.name,self.synonyms)

# met1 = Metabolite("Glucose")
# met1.add_synonyms(["Sugar"])
#
# met2 = Metabolite("Sugar")
# met1.sameas(met2)

# class Reaction:
#     def

def in_and_out_formating(in_and_outs):
    values,counts = np.unique(in_and_outs,return_counts=True)
    dic = dict(zip(values,counts))
    for label in ['in','out','inh'] :
        if label not in dic :
            dic[label] = 0
    #dic.pop('-')
    return dic

class Abstract_Reaction:
    def __init__(self,in_and_outs,env,kin):
        self.config = in_and_out_formating(in_and_outs)
        self.environment = env
        self.kinetics = kin


class Concrete_Reaction:
    def __init__(self,l):
        self.liste = l
        self.ec = self.liste[0]
        self.enzyme_name = self.liste[1]
        self.organism = self.liste[2]
        self.reactants = self.liste[3]
        self.products = self.liste[4]

    def soft_match(self,abs_react):
        nb_in, nb_out = len(self.reactants), len(self.products)
        Nb_in, Nb_out = abs_react.config['in'], abs_react.config['out']
        return ((nb_in == Nb_in) and (nb_out == Nb_out))

    def hard_match(self,abs_react):
        nb_in, nb_out = len(self.reactants), len(self.products)
        Nb_in, Nb_out = abs_react.config['in'], abs_react.config['out']
        return ((nb_in >= Nb_in) and (nb_out >= Nb_out))

class Concrete_Reaction_Library:
    def __init__ (self,big_liste):
        self.library = []
        for l in big_liste :
            self.library.append(Concrete_Reaction(l))

    def select_candidates(self,CRN_abstracts, soft_matching=True):
        Book = CRN_abstracts.reaction_book()
        Precandidates = dict()
        n = len(self.library)
        for key in Book :
            l = []
            for i in range(n) :
                react = self.library[i]
                if soft_matching and (react.soft_match(Book[key])):
                    l.append(i)
                if (not soft_matching) and (react.hard_match(Book[key])):
                    l.append(i)
            Precandidates[key] = l
        return Precandidates


Bibliotheque = Concrete_Reaction_Library(NumberedLecture)
#D = dictionnary_lengths(Bibliotheque.select_candidates(CRN_abs))

def dictionnarize(liste):
    dic = dict()
    for i in range(len(liste)):
```

```
        dic[liste[i]] = i
    return dic

def coloring(i,CRN_blue):
    for j in range(len(CRN_blue[i])):
        color = CRN_blue[i,j]
        if (color =='vert' or color=='rouge') :
            CRN_blue[i,j] = 'vert'

    return CRN_blue

def score(j,CRN_blue):
    values,counts = np.unique(CRN_blue[:,j],return_counts=True)
    dic = dict(zip(values,counts))
    if 'vert' not in dic :
        dic['vert'] = 0
    if 'rouge' not in dic :
        dic['rouge'] = 0
    return dic

def nb_red_out(j,CRNe,CRN_blue):
    N, nb_m = 0,len(CRNe[:,0])
    for i in range(nb_m):
        color,type = CRN_blue[i,j],CRNe[i,j]
        #print(color+" and "+type)
        if color == 'rouge' and type =='out' :
            N+=1
    return N


class CRNa :

    def __init__(self,liste):
        self.abstract = np.array(liste)
        self.extract = self.abstract[1::,1::]
        self.enzymes = self.abstract[0,1:]
        self.metabolites = self.abstract[1:,0]

    def reaction_book(self):
        EC_dic = dictionnarize(self.enzymes)
        CRNe = self.extract
        Book = dict()
        for key in dictionnarize(self.enzymes):
            Book[key] = Abstract_Reaction(CRNe[:,EC_dic[key]],"any","any")
        return Book

    def blueprint(self):
        return np.array([[(j=="-")*'gris' + (j=='out' or j=='in' or j=='inh')*'rouge' for j in listj] for listj in
list(np.copy(self.extract))])

    def Delta(self,A,E):
        #print(self.extract)
        CRNe = self.extract
        EC_dic, Meta_dic = dictionnarize(self.enzymes), dictionnarize(self.metabolites)
        return CRNe[Meta_dic[A],EC_dic[E]]


    def parcours(self,init):
        #print(self.extract)
        CRNe = self.extract
        CRN_EC = self.enzymes
        CRN_Meta = self.metabolites
        CRN_blue = self.blueprint()
        nb_e, nb_m = len(CRN_EC),len(CRN_Meta)
        EC_dic, Meta_dic = dictionnarize(self.enzymes), dictionnarize(self.metabolites)

        for Meta_i in init:
            #print(Meta_i)
            #print(Meta_dic[Meta_i])
            CRN_blue = coloring(Meta_dic[Meta_i],CRN_blue)

        Steps = [['Empty',init,'Empty']]
        To_do_list = [i for i in range(nb_e)]
        while len(To_do_list)>0 :
            J, G, R = -1,-1,-1
            #print([CRN_EC[j] for j in To_do_list])
            for j in range(len(To_do_list)):
                dic_j = score(To_do_list[j],CRN_blue)
                g,r = dic_j['vert'],dic_j['rouge']
                #print("EC :"+CRN_EC[To_do_list[j]]+" scores vert :"+str(g)+" score rouge :"+str(r))
                if g>0 :
                    if (g==G) and (r==R):
                        n,N = nb_red_out(To_do_list[j],CRNe,CRN_blue),nb_red_out(To_do_list[J],CRNe,CRN_blue)
                        #print("j EC :"+CRN_EC[To_do_list[j]]+" scores vert :"+str(g)+" score rouge :"+str(r)+" score rouge out
:"+str(n))
                        #print("J EC :"+CRN_EC[To_do_list[J]]+" scores vert :"+str(G)+" score rouge :"+str(R)+" score rouge out

:"+str(N))
                        if n>N :
                            J,G,R = j,g,r
                    if (g==G) and (r<R) :
                        J,G,R = j,g,r
                    if g > G :
                        J,G,R = j,g,r
            J = To_do_list.pop(J)
            Fillable, Filled = [],[]
```

```
            for i in range(nb_m):
                if CRN_blue[i,J] == 'vert' :
                    Filled.append(CRN_Meta[i])
                if CRN_blue[i,J] == 'rouge' :
                    Fillable.append(CRN_Meta[i])
                    CRN_blue = coloring(i,CRN_blue)
            Steps.append([CRN_EC[J],Fillable,Filled])
        return Steps

liste_abs = [['-','E','F','G','H'],['A','in','-','-','-'],['B','out','in','in','-'],['C','out','in','-','-'],
['D','out','out','-','-'],['I','inh','-','-','-'],['J','out','-','-','inh'],['K','-','out','in','-'],['L','-','out','-','out'],
['M','-','-','out','in'],['N','-','-','out','in'],['O','-','-','-','out']]

CRN_abs2 = np.array([['-','E','F','G','H'],['A','in','-','-','-'],['B','in','out','-','-'],['C','out','-','in','-'],
['D','out','-','in','-'],['I','inh','out','-','-'],['J','-','-','-','in'],['K','-','-','out','-'],['L','in','-','-','out'],
['M','-','in','-','out']])


CRN_REFERENCE = CRNa([
['-','W','X','Y','Z'],
['A','-','in','out','-'],
['B','in','-','-','-'],
['C','in','out','-','-'],
['D','out','in','-','-'],
['E','out','-','-','-'],
['G','-','out','in','-'],
['H','-','-','out','in'],
['I','-','-','-','in'],
['J','-','-','-','out'],
])

CRN_REF_LONG = CRNa([
['-','W','X','Y','Z','ZZ'],
['A','-','in','out','-','-'],
['B','in','-','-','-','-'],
['C','in','out','-','-','-'],
['D','out','in','-','-','-'],
['E','out','-','-','-','-'],
['G','-','out','in','-','-'],
['H','-','-','out','in','-'],
['I','-','-','-','in','-'],
['J','-','-','-','-','out'],
['K','-','-','-','out','in']
])

CRN_REF_COURT =CRNa([
['-','W','X','Y'],
['A','-','in','out'],
['B','in','-','-'],
['C','in','out','-'],
['D','out','in','-'],
['E','out','-','-'],
['G','-','out','in'],
['H','-','-','out'],
])


CRN_abs = CRNa(liste_abs)

CRN_test = CRNa([['-','E'],['A','in'],['B','out']])


def concatenate(dic):
    s = ''
    for key in dic.keys():
        s+=key+': '+dic[key]+'; '
    return s

def f(CRNabs,Library,Starter,Grey=False):
    Parcours = CRNabs.parcours(list(Starter.keys()))
    Precandidats = Library.select_candidates(CRNabs,not(Grey))
    # parcourir_dict = dict()
    # for key in Precandidats.keys() :
    #     parcourir_dict[key]=len(Precandidats[key])
    # parcourir = [parcourir_dict[step[0]] for step in Parcours[1:]]
    #print(parcourir)
    Blacklist_EC = []
    D_EC = dict()
    Blacklist_M = list(Starter.values())
    D_M = Starter.copy()
    Greylist_M = []
    global S, S_E
    S = []
    S_E = []
    def aux(Steps,B_EC,B_M):
        #print(Steps)
        if len(Steps)==0 :
            #print(len(Solutions))
            S.append([D_EC,D_M])
        else :
            E,A,X = Steps[0]
            #print(E,A,X,D_M,D_EC,B_EC,B_M)

            A_in = [key for key in A if (CRNabs.Delta(key,E)=='in')]
            A_out = [key for key in A if (CRNabs.Delta(key,E)=='out')]
```

```
        x_in = [D_M[key] for key in X if (CRNabs.Delta(key,E)=='in')]
        x_out = [D_M[key] for key in X if (CRNabs.Delta(key,E)=='out')]
        PCandidats = [Library.library[i] for i in Precandidats[E]]
        Candidats = []
        for c in PCandidats :
            #print(c.enzyme_name)
            c_in,c_out = c.reactants,c.products
            if not(c.ec in B_EC):
                #print(x_in,c_in,included(x_in,c_in),x_out,c_out,included(x_out,c_out))
                if included(x_in,c_in) and included(x_out,c_out):
                    a_in, a_out = diff(c_in,x_in),diff(c_out,x_out)
                    if disjoint(B_M,a_in+a_out):
                        Candidats.append(c)
                    #else :
                    #    print(B_M,a_in+a_out,"other metabolites already blacklisted")
                #else :
                #    print("already filled metabolite not found")
            #else :
            #    print (c.ec," enzyme blacklisted")
        if len(Candidats)!=0 :
            #print(E+" nbre candidats : "+str(len(Candidats)))
            for c in Candidats :
                e,ec = c.enzyme_name,c.ec
                D_EC[E] = e
                c_in,c_out = c.reactants,c.products
                e_in, e_out = diff(c_in,x_in),diff(c_out,x_out)
                permutations_in, permutations_out = findsets(e_in),findsets(e_out)
                for pi in permutations_in:
                    for po in permutations_out:
                        #print(A_in,A_out,pi,po)
                        for i in range(len(A_in)):
                            D_M[A_in[i]] = pi[i]
                        for j in range(len(A_out)):
                            D_M[A_out[j]] = po[j]
                        aux(Steps[1:],B_EC+[ec],B_M+e_in+e_out)

        #else :
        #    print("No Candidate Left")
        #for a in A:
        #    D_M.popite(a)
        #D_M.popitem(E)

def aux_grey(Steps,B_EC,B_M,G_M):
    #print(Steps)
    if len(Steps)==0 :
        s_EC = concatenate(D_EC)
        if not(s_EC in S_E):
            S_E.append(s_EC)
            print(D_EC)
        #print([D_EC,D_M,G_M])
        S.append([D_EC,D_M,G_M])
    else :
        E,A,X = Steps[0]
        #print(Fore.RED+str(E)+str(A),Fore.BLACK,X,D_M,D_EC,B_EC,B_M)

        A_in = [key for key in A if (CRNabs.Delta(key,E)=='in')]
        n_in = len(A_in)
        A_out = [key for key in A if (CRNabs.Delta(key,E)=='out')]
        n_out = len(A_out)
        x_in = [D_M[key] for key in X if (CRNabs.Delta(key,E)=='in')]
        x_out = [D_M[key] for key in X if (CRNabs.Delta(key,E)=='out')]
        PCandidats = [Library.library[i] for i in Precandidats[E]]
        Candidats = []
        for c in PCandidats :
            #print(c.enzyme_name)
            c_in,c_out = c.reactants,c.products
            if not(c.ec in B_EC):
                #print(x_in,c_in,included(x_in,c_in),x_out,c_out,included(x_out,c_out))
                if included(x_in,c_in) and included(x_out,c_out):
                    a_in, a_out = diff(c_in,x_in),diff(c_out,x_out)
                    if disjoint(B_M,a_in+a_out):
                        Candidats.append(c)
                    #else :
                    #    print(B_M,a_in+a_out,"other metabolites already blacklisted")
                #else :
                #    print("already filled metabolite not found")
            #else :
            #    print (c.ec," enzyme blacklisted")
        if len(Candidats)!=0 :
            #print(E+" nbre candidats : "+str(len(Candidats)))
            for c in Candidats :
                e,ec = c.enzyme_name,c.ec
                D_EC[E] = e
                c_in,c_out = c.reactants,c.products
                e_in, e_out = diff(diff(c_in,x_in),G_M),diff(diff(c_out,x_out),G_M)
                if (len(e_in)>= n_in)and(len(e_out)>= n_out):
                    permutations_in, permutations_out = findsets(e_in,n_in),findsets(e_out,n_out)
                    for pi in permutations_in:
                        for po in permutations_out:
                            #print(A_in,A_out,pi,po)
                            for i in range(len(A_in)):
                                D_M[A_in[i]] = pi[i]
                            for j in range(len(A_out)):
                                D_M[A_out[j]] = po[j]
                            aux_grey(Steps[1:],B_EC+[ec],B_M+e_in+e_out,G_M+diff(e_in,pi)+diff(e_out,po))
```

```
            #else :
            #    print("No Candidate Left")
            #for a in A:
            #    D_M.popite(a)
            #D_M.popitem(E)
    if Grey :
        aux_grey(Parcours[1:],Blacklist_EC,Blacklist_M,Greylist_M)
    else :
        aux(Parcours[1:],Blacklist_EC,Blacklist_M)
    return S

a = Lex1['Acetone']
b = Lex1['D-glucose']
h = Lex1['H2O2']
j = Lex1['resorufin']
i = Lex1['resazurin']

CRN_very_short=CRNa([
['-','Z','ZZ'],
['I','in','-'],
['J','-','out'],
['H','in','-'],
['K','out','in']
])
Starter = {'A': a, 'B': b, 'J': j}

S = f(CRN_REF_LONG,Bibliotheque,Starter,Grey=True)


### Reading solutionshort to find a good subset of reactions to include for sepi search
import ast
crypte = open("SolutionsShort.tsv",'r')
Texte = crypte.readlines()
crypte.close()
J = 5
CleanedT,CleanedM = set(),set()
for t in Texte[:J]:
    l,m = '',''
    b,c,d,e = False,False,False,False
    for s in t:
        if s=='{':
            b=True
        if b and not(c):
            l+=s
        if c and not(d):
            if s=='{':
                e=True
            if e and not(d):
                m+=s
            if s=='}':
                d=True
        if s=='}':
            c=True

    d1,d2=ast.literal_eval(l),ast.literal_eval(m)
    CleanedT = CleanedT.union(set(d1.values()))
    CleanedM = CleanedM.union(set(d2.values()))

CleanedM = CleanedM.difference({'1','2'})
SetR = set()
i=0

def does_intersect(set1,list1,list2):
    set2 = set(list1).union(set(list2))
    set3 = set1.intersection(set2)
    return (len(set3) > 0)

for n in NumberedLecture :
    if n[1] in CleanedT and does_intersect(CleanedM,n[3],n[4]) :
        s = 'm'+n[3][0]
        for r in n[3][1:]:
            s+=' + m'+r
        s+=' => m'+n[4][0]
        for p in n[4][1:]:
            s+=' + m'+p
        s+='.'
        if not(s in SetR):
            i+=1
            print(i,n[1])

            crypte = open('BrendaBiochamReactionsSmart'+str(J)+'SubsetM.bc','a')
            crypte.write(s+'\n')
            crypte.close
            SetR = SetR.union({s})
```

SEPI_Tryouts_beta3

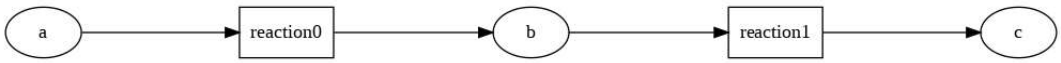
June 27, 2022

[2]:
```
about.
```

[2]:
```
Biocham 4.6.19
Copyright (C) 2003-2020 Inria, EPI Lifeware, Saclay-Île de France, France,
license GNU GPL 2, http://lifeware.inria.fr/biocham4/
```

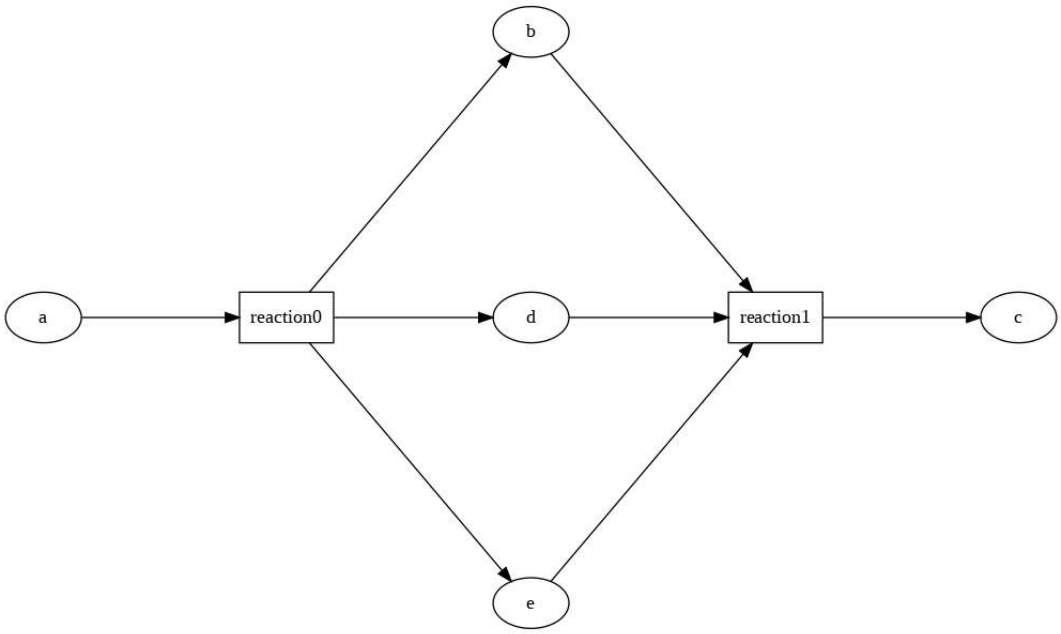
1 0. Playing with SEPI

[3]:
```
clear_model.
new_model(Basic1).
a => b.
b => c.
export_biocham(Basic1.bc).
list_model.
draw_reactions.
```



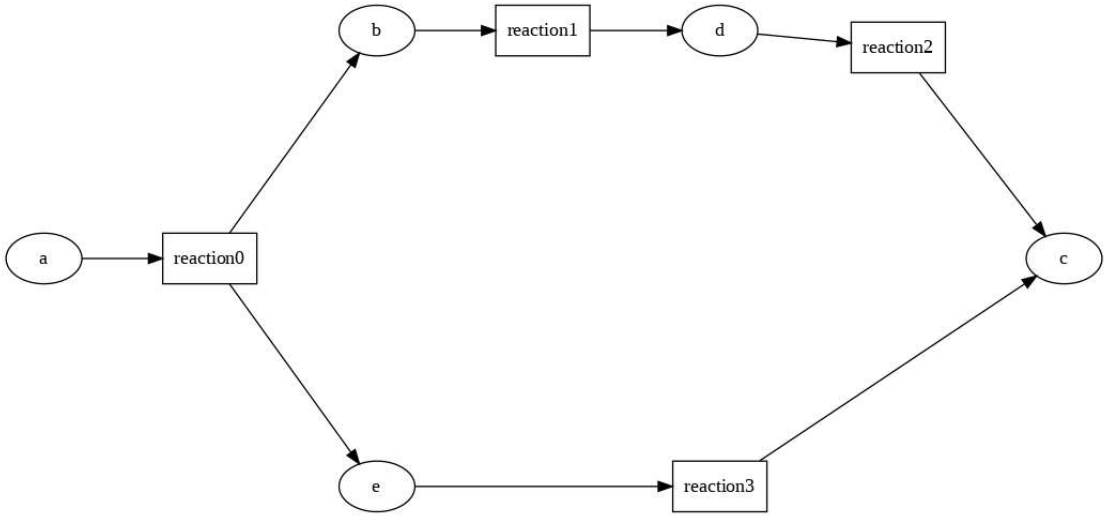
[3]:
```
MA(1) for a=>b.
MA(1) for b=>c.
```

[4]:
```
new_model(Basic2).
a => b+d+e.
b+d+e => c.
export_biocham(Basic2.bc).
list_model.
draw_reactions.
```



[4]:
```
MA(1) for a=>b+d+e.
MA(1) for b+d+e=>c.
```

[5]:
```
new_model(Basic3).
a => b+e.
b => d.
d => c.
e => c.
export_biocham(Basic3.bc).
list_model.
draw_reactions.
```



[5]:
```
MA(1) for a=>b+e.
MA(1) for b=>d.
MA(1) for d=>c.
MA(1) for e=>c.
```

How reductions work on these simple examples : with or without merge restriction

[6]:
```
search_reduction(Basic2.bc, Basic1.bc, mapping_restriction : [a->a,c->c],␣
↪merge_restriction : not_species).
```

[6]:
```
sepi
a -> a
b -> deleted
d -> deleted
e -> b
c -> c
{a => b+d+e} -> {a => b}
{b+d+e => c} -> {b => c}
Number of reductions: 1
```

[7]:
```
search_reduction(Basic2.bc, Basic1.bc, mapping_restriction : [a->a,c->c],␣
↪merge_restriction : not_species, show_support:yes).
```

[7]:
```
sepi
a
e
c
a => b+d+e
b+d+e => c
Number of reductions: 1
```

[16]:
```
search_reduction(Basic2.bc, Basic1.bc, mapping_restriction : [a->a,c->c]).
```

[16]:
```
sepi
a -> a
b -> b
d -> b
e -> b
c -> c
{a => b+d+e} -> {a => b}
{b+d+e => c} -> {b => c}
Number of reductions: 1
```

In the same way, with different paths to the reaction :

[17]:
```
search_reduction(Basic3.bc, Basic1.bc, mapping_restriction : [a->a,c->c],␣
↪merge_restriction : not_species).
```

[17]:
```
sepi
a -> a
b -> deleted
e -> b
d -> deleted
c -> c
{a => b+e} -> {a => b}
{b => d} -> deleted
{d => c} -> {b => c}
{e => c} -> {b => c}
Number of reductions: 1
```

[18]:
```
search_reduction(Basic3.bc, Basic1.bc, mapping_restriction : [a->a,c->c]).
```

[18]:
```
sepi
a -> a
b -> a
e -> b
d -> b
c -> c
{a => b+e} -> deleted
{b => d} -> {a => b}
```

```
{d => c} -> {b => c}
{e => c} -> {b => c}
Number of reductions: 1
```

We can also extract every sepi reduction :

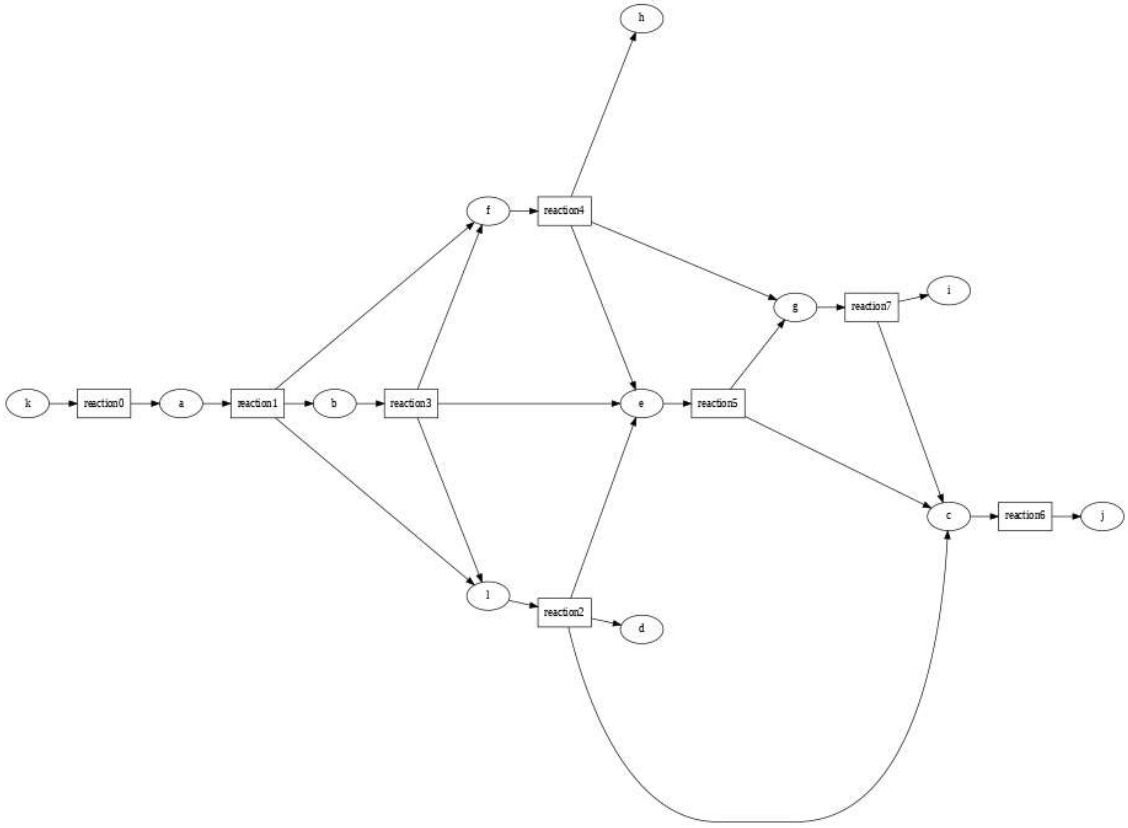
[19]:
```
search_reduction(Basic2.bc, Basic1.bc, mapping_restriction : [a->a,c->c],␣
↪merge_restriction : not_species, all_reductions : yes).
```

[19]:
```
sepi
a -> a
b -> deleted
d -> deleted
e -> b
c -> c
{a => b+d+e} -> {a => b}
{b+d+e => c} -> {b => c}
sepi
a -> a
b -> b
d -> deleted
e -> deleted
c -> c
{a => b+d+e} -> {a => b}
{b+d+e => c} -> {b => c}
sepi
a -> a
b -> deleted
d -> b
e -> deleted
c -> c
{a => b+d+e} -> {a => b}
{b+d+e => c} -> {b => c}
no sepi found
Number of reductions: 3
```

Let's try a more complicated example :

[9]:
```
new_model(Basic4).
k => a.
a => b+f+l.
l => d+e+c.
b => l+e+f.
f => h+e+g.
e => g+c.
c => j.
g => i+c.
export_biocham(Basic4.bc).
```

```
draw_reactions.
```



[10]:
```
search_reduction(Basic4.bc, Basic1.bc, mapping_restriction : [a->a,c->c],␣
↪merge_restriction : not_species).
```

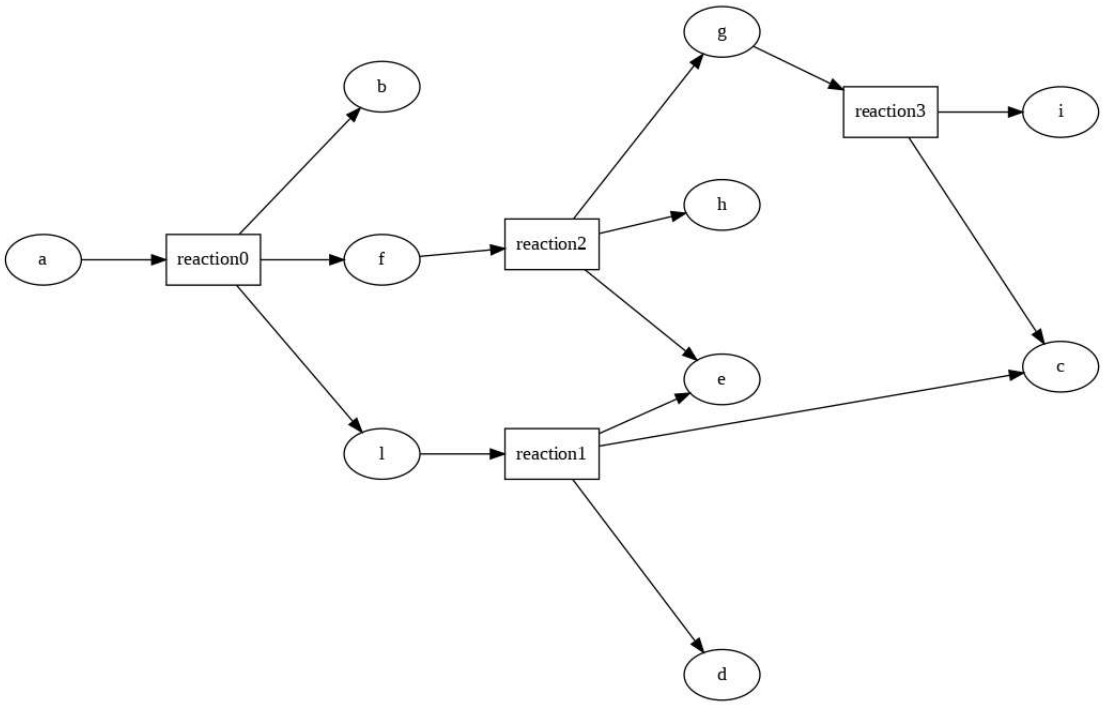
[10]:
```
sepi
k -> deleted
a -> a
b -> deleted
f -> deleted
l -> deleted
c -> c
d -> deleted
e -> deleted
g -> b
h -> deleted
j -> deleted
i -> deleted
{k => a} -> deleted
```

```
{a => b+f+l} -> {a => b}
{l => c+d+e} -> {b => c}
{b => e+f+l} -> deleted
{f => e+g+h} -> {a => b}
{e => c+g} -> deleted
{c => j} -> deleted
{g => c+i} -> {b => c}
Number of reductions: 1
```

[11]:
```
search_reduction(Basic4.bc, Basic1.bc, mapping_restriction : [a->a,c->c],␣
↪merge_restriction : not_species,show_support : yes).
```

[11]:
```
sepi
a
c
g
a => b+f+l
l => c+d+e
f => e+g+h
g => c+i
Number of reductions: 1
```

[12]:
```
new_model.
a => b+f+l.
l => c+d+e.
f => e+g+h.
g => c+i.
draw_reactions.
```



[49]:
```
search_reduction(Basic4.bc, Basic2.bc, mapping_restriction : [a->a,c->c],␣
↪merge_restriction : not_species).
```

[49]:
```
no sepi found
Number of reductions: 0
```

[50]:
```
search_reduction(Basic4.bc, Basic3.bc, mapping_restriction : [a->a,c->c],␣
↪merge_restriction : not_species).
```

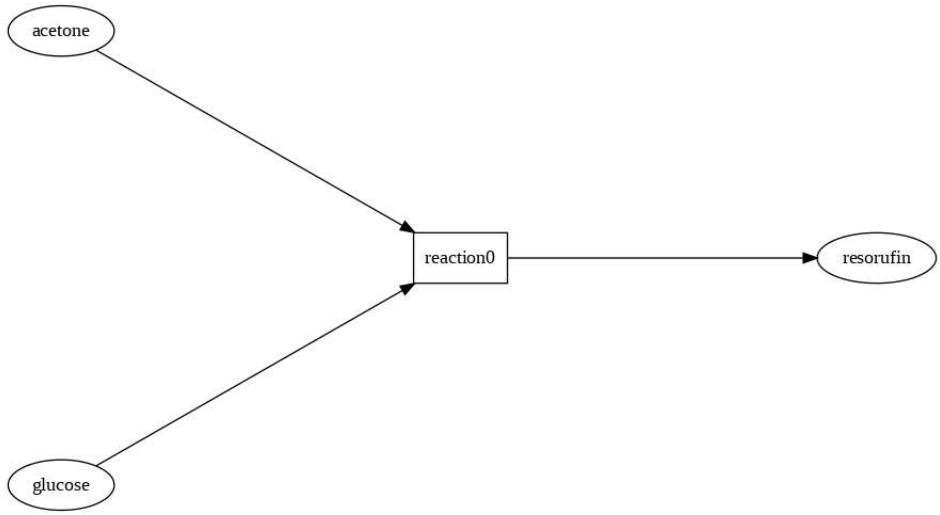
[50]:
```
sepi
k -> deleted
a -> a
b -> deleted
f -> b
l -> e
c -> c
d -> deleted
e -> deleted
g -> d
h -> deleted
j -> deleted
i -> deleted
{k => a} -> deleted
```

```
{a => b+f+l} -> {a => b+e}
{l => c+d+e} -> {e => c}
{b => e+f+l} -> {a => b+e}
{f => e+g+h} -> {b => d}
{e => c+g} -> deleted
{c => j} -> deleted
{g => c+i} -> {d => c}
Number of reductions: 1
```

2 1. First CRNs

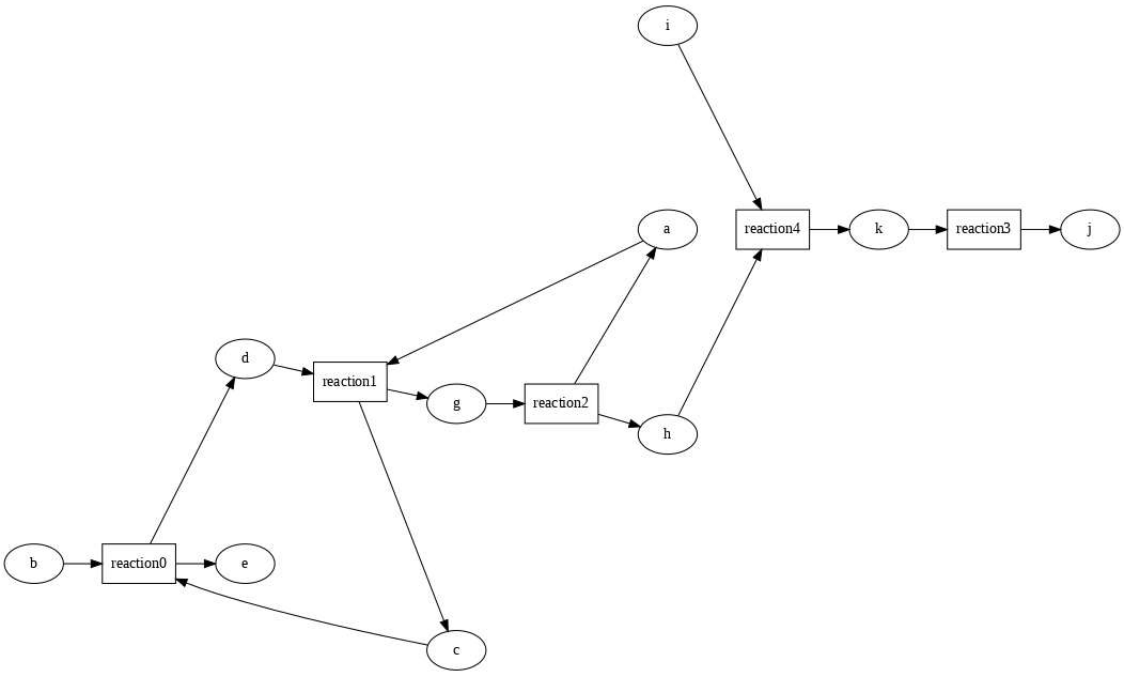
CRNAlexis is the cCRN developped in Alexis' thesis CRNVeryShort is one example of aCRN to concretize : acetone + glucose => resorufin CRNShort is Alexis' CRN but with letters to correspond more to a aCRN CRNLong is an aCRN on which solutions (as SISO*) can be found in BRENDA

[13]:
```
clear_model.
new_model(CRNVeryShort).
acetone+glucose=>resorufin.
export_biocham(CRNVeryShort.bc).
list_model.
draw_reactions.
```



[13]: MA(1) for acetone+glucose=>resorufin.

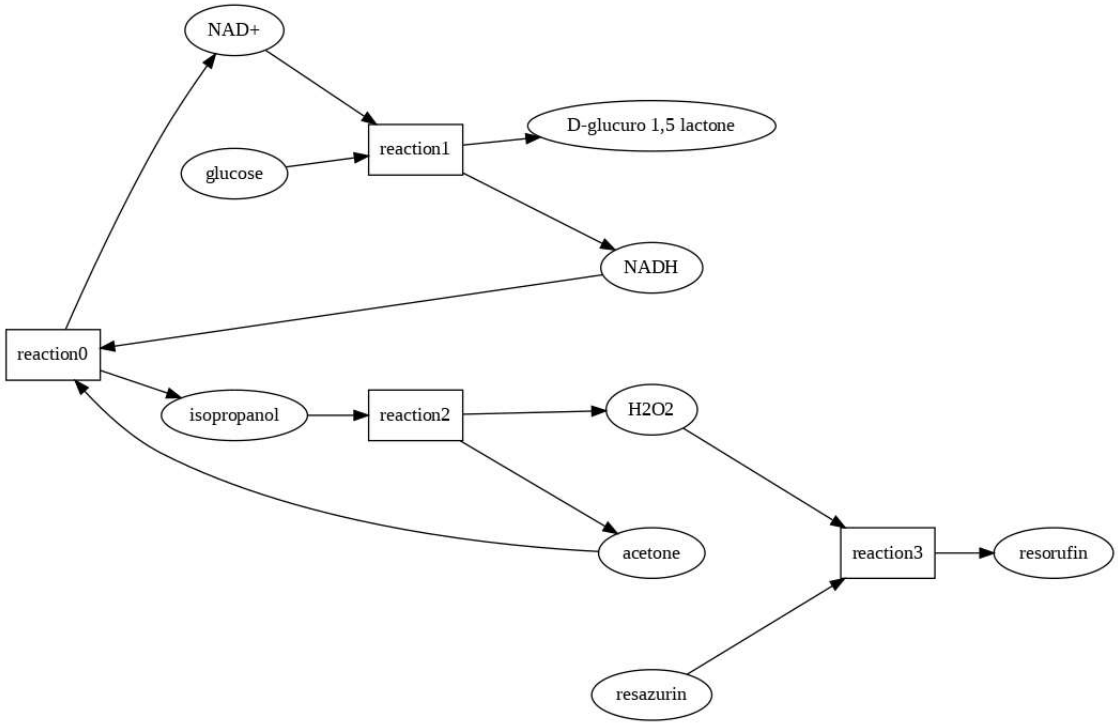
[14]:
```
new_model(CRNlong).
b+c=>d+e.
a+d=>c+g.
g=>a+h.
k=>j.
h+i=>k.
export_biocham(CRNlong.bc).
list_model.
draw_reactions.
```



[14]:
```
MA(1) for b+c=>d+e.
MA(1) for a+d=>c+g.
MA(1) for g=>a+h.
MA(1) for k=>j.
MA(1) for h+i=>k.
```

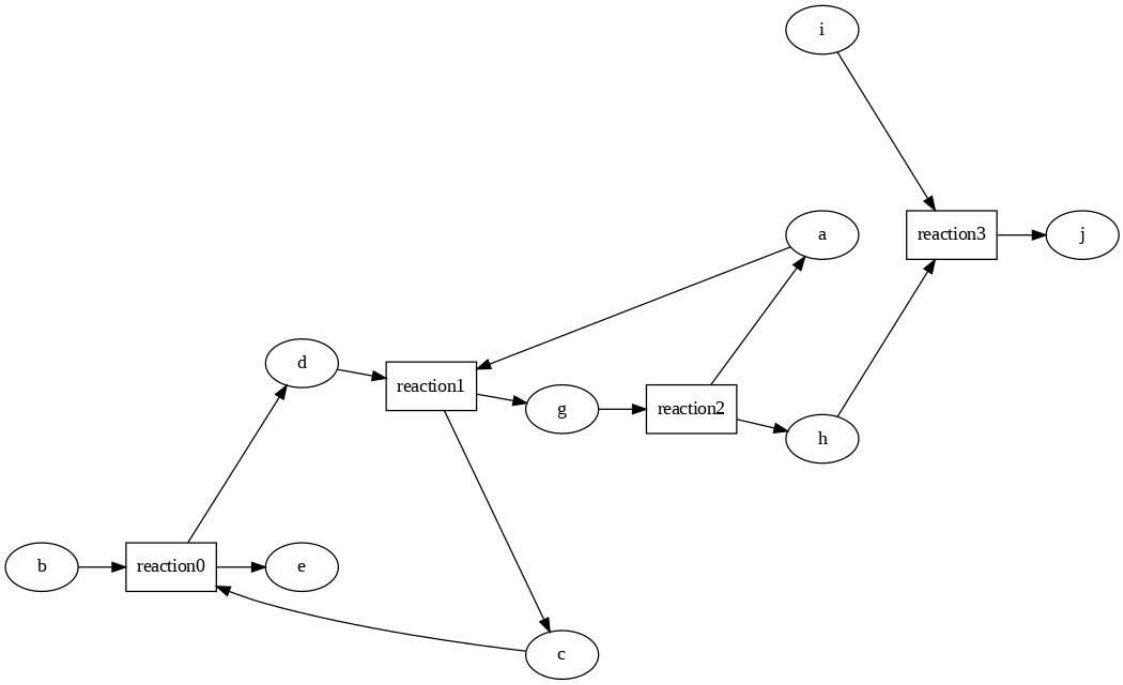
[1]:
```
new_model(CRNAlexis).
NADH+acetone=>'NAD+'+isopropanol.
'NAD+'+glucose=> 'D-glucuro 1,5 lactone' + NADH.
isopropanol=>H2O2+acetone.
H2O2+resazurin=>resorufin.
```

```
export_biocham(CRNAlexis.bc).
list_model.
draw_reactions.
```




```
[1]: MA(1) for NADH+acetone=>'NAD+'+isopropanol.
     MA(1) for 'NAD+'+glucose=>'D-glucuro 1,5 lactone'+NADH.
     MA(1) for isopropanol=>H2O2+acetone.
     MA(1) for H2O2+resazurin=>resorufin.
```

```
[2]: new_model(CRNshort).
     b+c => d+e.
     a+d => c+g.
     g => a+h.
     h+i => j.
     export_biocham(CRNshort.bc).
     list_model.
     draw_reactions.
```

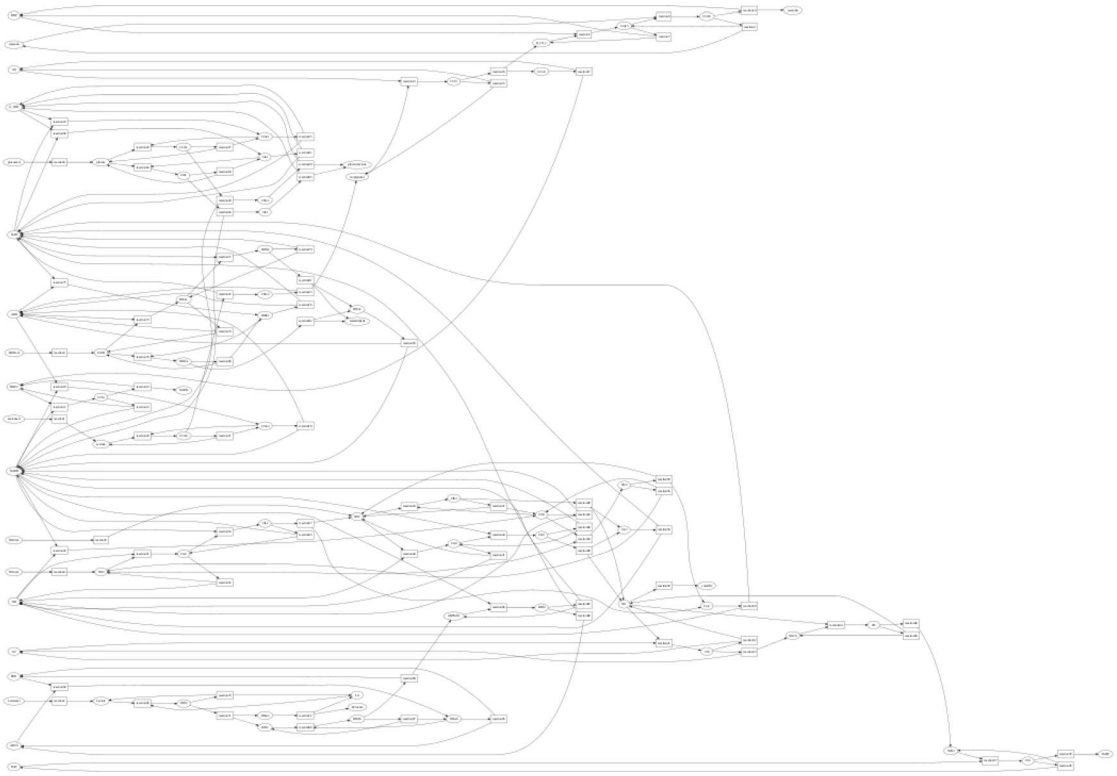


[2]:
```
MA(1) for b+c=>d+e.
MA(1) for a+d=>c+g.
MA(1) for g=>a+h.
MA(1) for h+i=>j.
```

[55]:
```
% list_models.
```

We'll be using the following subset of reactions for the rest of the notebook :

[3]:
```
clear_model.
load(library:examples/modalINF473L/catalog.bc).
export_biocham(Catalog.bc).
draw_reactions.
```



3 2. First SEPI examples on basic CRNs to understand functionnalities

Another example, this time a SEPI with a simple contraction :

[4]:
```
search_reduction(CRNlong.bc, CRNAlexis.bc, mapping_restriction : [a->acetone,␣
↪b->glucose, j->resorufin], merge_restriction : not_species).
```

[4]:
```
sepi
b -> glucose
c -> NAD+
d -> NADH
e -> D-glucuro 1,5 lactone
a -> acetone
g -> isopropanol
h -> H2O2
k -> deleted
j -> resorufin
i -> resazurin
{b+c => d+e} -> {NAD++glucose => D-glucuro 1,5 lactone+NADH}
{a+d => c+g} -> {NADH+acetone => NAD++isopropanol}
{g => a+h} -> {isopropanol => H2O2+acetone}
```

```
{k => j} -> {H2O2+resazurin => resorufin}
{h+i => k} -> {H2O2+resazurin => resorufin}
Number of reductions: 1
```

In the same fashion, we can also see that with the bigger CRNlong :

[30]:
```
search_reduction(CRNlong.bc, CRNVeryShort.bc, mapping_restriction :␣
↪[a->acetone, b->glucose, j->resorufin], merge_restriction : not_species).
```

[30]:
```
sepi
b -> glucose
c -> deleted
d -> deleted
e -> deleted
a -> acetone
g -> deleted
h -> deleted
i -> deleted
k -> deleted
j -> resorufin
{b+c => d+e} -> {acetone+glucose => resorufin}
{a+d => c+g} -> {acetone+glucose => resorufin}
{g => a+h} -> deleted
{h+i => k} -> deleted
{k => j} -> {acetone+glucose => resorufin}
Number of reductions: 1
```

This cell's goal is to show that Alexis' CRN is indeed a SEPI of the VeryShort, meaning it is a solution to it's concretization

[5]:
```
search_reduction(CRNAlexis.bc, CRNVeryShort.bc, mapping_restriction :␣
↪[acetone->acetone, glucose->glucose, resorufin->resorufin],␣
↪merge_restriction : not_species).
```

[5]:
```
sepi
NADH -> deleted
acetone -> acetone
NAD+ -> deleted
isopropanol -> deleted
glucose -> glucose
D-glucuro 1,5 lactone -> deleted
H2O2 -> deleted
resazurin -> deleted
resorufin -> resorufin
{NADH+acetone => NAD++isopropanol} -> {acetone+glucose => resorufin}
{NAD++glucose => D-glucuro 1,5 lactone+NADH} -> {acetone+glucose => resorufin}
{isopropanol => H2O2+acetone} -> deleted
{H2O2+resazurin => resorufin} -> {acetone+glucose => resorufin}
```

```
Number of reductions: 1
```

4 3. Implementation of these principles to search real CRNs in the catalog

Below : SEPI search of a concretization isomorphic to Alexis'

[6]:
```
search_reduction(Catalog.bc, CRNshort.bc, mapping_restriction : [acetone->a,␣
↪glucose->b, resorufin->j], merge_restriction : not_species).
```

[6]:
```
sepi
glucoseext -> deleted
glucose -> b
acetoneext -> deleted
acetone -> a
Lactateext -> deleted
Lactate -> e
EtOHext -> deleted
EtOH -> i
NO3ext -> deleted
NO3 -> deleted
NO2ext -> deleted
NO2 -> deleted
HRP -> deleted
H_2O_2 -> deleted
CCia5 -> deleted
resazurin -> deleted
CCib5 -> deleted
resorufin -> j
HRP2 -> deleted
NADH -> deleted
CCf4 -> deleted
NADN -> deleted
AO -> deleted
isopropanol -> deleted
CCf3 -> deleted
CCio3 -> deleted
ADH -> h
CCia2 -> deleted
CCib2 -> deleted
CCfa2 -> deleted
NAD -> deleted
G_1DH -> deleted
CCia1 -> deleted
CCib1 -> deleted
CCfa1 -> deleted
```

```
gluconolacrone -> deleted
NO -> deleted
volatNO -> deleted
O2 -> deleted
Cf6 -> deleted
NO2b -> deleted
Cf5 -> deleted
N2O3 -> deleted
DAF -> deleted
Cf4 -> deleted
DAFF -> deleted
NR -> deleted
Cia3 -> deleted
Cib3 -> deleted
Cfa3 -> deleted
Cfb3 -> deleted
Cio3 -> deleted
Cia2 -> deleted
Cfa2 -> deleted
Cfb2 -> deleted
Cio2 -> deleted
Cia1 -> deleted
Cib1 -> deleted
Cfa1 -> deleted
ABTSOX -> g
DDf3 -> deleted
ABTS -> deleted
LO -> deleted
DDf2 -> deleted
DDio2 -> deleted
H2O2 -> deleted
Pyruvate -> deleted
DDia1 -> deleted
DDib1 -> deleted
DDfa1 -> deleted
DDfb1 -> deleted
DDio1 -> deleted
acetaldehyde -> deleted
POD -> c
DDia5 -> deleted
DDib5 -> d
{glucoseext => glucose} -> deleted
{acetoneext => acetone} -> deleted
{Lactateext => Lactate} -> deleted
{EtOHext => EtOH} -> deleted
{NO3ext => NO3} -> deleted
{NO2ext => NO2} -> {a+d => c+g}
```

{HRP+H_2O_2 => CCia5} -> deleted
{CCia5 => HRP+H_2O_2} -> deleted
{CCia5+resazurin => CCib5} -> {h+i => j}
{CCib5 => CCia5+resazurin} -> {b+c => d+e}
{CCib5 => HRP+resorufin} -> {h+i => j}
{HRP2+NADH => CCf4} -> deleted
{CCf4 => HRP2+NADH} -> deleted
{CCf4 => HRP2+NADN} -> deleted
{AO+isopropanol => CCf3} -> deleted
{CCf3 => AO+isopropanol} -> {a+d => c+g}
{CCf3 => CCio3+H_2O_2} -> deleted
{CCio3 => AO+HRP2} -> deleted
{ADH+NADH => CCia2} -> deleted
{CCia2 => ADH+NADH} -> deleted
{CCia2+acetone => CCib2} -> {a+d => c+g}
{CCib2 => CCia2+acetone} -> {g => a+h}
{CCib2 => CCfa2+NAD} -> deleted
{CCfa2 => ADH+isopropanol} -> deleted
{G_1DH+NAD => CCia1} -> deleted
{CCia1 => G_1DH+NAD} -> {h+i => j}
{CCia1+glucose => CCib1} -> deleted
{CCib1 => CCia1+glucose} -> deleted
{CCib1 => CCfa1+NADH} -> deleted
{CCfa1 => G_1DH+gluconolacrone} -> deleted
{NO => volatNO} -> deleted
{NO+O2 => Cf6} -> deleted
{Cf6 => NO+O2} -> deleted
{Cf6 => NO2b+O2} -> deleted
{NO+NO2b => Cf5} -> deleted
{Cf5 => NO+NO2b} -> deleted
{Cf5 => N2O3} -> deleted
{DAF+N2O3 => Cf4} -> deleted
{Cf4 => DAF+N2O3} -> deleted
{Cf4 => DAFF} -> deleted
{NO2+NR => Cia3} -> deleted
{Cia3 => NO2+NR} -> deleted
{NADH+NR => Cib3} -> deleted
{Cib3 => NADH+NR} -> deleted
{Cia3+NADH => Cfa3} -> deleted
{Cib3+NO2 => Cfb3} -> {h+i => j}
{Cfa3 => Cia3+NADH} -> deleted
{Cfb3 => Cib3+NO2} -> deleted
{Cfa3 => Cio3+NO} -> deleted
{Cfb3 => Cio3+NO} -> deleted
{Cio3 => NAD+NR} -> deleted
{NO3+NR => Cia2} -> deleted
{Cia2 => NO3+NR} -> deleted

```
{Cia2+NADH => Cfa2} -> deleted
{Cib3+NO3 => Cfb2} -> deleted
{Cfa2 => Cia2+NADH} -> deleted
{Cfb2 => Cib3+NO3} -> deleted
{Cfa2 => Cio2+NO2} -> deleted
{Cfb2 => Cio2+NO2} -> deleted
{Cio2 => NAD+NR} -> deleted
{G_1DH+NAD => Cia1} -> deleted
{Cia1 => G_1DH+NAD} -> deleted
{Cia1+glucose => Cib1} -> {b+c => d+e}
{Cib1 => Cia1+glucose} -> deleted
{Cib1 => Cfa1+NADH} -> {b+c => d+e}
{Cfa1 => G_1DH+gluconolacrone} -> deleted
{ABTSOX+NADH => DDf3} -> {g => a+h}
{DDf3 => ABTSOX+NADH} -> deleted
{DDf3 => ABTS+NAD} -> {b+c => d+e}
{LO+Lactate => DDf2} -> deleted
{DDf2 => LO+Lactate} -> {b+c => d+e}
{DDf2 => DDio2+H2O2} -> deleted
{DDio2 => LO+Pyruvate} -> deleted
{ADH+EtOH => DDia1} -> deleted
{DDia1 => ADH+EtOH} -> deleted
{ADH+NAD => DDib1} -> {h+i => j}
{DDib1 => ADH+NAD} -> {g => a+h}
{DDia1+NAD => DDfa1} -> deleted
{DDib1+EtOH => DDfb1} -> {h+i => j}
{DDfa1 => DDia1+NAD} -> deleted
{DDfb1 => DDib1+EtOH} -> deleted
{DDfa1 => DDio1+acetaldehyde} -> deleted
{DDfb1 => DDio1+acetaldehyde} -> {g => a+h}
{DDio1 => ADH+NADH} -> deleted
{ABTS+POD => DDia5} -> {b+c => d+e}
{DDia5 => ABTS+POD} -> deleted
{DDia5+H2O2 => DDib5} -> {b+c => d+e}
{DDib5 => DDia5+H2O2} -> deleted
{DDib5 => ABTSOX+POD} -> {a+d => c+g}
Number of reductions: 1
```

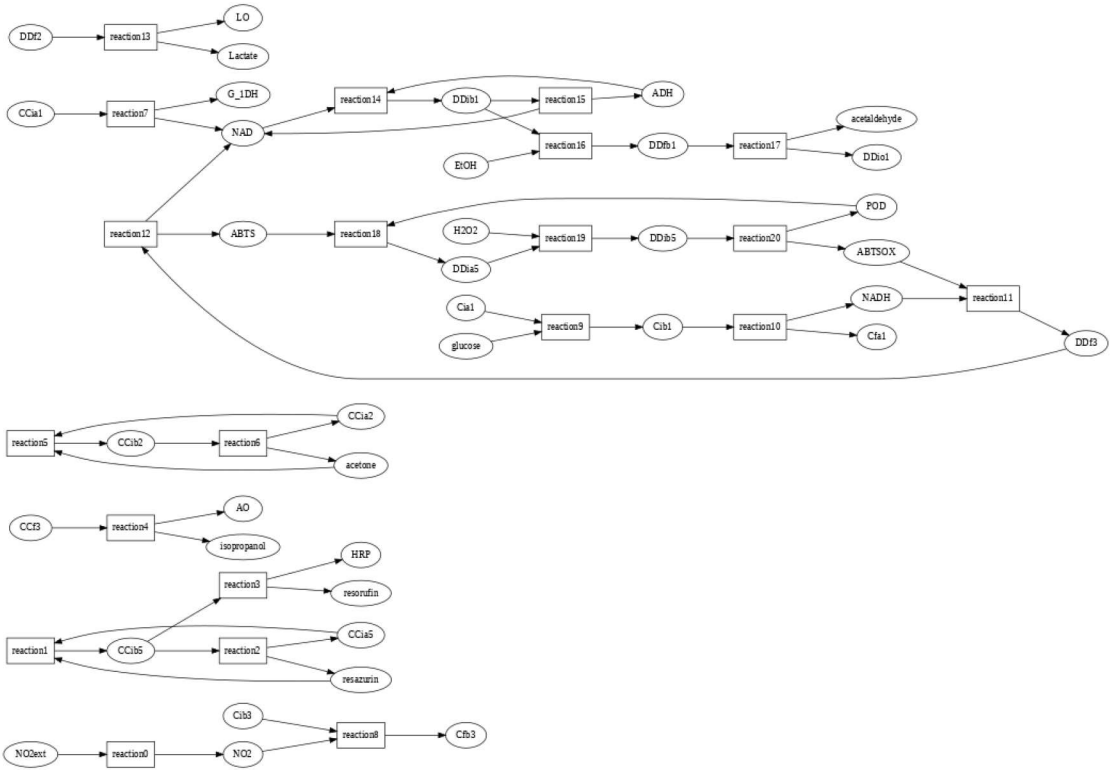
[7]:
```
search_reduction(Catalog.bc, CRNshort.bc, mapping_restriction : [acetone->a,␣
↪glucose->b, resorufin->j], merge_restriction : not_species, show_support :␣
↪yes).
```

[7]:
```
sepi
glucose
acetone
Lactate
EtOH
```

```
resorufin
ADH
ABTSOX
POD
DDib5
NO2ext => NO2
CCia5+resazurin => CCib5
CCib5 => CCia5+resazurin
CCib5 => HRP+resorufin
CCf3 => AO+isopropanol
CCia2+acetone => CCib2
CCib2 => CCia2+acetone
CCia1 => G_1DH+NAD
Cib3+NO2 => Cfb3
Cia1+glucose => Cib1
Cib1 => Cfa1+NADH
ABTSOX+NADH => DDf3
DDf3 => ABTS+NAD
DDf2 => LO+Lactate
ADH+NAD => DDib1
DDib1 => ADH+NAD
DDib1+EtOH => DDfb1
DDfb1 => DDio1+acetaldehyde
ABTS+POD => DDia5
DDia5+H2O2 => DDib5
DDib5 => ABTSOX+POD
Number of reductions: 1
```

[8]:
```
new_model.
NO2ext => NO2.
CCia5+resazurin => CCib5.
CCib5 => CCia5+resazurin.
CCib5 => HRP+resorufin.
CCf3 => AO+isopropanol.
CCia2+acetone => CCib2.
CCib2 => CCia2+acetone.
CCia1 => G_1DH+NAD.
Cib3+NO2 => Cfb3.
Cia1+glucose => Cib1.
Cib1 => Cfa1+NADH.
ABTSOX+NADH => DDf3.
DDf3 => ABTS+NAD.
DDf2 => LO+Lactate.
ADH+NAD => DDib1.
DDib1 => ADH+NAD.
DDib1+EtOH => DDfb1.
DDfb1 => DDio1+acetaldehyde.
```

```
ABTS+POD => DDia5.
DDia5+H2O2 => DDib5.
DDib5 => ABTSOX+POD.
draw_reactions.
```



Same search but with an additional step before resorufin (analoguous to the solutions found with the SISO* heuristic algorithms).

[9]:
```
search_reduction(Catalog.bc, CRNlong.bc, mapping_restriction : [acetone->a,␣
↪glucose->b, resorufin->j], merge_restriction : not_species).
```

[9]:
```
sepi
glucoseext -> deleted
glucose -> b
acetoneext -> deleted
acetone -> a
Lactateext -> deleted
Lactate -> deleted
EtOHext -> deleted
EtOH -> i
NO3ext -> deleted
NO3 -> deleted
NO2ext -> deleted
```

```
NO2 -> deleted
HRP -> deleted
H_2O_2 -> h
CCia5 -> deleted
resazurin -> deleted
CCib5 -> deleted
resorufin -> j
HRP2 -> deleted
NADH -> k
CCf4 -> deleted
NADN -> deleted
AO -> deleted
isopropanol -> deleted
CCf3 -> deleted
CCio3 -> deleted
ADH -> deleted
CCia2 -> deleted
CCib2 -> deleted
CCfa2 -> deleted
NAD -> deleted
G_1DH -> deleted
CCia1 -> g
CCib1 -> deleted
CCfa1 -> deleted
gluconolacrone -> deleted
NO -> c
volatNO -> deleted
O2 -> deleted
Cf6 -> deleted
NO2b -> deleted
Cf5 -> deleted
N2O3 -> deleted
DAF -> deleted
Cf4 -> deleted
DAFF -> deleted
NR -> deleted
Cia3 -> deleted
Cib3 -> deleted
Cfa3 -> deleted
Cfb3 -> deleted
Cio3 -> deleted
Cia2 -> deleted
Cfa2 -> deleted
Cfb2 -> deleted
Cio2 -> deleted
Cia1 -> deleted
Cib1 -> deleted
```

```
Cfa1 -> deleted
ABTSOX -> e
DDf3 -> deleted
ABTS -> deleted
LO -> deleted
DDf2 -> deleted
DDio2 -> deleted
H2O2 -> deleted
Pyruvate -> deleted
DDia1 -> deleted
DDib1 -> deleted
DDfa1 -> deleted
DDfb1 -> deleted
DDio1 -> deleted
acetaldehyde -> deleted
POD -> d
DDia5 -> deleted
DDib5 -> deleted
{glucoseext => glucose} -> deleted
{acetoneext => acetone} -> deleted
{Lactateext => Lactate} -> {h+i => k}
{EtOHext => EtOH} -> deleted
{NO3ext => NO3} -> deleted
{NO2ext => NO2} -> deleted
{HRP+H_2O_2 => CCia5} -> {h+i => k}
{CCia5 => HRP+H_2O_2} -> {g => a+h}
{CCia5+resazurin => CCib5} -> {h+i => k}
{CCib5 => CCia5+resazurin} -> {h+i => k}
{CCib5 => HRP+resorufin} -> {k => j}
{HRP2+NADH => CCf4} -> {k => j}
{CCf4 => HRP2+NADH} -> deleted
{CCf4 => HRP2+NADN} -> deleted
{AO+isopropanol => CCf3} -> {g => a+h}
{CCf3 => AO+isopropanol} -> {h+i => k}
{CCf3 => CCio3+H_2O_2} -> deleted
{CCio3 => AO+HRP2} -> deleted
{ADH+NADH => CCia2} -> deleted
{CCia2 => ADH+NADH} -> deleted
{CCia2+acetone => CCib2} -> {a+d => c+g}
{CCib2 => CCia2+acetone} -> {g => a+h}
{CCib2 => CCfa2+NAD} -> deleted
{CCfa2 => ADH+isopropanol} -> deleted
{G_1DH+NAD => CCia1} -> {a+d => c+g}
{CCia1 => G_1DH+NAD} -> {g => a+h}
{CCia1+glucose => CCib1} -> deleted
{CCib1 => CCia1+glucose} -> deleted
{CCib1 => CCfa1+NADH} -> deleted
```

{CCfa1 => G_1DH+gluconolacrone} -> deleted
{NO => volatNO} -> deleted
{NO+O2 => Cf6} -> deleted
{Cf6 => NO+O2} -> {a+d => c+g}
{Cf6 => NO2b+O2} -> deleted
{NO+NO2b => Cf5} -> {b+c => d+e}
{Cf5 => NO+NO2b} -> deleted
{Cf5 => N2O3} -> deleted
{DAF+N2O3 => Cf4} -> deleted
{Cf4 => DAF+N2O3} -> deleted
{Cf4 => DAFF} -> deleted
{NO2+NR => Cia3} -> deleted
{Cia3 => NO2+NR} -> deleted
{NADH+NR => Cib3} -> deleted
{Cib3 => NADH+NR} -> deleted
{Cia3+NADH => Cfa3} -> deleted
{Cib3+NO2 => Cfb3} -> deleted
{Cfa3 => Cia3+NADH} -> deleted
{Cfb3 => Cib3+NO2} -> deleted
{Cfa3 => Cio3+NO} -> deleted
{Cfb3 => Cio3+NO} -> deleted
{Cio3 => NAD+NR} -> deleted
{NO3+NR => Cia2} -> deleted
{Cia2 => NO3+NR} -> deleted
{Cia2+NADH => Cfa2} -> deleted
{Cib3+NO3 => Cfb2} -> deleted
{Cfa2 => Cia2+NADH} -> deleted
{Cfb2 => Cib3+NO3} -> deleted
{Cfa2 => Cio2+NO2} -> deleted
{Cfb2 => Cio2+NO2} -> deleted
{Cio2 => NAD+NR} -> deleted
{G_1DH+NAD => Cia1} -> deleted
{Cia1 => G_1DH+NAD} -> deleted
{Cia1+glucose => Cib1} -> {b+c => d+e}
{Cib1 => Cia1+glucose} -> deleted
{Cib1 => Cfa1+NADH} -> deleted
{Cfa1 => G_1DH+gluconolacrone} -> deleted
{ABTSOX+NADH => DDf3} -> deleted
{DDf3 => ABTSOX+NADH} -> deleted
{DDf3 => ABTS+NAD} -> deleted
{LO+Lactate => DDf2} -> deleted
{DDf2 => LO+Lactate} -> deleted
{DDf2 => DDio2+H2O2} -> deleted
{DDio2 => LO+Pyruvate} -> deleted
{ADH+EtOH => DDia1} -> deleted
{DDia1 => ADH+EtOH} -> deleted
{ADH+NAD => DDib1} -> deleted

```
{DDib1 => ADH+NAD} -> deleted
{DDia1+NAD => DDfa1} -> deleted
{DDib1+EtOH => DDfb1} -> {h+i => k}
{DDfa1 => DDia1+NAD} -> deleted
{DDfb1 => DDib1+EtOH} -> deleted
{DDfa1 => DDio1+acetaldehyde} -> deleted
{DDfb1 => DDio1+acetaldehyde} -> deleted
{DDio1 => ADH+NADH} -> {h+i => k}
{ABTS+POD => DDia5} -> {a+d => c+g}
{DDia5 => ABTS+POD} -> deleted
{DDia5+H2O2 => DDib5} -> deleted
{DDib5 => DDia5+H2O2} -> deleted
{DDib5 => ABTSOX+POD} -> {b+c => d+e}
Number of reductions: 1
```

SEPI search to find a regular solution to the acetone∧ glucose = resorufin CRN

[10]:
```
search_reduction(Catalog.bc, CRNVeryShort.bc, mapping_restriction :␣
↪[acetone->acetone, glucose->glucose, resorufin->resorufin],␣
↪merge_restriction : not_species).
```

[10]:
```
sepi
glucoseext -> deleted
glucose -> glucose
acetoneext -> deleted
acetone -> acetone
Lactateext -> deleted
Lactate -> deleted
EtOHext -> deleted
EtOH -> deleted
NO3ext -> deleted
NO3 -> deleted
NO2ext -> deleted
NO2 -> deleted
HRP -> deleted
H_2O_2 -> deleted
CCia5 -> deleted
resazurin -> deleted
CCib5 -> deleted
resorufin -> resorufin
HRP2 -> deleted
NADH -> deleted
CCf4 -> deleted
NADN -> deleted
AO -> deleted
isopropanol -> deleted
CCf3 -> deleted
```

```
CCio3 -> deleted
ADH -> deleted
CCia2 -> deleted
CCib2 -> deleted
CCfa2 -> deleted
NAD -> deleted
G_1DH -> deleted
CCia1 -> deleted
CCib1 -> deleted
CCfa1 -> deleted
gluconolacrone -> deleted
NO -> deleted
volatNO -> deleted
O2 -> deleted
Cf6 -> deleted
NO2b -> deleted
Cf5 -> deleted
N2O3 -> deleted
DAF -> deleted
Cf4 -> deleted
DAFF -> deleted
NR -> deleted
Cia3 -> deleted
Cib3 -> deleted
Cfa3 -> deleted
Cfb3 -> deleted
Cio3 -> deleted
Cia2 -> deleted
Cfa2 -> deleted
Cfb2 -> deleted
Cio2 -> deleted
Cia1 -> deleted
Cib1 -> deleted
Cfa1 -> deleted
ABTSOX -> deleted
DDf3 -> deleted
ABTS -> deleted
LO -> deleted
DDf2 -> deleted
DDio2 -> deleted
H2O2 -> deleted
Pyruvate -> deleted
DDia1 -> deleted
DDib1 -> deleted
DDfa1 -> deleted
DDfb1 -> deleted
DDio1 -> deleted
```

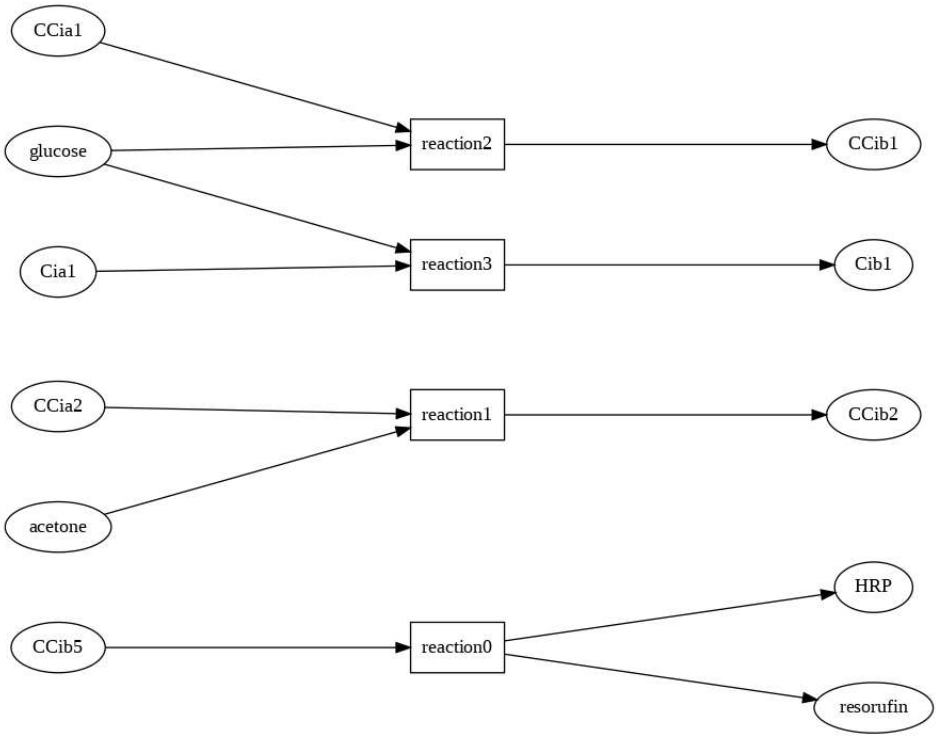
```
acetaldehyde -> deleted
POD -> deleted
DDia5 -> deleted
DDib5 -> deleted
{glucoseext => glucose} -> deleted
{acetoneext => acetone} -> deleted
{Lactateext => Lactate} -> deleted
{EtOHext => EtOH} -> deleted
{NO3ext => NO3} -> deleted
{NO2ext => NO2} -> deleted
{HRP+H_2O_2 => CCia5} -> deleted
{CCia5 => HRP+H_2O_2} -> deleted
{CCia5+resazurin => CCib5} -> deleted
{CCib5 => CCia5+resazurin} -> deleted
{CCib5 => HRP+resorufin} -> {acetone+glucose => resorufin}
{HRP2+NADH => CCf4} -> deleted
{CCf4 => HRP2+NADH} -> deleted
{CCf4 => HRP2+NADN} -> deleted
{AO+isopropanol => CCf3} -> deleted
{CCf3 => AO+isopropanol} -> deleted
{CCf3 => CCio3+H_2O_2} -> deleted
{CCio3 => AO+HRP2} -> deleted
{ADH+NADH => CCia2} -> deleted
{CCia2 => ADH+NADH} -> deleted
{CCia2+acetone => CCib2} -> {acetone+glucose => resorufin}
{CCib2 => CCia2+acetone} -> deleted
{CCib2 => CCfa2+NAD} -> deleted
{CCfa2 => ADH+isopropanol} -> deleted
{G_1DH+NAD => CCia1} -> deleted
{CCia1 => G_1DH+NAD} -> deleted
{CCia1+glucose => CCib1} -> {acetone+glucose => resorufin}
{CCib1 => CCia1+glucose} -> deleted
{CCib1 => CCfa1+NADH} -> deleted
{CCfa1 => G_1DH+gluconolacrone} -> deleted
{NO => volatNO} -> deleted
{NO+O2 => Cf6} -> deleted
{Cf6 => NO+O2} -> deleted
{Cf6 => NO2b+O2} -> deleted
{NO+NO2b => Cf5} -> deleted
{Cf5 => NO+NO2b} -> deleted
{Cf5 => N2O3} -> deleted
{DAF+N2O3 => Cf4} -> deleted
{Cf4 => DAF+N2O3} -> deleted
{Cf4 => DAFF} -> deleted
{NO2+NR => Cia3} -> deleted
{Cia3 => NO2+NR} -> deleted
{NADH+NR => Cib3} -> deleted
```

```
{Cib3 => NADH+NR} -> deleted
{Cia3+NADH => Cfa3} -> deleted
{Cib3+NO2 => Cfb3} -> deleted
{Cfa3 => Cia3+NADH} -> deleted
{Cfb3 => Cib3+NO2} -> deleted
{Cfa3 => Cio3+NO} -> deleted
{Cfb3 => Cio3+NO} -> deleted
{Cio3 => NAD+NR} -> deleted
{NO3+NR => Cia2} -> deleted
{Cia2 => NO3+NR} -> deleted
{Cia2+NADH => Cfa2} -> deleted
{Cib3+NO3 => Cfb2} -> deleted
{Cfa2 => Cia2+NADH} -> deleted
{Cfb2 => Cib3+NO3} -> deleted
{Cfa2 => Cio2+NO2} -> deleted
{Cfb2 => Cio2+NO2} -> deleted
{Cio2 => NAD+NR} -> deleted
{G_1DH+NAD => Cia1} -> deleted
{Cia1 => G_1DH+NAD} -> deleted
{Cia1+glucose => Cib1} -> {acetone+glucose => resorufin}
{Cib1 => Cia1+glucose} -> deleted
{Cib1 => Cfa1+NADH} -> deleted
{Cfa1 => G_1DH+gluconolacrone} -> deleted
{ABTSOX+NADH => DDf3} -> deleted
{DDf3 => ABTSOX+NADH} -> deleted
{DDf3 => ABTS+NAD} -> deleted
{LO+Lactate => DDf2} -> deleted
{DDf2 => LO+Lactate} -> deleted
{DDf2 => DDio2+H2O2} -> deleted
{DDio2 => LO+Pyruvate} -> deleted
{ADH+EtOH => DDia1} -> deleted
{DDia1 => ADH+EtOH} -> deleted
{ADH+NAD => DDib1} -> deleted
{DDib1 => ADH+NAD} -> deleted
{DDia1+NAD => DDfa1} -> deleted
{DDib1+EtOH => DDfb1} -> deleted
{DDfa1 => DDia1+NAD} -> deleted
{DDfb1 => DDib1+EtOH} -> deleted
{DDfa1 => DDio1+acetaldehyde} -> deleted
{DDfb1 => DDio1+acetaldehyde} -> deleted
{DDio1 => ADH+NADH} -> deleted
{ABTS+POD => DDia5} -> deleted
{DDia5 => ABTS+POD} -> deleted
{DDia5+H2O2 => DDib5} -> deleted
{DDib5 => DDia5+H2O2} -> deleted
{DDib5 => ABTSOX+POD} -> deleted
Number of reductions: 1
```

[11]:
```
search_reduction(Catalog.bc, CRNVeryShort.bc, mapping_restriction :␣
↪[acetone->acetone, glucose->glucose, resorufin->resorufin],␣
↪merge_restriction : not_species, show_support : yes).
```

[11]:
```
sepi
glucose
acetone
resorufin
CCib5 => HRP+resorufin
CCia2+acetone => CCib2
CCia1+glucose => CCib1
Cia1+glucose => Cib1
Number of reductions: 1
```

[12]:
```
new_model.
CCib5 => HRP+resorufin.
CCia2+acetone => CCib2.
CCia1+glucose => CCib1.
Cia1+glucose => Cib1.
draw_reactions.
```



AVOID USING, former usage but network is too large Smart1Subset is made out of every reaction with an enzyme involved in the first solution heuristically found, and at least one metabolite in it

[32]:
```
clear_model.
load(library:examples/sepi/BrendaBiochamReactionsSmart1SubsetM.bc).
draw_reactions.
```

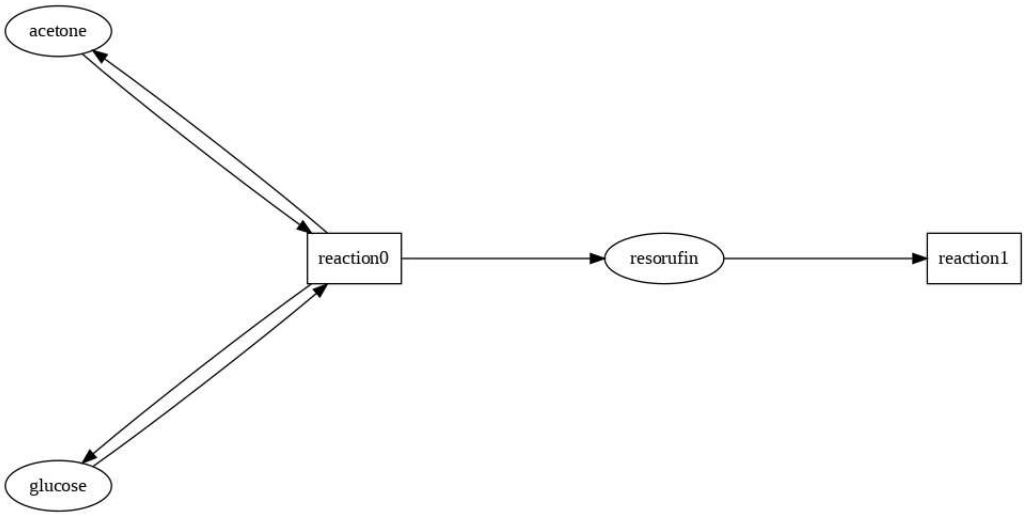
[ ]:
```
search_reduction(library:examples/sepi/BrendaBiochamReactionsSmart1SubsetM.bc,␣
↪CRNlong.bc, timeout : 10000000000000000000000000, stats : yes,␣
↪merge_restriction : not_species, mapping_restriction : [m286->a, m33->b,␣
↪m25014->j]).
```

5 4. Bigger examples of concretization : GluONe, LacOH and GluNOx

These three logical circuits can be read as : GluONe → Resorufin = glucose∧acetone et NADH = glucose ∧(1-acetone) LacOH → NADH = EtOH et ABTSOX = Lactate∧(1-EtOH) GluNOx → DAFF = glucose∧NOx et NADH = glucose∧(1-NOx) We will mostly focus on the none NADH branches.

5.1 GluONe

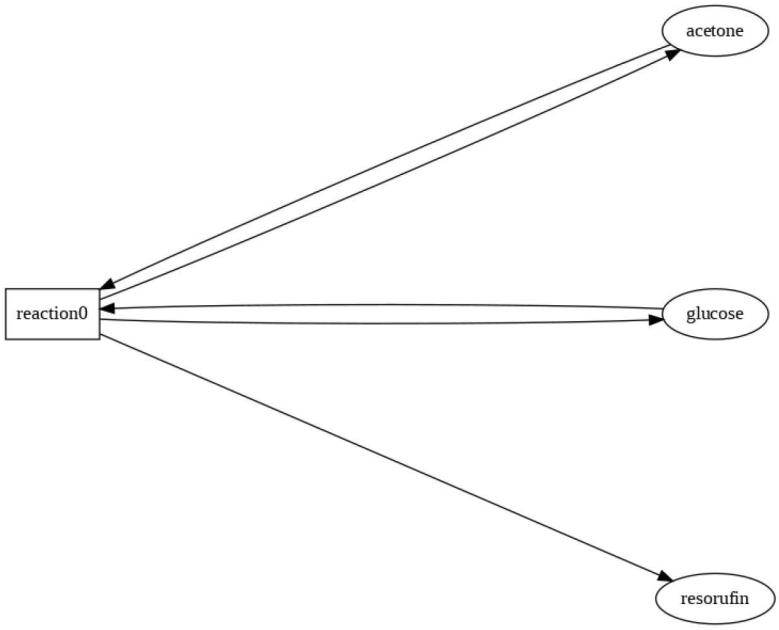
[23]:
```
clear_model.
%add_function( resorufin = (acetone * glucose), NADH = (glucose * (1 -␣
 ↪acetone))).
stabilize_expression(acetone*glucose-resorufin,resorufin,[acetone=1,glucose=1]).
%stabilize_expression(ABTSOX-lactate*(1-EtOH),ABTSOX,[lactate=1,EtOH=0,ABTSOX=1]).
 ↪
export_biocham(GluONe.bc).
list_model.
draw_reactions.
```



[23]:
```
MA(1.0) for acetone+glucose=>acetone+glucose+resorufin.
MA(1.0) for resorufin=>_.
initial_state(acetone=1).
initial_state(glucose=1).
```

A simplified aCRN without the resorufin→_ , as it leads to some issues :

[22]:
```
new_model(GluONe_Simple).
acetone+glucose=>acetone+glucose+resorufin.
export_biocham(GluONe_Simple.bc).
draw_reactions.
```



[24]:
```
search_reduction(Catalog.bc, GluONe_Simple.bc, mapping_restriction :␣
↪[acetone->acetone, glucose->glucose, resorufin->resorufin],␣
↪merge_restriction : not_species).
```

[24]:
```
sepi
glucoseext -> deleted
glucose -> glucose
acetoneext -> deleted
acetone -> acetone
Lactateext -> deleted
Lactate -> deleted
EtOHext -> deleted
EtOH -> deleted
NO3ext -> deleted
NO3 -> deleted
NO2ext -> deleted
NO2 -> deleted
HRP -> deleted
H_2O_2 -> deleted
CCia5 -> deleted
resazurin -> deleted
CCib5 -> deleted
resorufin -> resorufin
```

```
HRP2 -> deleted
NADH -> deleted
CCf4 -> deleted
NADN -> deleted
AO -> deleted
isopropanol -> deleted
CCf3 -> deleted
CCio3 -> deleted
ADH -> deleted
CCia2 -> deleted
CCib2 -> deleted
CCfa2 -> deleted
NAD -> deleted
G_1DH -> deleted
CCia1 -> deleted
CCib1 -> deleted
CCfa1 -> deleted
gluconolacrone -> deleted
NO -> deleted
volatNO -> deleted
O2 -> deleted
Cf6 -> deleted
NO2b -> deleted
Cf5 -> deleted
N2O3 -> deleted
DAF -> deleted
Cf4 -> deleted
DAFF -> deleted
NR -> deleted
Cia3 -> deleted
Cib3 -> deleted
Cfa3 -> deleted
Cfb3 -> deleted
Cio3 -> deleted
Cia2 -> deleted
Cfa2 -> deleted
Cfb2 -> deleted
Cio2 -> deleted
Cia1 -> deleted
Cib1 -> deleted
Cfa1 -> deleted
ABTSOX -> deleted
DDf3 -> deleted
ABTS -> deleted
LO -> deleted
DDf2 -> deleted
DDio2 -> deleted
```

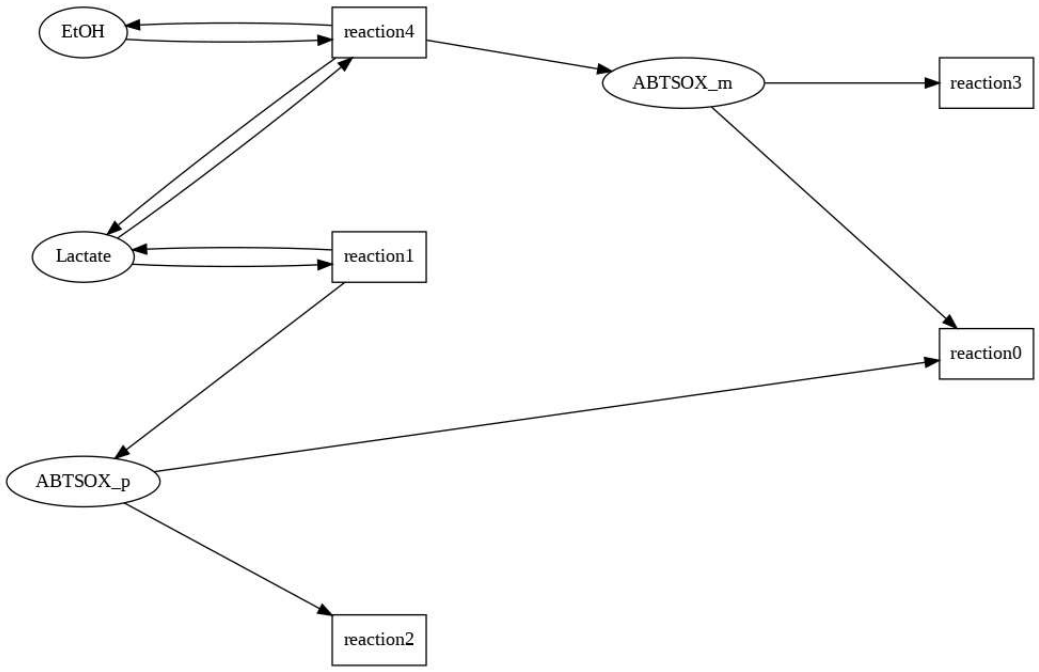
```
H2O2 -> deleted
Pyruvate -> deleted
DDia1 -> deleted
DDib1 -> deleted
DDfa1 -> deleted
DDfb1 -> deleted
DDio1 -> deleted
acetaldehyde -> deleted
POD -> deleted
DDia5 -> deleted
DDib5 -> deleted
{glucoseext => glucose} -> {acetone+glucose => acetone+glucose+resorufin}
{acetoneext => acetone} -> {acetone+glucose => acetone+glucose+resorufin}
{Lactateext => Lactate} -> deleted
{EtOHext => EtOH} -> deleted
{NO3ext => NO3} -> deleted
{NO2ext => NO2} -> deleted
{HRP+H_2O_2 => CCia5} -> deleted
{CCia5 => HRP+H_2O_2} -> deleted
{CCia5+resazurin => CCib5} -> deleted
{CCib5 => CCia5+resazurin} -> deleted
{CCib5 => HRP+resorufin} -> {acetone+glucose => acetone+glucose+resorufin}
{HRP2+NADH => CCf4} -> deleted
{CCf4 => HRP2+NADH} -> deleted
{CCf4 => HRP2+NADN} -> deleted
{AO+isopropanol => CCf3} -> deleted
{CCf3 => AO+isopropanol} -> deleted
{CCf3 => CCio3+H_2O_2} -> deleted
{CCio3 => AO+HRP2} -> deleted
{ADH+NADH => CCia2} -> deleted
{CCia2 => ADH+NADH} -> deleted
{CCia2+acetone => CCib2} -> {acetone+glucose => acetone+glucose+resorufin}
{CCib2 => CCia2+acetone} -> {acetone+glucose => acetone+glucose+resorufin}
{CCib2 => CCfa2+NAD} -> deleted
{CCfa2 => ADH+isopropanol} -> deleted
{G_1DH+NAD => CCia1} -> deleted
{CCia1 => G_1DH+NAD} -> deleted
{CCia1+glucose => CCib1} -> {acetone+glucose => acetone+glucose+resorufin}
{CCib1 => CCia1+glucose} -> {acetone+glucose => acetone+glucose+resorufin}
{CCib1 => CCfa1+NADH} -> deleted
{CCfa1 => G_1DH+gluconolacrone} -> deleted
{NO => volatNO} -> deleted
{NO+O2 => Cf6} -> deleted
{Cf6 => NO+O2} -> deleted
{Cf6 => NO2b+O2} -> deleted
{NO+NO2b => Cf5} -> deleted
{Cf5 => NO+NO2b} -> deleted
```

{Cf5 => N2O3} -> deleted
{DAF+N2O3 => Cf4} -> deleted
{Cf4 => DAF+N2O3} -> deleted
{Cf4 => DAFF} -> deleted
{NO2+NR => Cia3} -> deleted
{Cia3 => NO2+NR} -> deleted
{NADH+NR => Cib3} -> deleted
{Cib3 => NADH+NR} -> deleted
{Cia3+NADH => Cfa3} -> deleted
{Cib3+NO2 => Cfb3} -> deleted
{Cfa3 => Cia3+NADH} -> deleted
{Cfb3 => Cib3+NO2} -> deleted
{Cfa3 => Cio3+NO} -> deleted
{Cfb3 => Cio3+NO} -> deleted
{Cio3 => NAD+NR} -> deleted
{NO3+NR => Cia2} -> deleted
{Cia2 => NO3+NR} -> deleted
{Cia2+NADH => Cfa2} -> deleted
{Cib3+NO3 => Cfb2} -> deleted
{Cfa2 => Cia2+NADH} -> deleted
{Cfb2 => Cib3+NO3} -> deleted
{Cfa2 => Cio2+NO2} -> deleted
{Cfb2 => Cio2+NO2} -> deleted
{Cio2 => NAD+NR} -> deleted
{G_1DH+NAD => Cia1} -> deleted
{Cia1 => G_1DH+NAD} -> deleted
{Cia1+glucose => Cib1} -> {acetone+glucose => acetone+glucose+resorufin}
{Cib1 => Cia1+glucose} -> {acetone+glucose => acetone+glucose+resorufin}
{Cib1 => Cfa1+NADH} -> deleted
{Cfa1 => G_1DH+gluconolacrone} -> deleted
{ABTSOX+NADH => DDf3} -> deleted
{DDf3 => ABTSOX+NADH} -> deleted
{DDf3 => ABTS+NAD} -> deleted
{LO+Lactate => DDf2} -> deleted
{DDf2 => LO+Lactate} -> deleted
{DDf2 => DDio2+H2O2} -> deleted
{DDio2 => LO+Pyruvate} -> deleted
{ADH+EtOH => DDia1} -> deleted
{DDia1 => ADH+EtOH} -> deleted
{ADH+NAD => DDib1} -> deleted
{DDib1 => ADH+NAD} -> deleted
{DDia1+NAD => DDfa1} -> deleted
{DDib1+EtOH => DDfb1} -> deleted
{DDfa1 => DDia1+NAD} -> deleted
{DDfb1 => DDib1+EtOH} -> deleted
{DDfa1 => DDio1+acetaldehyde} -> deleted
{DDfb1 => DDio1+acetaldehyde} -> deleted

```
{DDio1 => ADH+NADH} -> deleted
{ABTS+POD => DDia5} -> deleted
{DDia5 => ABTS+POD} -> deleted
{DDia5+H2O2 => DDib5} -> deleted
{DDib5 => DDia5+H2O2} -> deleted
{DDib5 => ABTSOX+POD} -> deleted
Number of reductions: 1
```

5.2 LacOH

[15]:
```
clear_model.
%add_function(NADH = EtOH, ABTSOX = (1-EtOH)*Lactate).
%compile_function(NADH = EtOH, ABTSOX = (1-EtOH)*Lactate).
%stabilize_expression(NADH - EtOH, (1 - EtOH) * Lactate - ABTSOX, [NADH=1,
↪EtOH=1, ABTSOX=1, Lactate=1]).
%stabilize_expression(NADH - EtOH, NADH, [NADH=1,EtOH=1,Lactate=1]).
stabilize_expression(Lactate*(1-EtOH)-ABTSOX, ABTSOX, [Lactate=1, EtOH=1,
↪ABTSOX=1]).
export_biocham(LacOH.bc).
list_model.
draw_reactions.
```



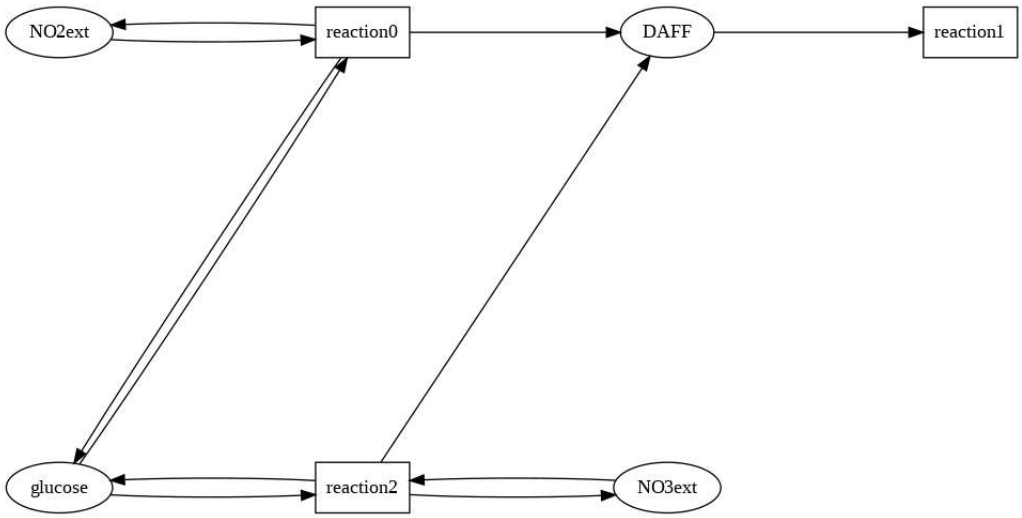
[15]:
```
MA(fast) for ABTSOX_m+ABTSOX_p=>_.
MA(1.0) for Lactate=>ABTSOX_p+Lactate.
MA(1.0) for ABTSOX_p=>_.
MA(1.0) for ABTSOX_m=>_.
MA(1.0) for EtOH+Lactate=>ABTSOX_m+EtOH+Lactate.
initial_state(ABTSOX_p=1).
initial_state(Lactate=1).
initial_state(EtOH=1).
parameter(
  fast = 1000
).
```

[16]:
```
search_reduction(Catalog.bc, LacOH.bc, mapping_restriction : [Lactate->Lactate,␣
↪EtOH->EtOH, ABTSOX->ABTSOX_p], merge_restriction : not_species).
```

[16]:
```
no sepi found
Number of reductions: 0
```

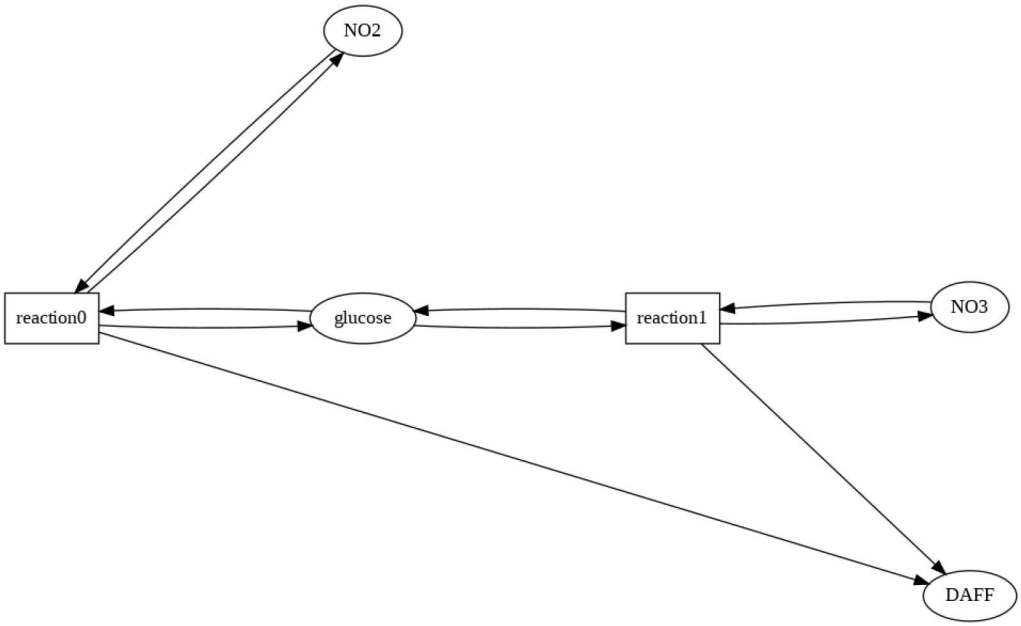
5.3 GluNOx

[29]:
```
clear_model.
%add_function( NADH = glucose*(1-NOx), DAFF = glucose*NOx).
stabilize_expression(DAFF-glucose*(NO2ext+NO3ext),DAFF,[glucose=1,NO2ext=1,NO3ext=0]).
↪
export_biocham(GluNOx.bc).
list_model.
draw_reactions.
```



[29]:
```
MA(1.0) for NO2ext+glucose=>DAFF+NO2ext+glucose.
MA(1.0) for DAFF=>_.
MA(1.0) for NO3ext+glucose=>DAFF+NO3ext+glucose.
initial_state(glucose=1).
initial_state(NO2ext=1).
```

[30]:
```
clear_model.
NO2+glucose=>DAFF+NO2+glucose.
NO3+glucose=>DAFF+NO3+glucose.
export_biocham(GluNOx_Simple.bc).
draw_reactions.
```

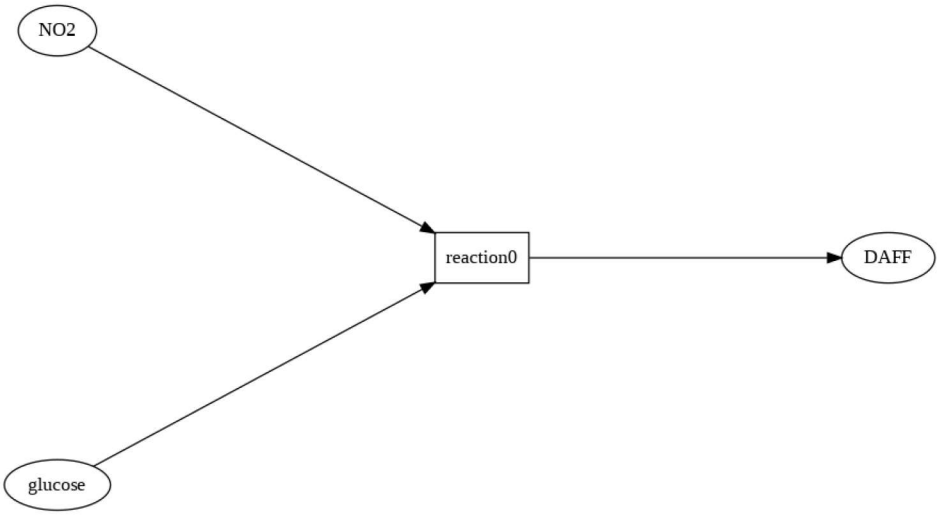


[33]:
```
search_reduction(Catalog.bc, GluNOx_Simple.bc, mapping_restriction :␣
↪[NO2->NO2,NO3->NO3, glucose->glucose, DAFF->DAFF], merge_restriction :␣
↪not_species).
```

[33]:
```
no sepi found
Number of reductions: 0
```

[ ]:
```
search_reduction(Catalog.bc, GluNOx.bc, mapping_restriction : [NO->NO,NO2->NO2,␣
↪glucose->glucose, DAFF->DAFF], merge_restriction : not_species).
```

[46]:
```
clear_model.
NO2 + glucose => DAFF.
%NO + glucose => DAFF.
%NO2 + glucose => DAFF + NO2 + glucose.
%DAFF=>_.
%NO3 + glucose => DAFF + NO3 + glucose.
export_biocham(CRNVeryShort2.bc).
draw_reactions.
```



[48]:
```
search_reduction(Catalog.bc, CRNVeryShort2.bc, mapping_restriction : [NO2->NO2,␣
↪glucose->glucose, DAFF->DAFF], merge_restriction : not_species).
```

[48]:
```
sepi
glucoseext -> deleted
glucose -> glucose
acetoneext -> deleted
acetone -> deleted
Lactateext -> deleted
Lactate -> deleted
EtOHext -> deleted
EtOH -> deleted
NO3ext -> deleted
NO3 -> deleted
NO2ext -> deleted
NO2 -> NO2
HRP -> deleted
H_2O_2 -> deleted
CCia5 -> deleted
resazurin -> deleted
CCib5 -> deleted
resorufin -> deleted
HRP2 -> deleted
```

```
NADH -> deleted
CCf4 -> deleted
NADN -> deleted
AO -> deleted
isopropanol -> deleted
CCf3 -> deleted
CCio3 -> deleted
ADH -> deleted
CCia2 -> deleted
CCib2 -> deleted
CCfa2 -> deleted
NAD -> deleted
G_1DH -> deleted
CCia1 -> deleted
CCib1 -> deleted
CCfa1 -> deleted
gluconolacrone -> deleted
NO -> deleted
volatNO -> deleted
O2 -> deleted
Cf6 -> deleted
NO2b -> deleted
Cf5 -> deleted
N2O3 -> deleted
DAF -> deleted
Cf4 -> deleted
DAFF -> DAFF
NR -> deleted
Cia3 -> deleted
Cib3 -> deleted
Cfa3 -> deleted
Cfb3 -> deleted
Cio3 -> deleted
Cia2 -> deleted
Cfa2 -> deleted
Cfb2 -> deleted
Cio2 -> deleted
Cia1 -> deleted
Cib1 -> deleted
Cfa1 -> deleted
ABTSOX -> deleted
DDf3 -> deleted
ABTS -> deleted
LO -> deleted
DDf2 -> deleted
DDio2 -> deleted
H2O2 -> deleted
```

```
Pyruvate -> deleted
DDia1 -> deleted
DDib1 -> deleted
DDfa1 -> deleted
DDfb1 -> deleted
DDio1 -> deleted
acetaldehyde -> deleted
POD -> deleted
DDia5 -> deleted
DDib5 -> deleted
{glucoseext => glucose} -> deleted
{acetoneext => acetone} -> deleted
{Lactateext => Lactate} -> deleted
{EtOHext => EtOH} -> deleted
{NO3ext => NO3} -> deleted
{NO2ext => NO2} -> deleted
{HRP+H_2O_2 => CCia5} -> deleted
{CCia5 => HRP+H_2O_2} -> deleted
{CCia5+resazurin => CCib5} -> deleted
{CCib5 => CCia5+resazurin} -> deleted
{CCib5 => HRP+resorufin} -> deleted
{HRP2+NADH => CCf4} -> deleted
{CCf4 => HRP2+NADH} -> deleted
{CCf4 => HRP2+NADN} -> deleted
{AO+isopropanol => CCf3} -> deleted
{CCf3 => AO+isopropanol} -> deleted
{CCf3 => CCio3+H_2O_2} -> deleted
{CCio3 => AO+HRP2} -> deleted
{ADH+NADH => CCia2} -> deleted
{CCia2 => ADH+NADH} -> deleted
{CCia2+acetone => CCib2} -> deleted
{CCib2 => CCia2+acetone} -> deleted
{CCib2 => CCfa2+NAD} -> deleted
{CCfa2 => ADH+isopropanol} -> deleted
{G_1DH+NAD => CCia1} -> deleted
{CCia1 => G_1DH+NAD} -> deleted
{CCia1+glucose => CCib1} -> {NO2+glucose => DAFF}
{CCib1 => CCia1+glucose} -> deleted
{CCib1 => CCfa1+NADH} -> deleted
{CCfa1 => G_1DH+gluconolacrone} -> deleted
{NO => volatNO} -> deleted
{NO+O2 => Cf6} -> deleted
{Cf6 => NO+O2} -> deleted
{Cf6 => NO2b+O2} -> deleted
{NO+NO2b => Cf5} -> deleted
{Cf5 => NO+NO2b} -> deleted
{Cf5 => N2O3} -> deleted
```

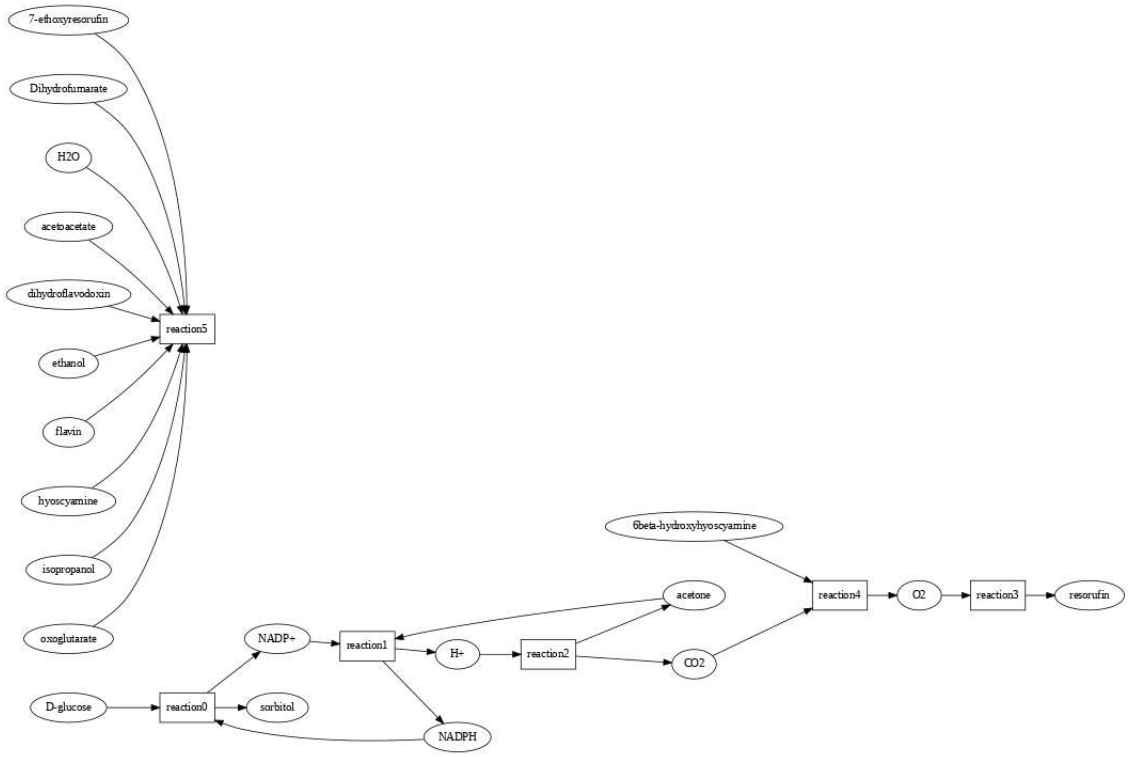
{DAF+N2O3 => Cf4} -> deleted
{Cf4 => DAF+N2O3} -> deleted
{Cf4 => DAFF} -> {NO2+glucose => DAFF}
{NO2+NR => Cia3} -> {NO2+glucose => DAFF}
{Cia3 => NO2+NR} -> deleted
{NADH+NR => Cib3} -> deleted
{Cib3 => NADH+NR} -> deleted
{Cia3+NADH => Cfa3} -> deleted
{Cib3+NO2 => Cfb3} -> {NO2+glucose => DAFF}
{Cfa3 => Cia3+NADH} -> deleted
{Cfb3 => Cib3+NO2} -> deleted
{Cfa3 => Cio3+NO} -> deleted
{Cfb3 => Cio3+NO} -> deleted
{Cio3 => NAD+NR} -> deleted
{NO3+NR => Cia2} -> deleted
{Cia2 => NO3+NR} -> deleted
{Cia2+NADH => Cfa2} -> deleted
{Cib3+NO3 => Cfb2} -> deleted
{Cfa2 => Cia2+NADH} -> deleted
{Cfb2 => Cib3+NO3} -> deleted
{Cfa2 => Cio2+NO2} -> deleted
{Cfb2 => Cio2+NO2} -> deleted
{Cio2 => NAD+NR} -> deleted
{G_1DH+NAD => Cia1} -> deleted
{Cia1 => G_1DH+NAD} -> deleted
{Cia1+glucose => Cib1} -> {NO2+glucose => DAFF}
{Cib1 => Cia1+glucose} -> deleted
{Cib1 => Cfa1+NADH} -> deleted
{Cfa1 => G_1DH+gluconolacrone} -> deleted
{ABTSOX+NADH => DDf3} -> deleted
{DDf3 => ABTSOX+NADH} -> deleted
{DDf3 => ABTS+NAD} -> deleted
{LO+Lactate => DDf2} -> deleted
{DDf2 => LO+Lactate} -> deleted
{DDf2 => DDio2+H2O2} -> deleted
{DDio2 => LO+Pyruvate} -> deleted
{ADH+EtOH => DDia1} -> deleted
{DDia1 => ADH+EtOH} -> deleted
{ADH+NAD => DDib1} -> deleted
{DDib1 => ADH+NAD} -> deleted
{DDia1+NAD => DDfa1} -> deleted
{DDib1+EtOH => DDfb1} -> deleted
{DDfa1 => DDia1+NAD} -> deleted
{DDfb1 => DDib1+EtOH} -> deleted
{DDfa1 => DDio1+acetaldehyde} -> deleted
{DDfb1 => DDio1+acetaldehyde} -> deleted
{DDio1 => ADH+NADH} -> deleted

```
{ABTS+POD => DDia5} -> deleted
{DDia5 => ABTS+POD} -> deleted
{DDia5+H2O2 => DDib5} -> deleted
{DDib5 => DDia5+H2O2} -> deleted
{DDib5 => ABTSOX+POD} -> deleted
Number of reductions: 1
```

6 5. SISO Solutions

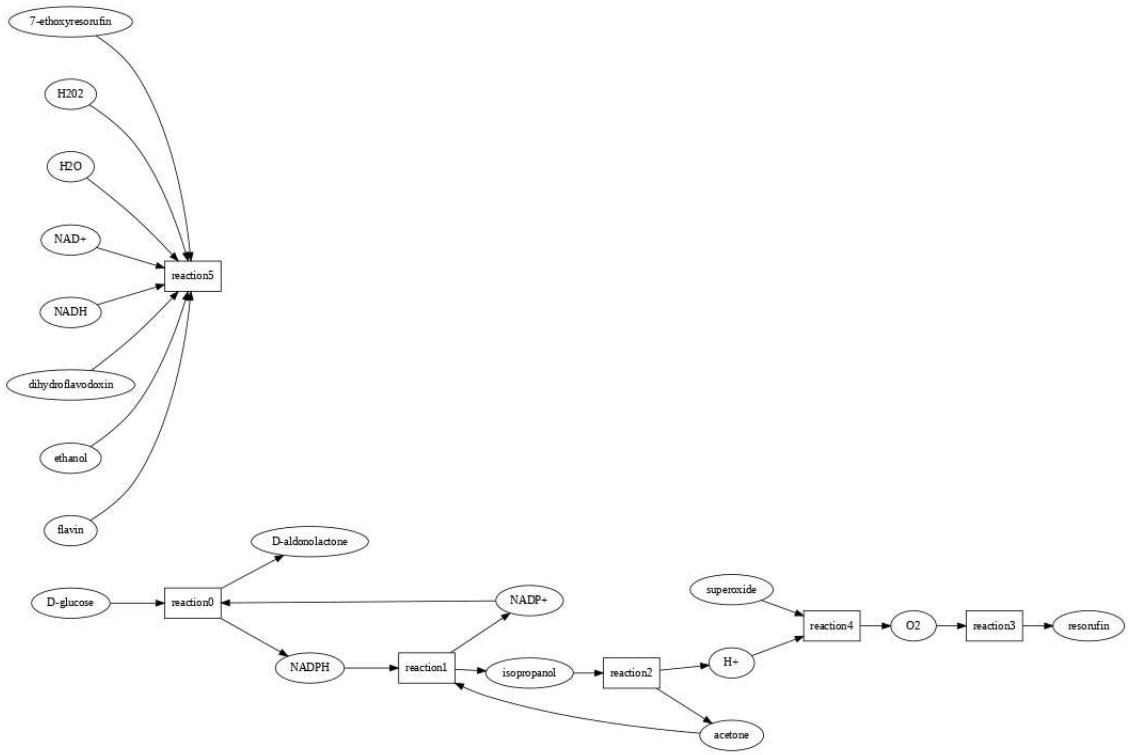
[13]:
```
new_model(CRNlong).
'D-glucose'+'NADPH'=>'NADP+'+'sorbitol'. %aldose reductase
'acetone'+'NADP+'=>'NADPH'+'H+'. %alcohol dehydrogenase (NADP+)
'H+'=>'acetone'+'CO2'. %2-oxopropyl-CoM reductase (carboxylating)
'O2'=>'resorufin'. %unspecific monooxygenase
'CO2'+'6beta-hydroxyhyoscyamine'=>'O2'. %hyoscyamine (6S)-dioxygenase
'dihydroflavodoxin'+'7-ethoxyresorufin'+'H2O'+'ethanol'+'flavin'+'acetoacetate'+'Dihydrofumara
 ↪=> _. %espèces envirronantes

export_biocham(CRNlong.bc).
list_model.
draw_reactions.
```



[13]:
```
MA(1) for 'D-glucose'+NADPH=>'NADP+'+sorbitol.
MA(1) for 'NADP+'+acetone=>'H+'+NADPH.
MA(1) for 'H+'=>CO2+acetone.
MA(1) for O2=>resorufin.
MA(1) for '6beta-hydroxyhyoscyamine'+CO2=>O2.
MA(1) for '7-ethoxyresorufin'+Dihydrofumarate+H2O+acetoacetate+dihydroflavodoxin
+ethanol+flavin+hyoscyamine+isopropanol+oxoglutarate=>_.
```

[16]:
```
new_model(CRNlong).
'D-glucose'+'NADP+'=>'NADPH'+'D-aldonolactone'.
'acetone'+'NADPH'=>'NADP+'+'isopropanol'.
'isopropanol'=>'acetone'+'H+'.
'O2'=>'resorufin'.
'H+'+'superoxide'=>'O2'.
'dihydroflavodoxin'+'7-ethoxyresorufin'+'H2O'+'ethanol'+'flavin'+'NAD+'+'NADH'+'H2O2'␣
↪=> _.
export_biocham(CRNlong.bc).
list_model.
draw_reactions.
```



[16]:
```
MA(1) for 'D-glucose'+'NADP+'=>'D-aldonolactone'+NADPH.
MA(1) for NADPH+acetone=>'NADP+'+isopropanol.
MA(1) for isopropanol=>'H+'+acetone.
```

```
MA(1) for O2=>resorufin.
MA(1) for 'H+'+superoxide=>O2.
MA(1) for
'7-ethoxyresorufin'+H2O2+H2O+'NAD+'+NADH+dihydroflavodoxin+ethanol+flavin=>_.
```