RAPPORT TRAVAIL D'ETUDE ET DE RECHERCHE

Analyse des phénotypes dans les réseaux de régulation génétiques: étude de cas du métabolisme énergétique dans le cancer du pancréas

MASTER 1 INFORMATIQUE
ROMAIN MICHELUCCI ET NINA SINGLAN
SOUS LA DIRECTION D'HELENE COLLAVIZZA

TABLE DES MATIERES

INTRODUCTION

RESUME

Tout au long de ce semestre, nous nous sommes attachés à comprendre le fonctionnement des réseaux de régulation génétique et la manière dont l'informatique entre en jeu pour aider les biologistes à les étudier.

A travers cette étude, nous avons notamment pu approfondir nos connaissances sur les logiques booléenne et temporelle, ainsi que sur la théorie des graphes et les méthodes de *model checking*.

Notre travail s'inscrit dans le développement d'une méthode de visualisation pertinente pour l'analyse des réseaux biologiques discrets. En effet, notre objectif a été de proposer des pistes de recherches pour adapter les méthodes déjà existantes dans le cadre booléen. Pour ce faire, nous avons étudié l'article « *Basins of Attraction, Commitment Sets and Phenotypes of Boolean Networks* » [KLA 18].

Par la suite, nous avons analysé plusieurs librairies pour comprendre comment, informatiquement, nous pouvions rendre cette solution applicable dans notre cas d'étude.

SUJET

Les réseaux de régulation génétiques définissent l'ensemble des influences individuelles d'un gène sur l'expression d'autres gènes. Ces interactions sont généralement bien connues des biologistes, mais la dynamique globale du système étant plus difficile à analyser une approche informatique s'avère nécessaire.

L'approche de l'équipe BioInfo Formelle est basée sur le formalisme discret de René Thomas [THO 01]. Le comportement dynamique est défini par un graphe de transition obtenu à partir du graphe d'interaction et d'un ensemble de paramètres cinétiques.

Le problème réside dans l'identification des ensembles de paramètres cohérents avec les propriétés dynamiques connues du système. Des méthodes formelles basées sur les CTL et le model-checking pour l'identification de ces paramètres ont été proposées par l'équipe. Cependant, pour les grands réseaux, le nombre de paramétrisations est trop important, il est alors impossible de tous les énumérer et tester.

L'existence d'une approche pour analyser les systèmes biologiques en termes de phénotypes dans le cadre des réseaux booléens a amené l'équipe à s'intéresser à l'application de cette méthode au cadre discret.

1 - CONTEXTE

1.1 - PRESENTATION DE LA RECHERCHE

Dans le cadre d'une cellule saine, un modèle de réseau de régulation du métabolisme énergétique a été proposé par l'équipe [KHD17]. Ce réseau et ses paramètres ont été identifiés à la main. Lors d'une rencontre avec les chercheurs de l'INSERM, la question de la modification de cette régulation dans le cas d'une cellule atteinte du cancer du pancréas s'est posée. Et elle a donné lieu à une collaboration entre les équipes. Le réseau a donc été modifié, notamment en introduisant des variables et en modifiant les seuils[1] du réseau. Lorsque le réseau subit une modification certains des paramètres doivent être de nouveau identifiés. Il est le plus gros réseau sur lequel l'équipe a été amenée à chercher une paramétrisation.

Ces modèles peuvent être représentés sous forme de graphes, les nœuds sont les objets biologiques et les arcs leurs interactions. Un tel graphe est appelé graphe d'interaction, il représente les influences individuelles des objets biologiques. A partir de ce graphe et d'un ensemble de paramètres cinétiques, on peut définir le graphe d'états, qui représente la dynamique globale du système.

En modélisant de cette manière, le graphe de notre réseau possède 11 variables (5 booléennes et 6 multivaluées, dont 5 appartenant à {0,1,2} et 1 appartenant à {0,1,2,3}), son graphe d'interaction possède 32 régulations (i.e. 32 arcs, il n'y a pas de calcul du nombre d'arcs car cela dépend du réseau on peut ne pas avoir d'arc entre 2 sommets !) et son graphe d'états possède 31 104 états ($2^5 * 3^5 * 4$).

Ainsi, il est impensable de pouvoir visualiser le graphe d'états, et il est nécessaire d'en obtenir une vision abstraite.

1.2 - MODELISATION INFORMATIQUE

Comme l'explique G. Bernot, J.-P. Comet et E. H. Snoussi, la biologie théorique manque de lois universelles, en effet lors de l'étude d'un phénomène biologique, la quantité de « propriétés universelles » est très faible par rapport à la quantité de propriétés *ad hoc*. Ainsi, les biologistes doivent souvent faire face à des résultats expérimentaux surprenants, et trouver les chaînes de causalité permettant d'expliquer ces résultats est un grand défi pour la biologie théorique [BER 14].

1.2.1 - MODELISATION BOOLEENNE

Depuis de nombreuses années, les systèmes biologiques sont assimilés à des réseaux booléens pour pouvoir les étudier. Il est possible de représenter les états d'une entité biologique avec une variable binaire, et de traduire les interactions entre ces différentes entités par des fonctions booléennes, permettant ainsi l'encodage du graphe d'interaction. Ceci nous permet la construction de table de vérité, mais également du graphe de transition d'états comme expliqué ci-après (Figure 1).

[1] Voir page 4.

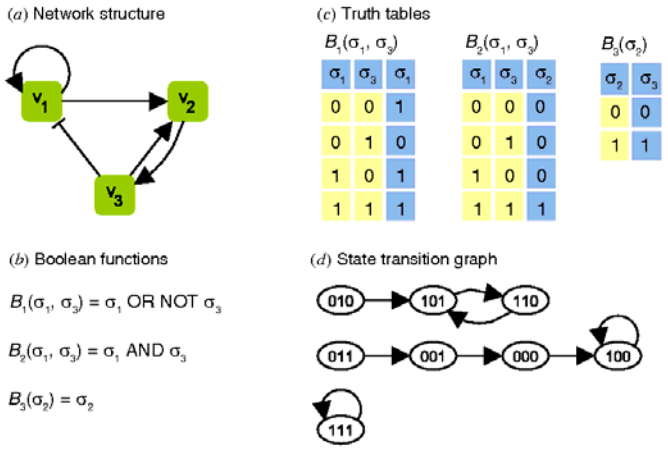

Figure 1. Exemple d'un réseau booléen simple. (a) Le graphe orienté associé au modèle booléen. Les arcs fléchés représentent des effets positifs et les arcs avec des flèches incomplètes indiquent des effets négatifs. Notons que le graphique ne détermine pas de manière unique les fonctions booléennes pour les nœuds v1 et v2. (b) Les fonctions booléennes dans le modèle. Le graphe peut prendre en charge des fonctions booléennes alternatives, en particulier B1 (σ1, σ3) = σ1 AND NOT σ3, B2 (σ1, σ3) = σ1 OR σ3. (c) Les tables de vérité des fonctions booléennes données en (b). (d) Le graphe de transition d'état du modèle booléen construit de manière synchrone. Les états 100 et 111 sont les points fixes du système, et les états 101 et 110 forment un cycle tiré de [WAN 12]

Il est intéressant de remarquer que le graphe d'interactions peut être recréé à partir des fonctions logiques et des tables de vérités mais l'inverse n'est pas vrai. Cela s'explique par le fait que les règles logiques encodent précisément les interactions tandis qu'à l'inverse plusieurs ensembles de règles logiques peuvent correspondre à un même graphe d'interactions.

1.2.2 – De la modelisation booleenne a la modelisation discrete

L'exemple suivant motive l'introduction d'un réseau multivalué. En effet, il met en évidence les limites de la modélisation booléenne.

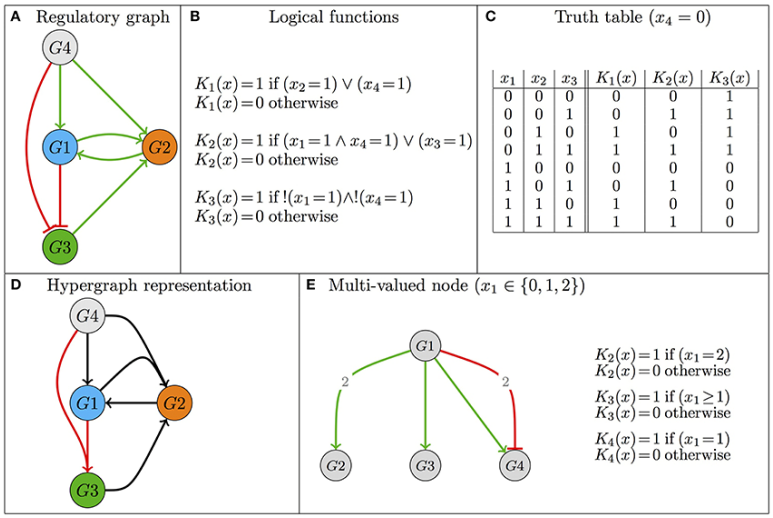

Figure 2 Le graphe d'interactions (A) permet de définir la topologie de la structure de régulation où les nœuds représentent des gènes et les arcs représentes les effets (activateurs en vert et inhibiteurs en rouge). Comme pour les réseaux booléens l'évolution des variables associées aux gènes régulateurs est définie par des formules logiques. (B). Cette évolution peut aussi être représentée sous forme de tables de vérité. (C). L'hypergraphe du réseau est une représentation alternative du modèle. (D). (E) G1 active G2 et G3 à différents seuils (respectivement 2 et 1) mais n'active G4 que si son seuil ne dépasse pas 1 (s'il le dépasse il l'inhibe). Cet exemple est tiré de [ABO16]

On pourra ainsi définir formellement un modèle logique (G, K) d'un réseau de régulation par un ensemble de n composants régulateurs (des gènes par exemple) $\mathrm{G} = \{ g_1, g_2, \ldots, g_n \}$ de cardinal n.

Pour chaque g_i on associe :

- Une variable entière définissant une cartographie discrète de l'ensemble des niveaux fonctionnels des composants.
- Un ensemble de seuil représentant le niveau nécessaire d'activation, ou d'inhibition.
- Une fonction discrète K_i qui définit sa valeur en fonction de ces seuils.

En effet, à la différence d'une discrétisation booléenne, un gène peut opérer à différents niveaux sur des cibles distinctes et ainsi avoir des effets différents. Dans ce cas, il est nécessaire de considérer des variables multivaluées dont la valeur maximale est supérieure à 1.

Les biologistes connaissent généralement bien les influences statiques, mais la dynamique générale du système se révèle souvent complexe. Mais en étudiant ces graphes, l'informatique peut permettre une analyse efficace de la dynamique de ces systèmes

1.2.3 – MODELISATION DISCRETE

Il est possible à chaque instant d'associer à chaque nœud une valeur numérique qui décrit le niveau de concentration de son produit dans la cellule. Les évolutions temporelles de ces niveaux constituent la dynamique du système.

Dans la continuité de l'exemple précédent (Figure 3), on peut introduire les graphes de transitions d'états (STG pour « *State Transition Graph* »).

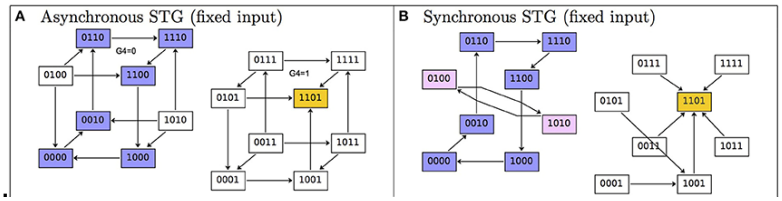

Figure 3. Graphe d'état asynchrone – « ATG » (A) et synchrone (B) tiré de [ABO16]

Dans le cadre des transitions synchrones, toutes les variables sont mises à jour simultanément. Dans celui des transitions asynchrones, les valeurs de plusieurs d'entre elles (mais pas de toutes) le sont.

Comme le nombre d'états est fini, lors des simulations de modèle on observe soit un « état stable » (en jaune, Figure 3) soit une trajectoire cyclique (en rose, Figure 3). Les états dits stables sont ceux qui n'ont aucune transition sortante et ceux dans une trajectoire cyclique forment une boucle de transition. Ces comportements asymptotiques sont appelés des « attracteurs ». Ils représentent les composants fortement connexes d'un STG.

Pour modéliser ces réseaux, l'équipe se base sur le formalise de René Thomas [THO 01].

Afin de mieux comprendre ce formalisme de modélisation discrète, nous allons prendre l'exemple de l'horloge circadienne de mammifère [BERXX].

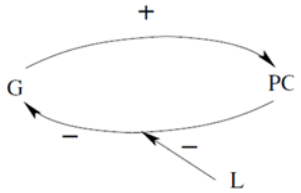

Figure 4. Modèle abstrait du système d'horloge circadienne des mammifères.

Ce système est constitué de 3 variables :

- G : les gènes de l'horloge
- PC : une protéine complexe
- L : l'apport en lumière

Ici, l'inhibition est représentée par des arcs négatifs et l'activation par des arcs positifs.

Ainsi, les dynamiques du système peuvent être résumées de la manière suivante : les gènes G produisent la protéine PC. Lorsque cette dernière est activée elle inhibe l'expression de G, sauf lorsque L entre en jeu.

En construisant le graphe d'influence de ce modèle on obtient la figure suivante :

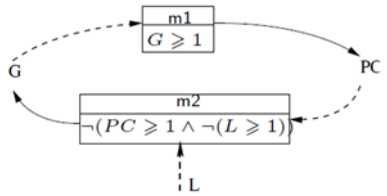

Figure 5. Graphe d'influence du modèle abstrait de l'horloge circadienne.

On y retrouve les informations suivantes :

- Seuils : de 0 à n (où n correspond au nombre d'arcs sortants)
- Activation : (x ≥ n) équivaut à « activation de x lorsque le seuil n est atteint »
- Inhibition : ¬ (x ≥ n) signifie « inhibition de x lorsque n est atteint »
- Multiplexes (m1 et m2) : formules logiques exprimant les combinaisons d'influence.

Le multiplexe m1 exprime l'idée suivante : « activation de PC lorsque G atteint le seuil 1 ». Celui de m2 signifie « inhibition de G lorsque PC atteint le seuil 1 lorsque L n'a pas atteint son seuil de 1 » (i.e. L n'inhibe pas la protéine PC).

On peut maintenant s'intéresser à la dynamique du système via ses paramètres notés K. Ils permettent de faire le lien entre la vue statique (graphe d'influence) et la vue dynamique du modèle (graphe de transition asynchrone). Ils représentent ainsi les combinaisons possibles d'influences sur x. Si un objet biologique x a des prédécesseurs p_1, p_2, ..., p_n ; le paramètre Kx, p_1 représente uniquement l'influence de p_1 sur x, alors que le paramètre $Kx, p_1 p_2$ représente les influences combinées de p_1 et p_2 sur x.

Ainsi pour notre exemple nous avons 5 paramètres :

$$K_{G,\{\}}, K_{G,\{m2\}}, K_{PC,\{\}}, K_{PC,\{m1\}} \; et \; K_{L,\{\}}$$

$K_{x,n}$ représente le seul paramètre dont les formules multiplexes sont valides. Ou plus simplement, il représente l'état où x « veut aller ».

A partir des paramètres calculés il est maintenant possible de construire le graphe de transition asynchrone.

Dans un premier temps pour chaque état n et variable x on trouve le paramètre $K_{x,n}$ applicable. Ensuite on choisit une interprétation notée σ à partir des paramètres et de leurs valeurs. Pour finir σ($K_{x,n}$) est une prochaine valeur possible pour x.

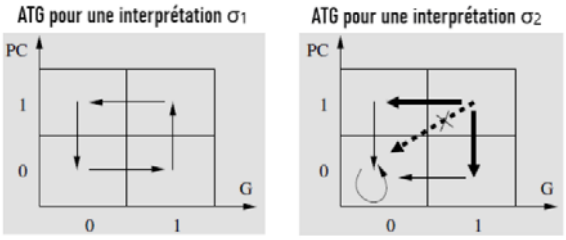

Figure 6. Graphes d'états résultants de deux interprétations différentes.

1.3 – PROBLEMATIQUE

Actuellement, il existe un certain nombre d'outils efficaces pour l'analyse et la visualisation des modélisations booléennes. Mais ce n'est pas le cas pour les modélisations discrètes. Bien que la transformation d'un réseau discret en un réseau booléen soit possible, elle provoque une explosion du nombre d'états. Sachant que la grande majorité des systèmes biologiques en contiennent déjà un grand nombre, l'explosion provoquée par leur transformation en valeur binaire rend pratiquement impossible leur analyse.

L'équipe a déjà mis en place un processus informatique permettant d'affiner le paramétrage de ces réseaux discrets (Figure 7). Mais, au vu du grand nombre de possibilités, le travail des biologistes reste laborieux. Un outil de visualisation performant leur permettrait une analyse plus rapide et plus efficace des réseaux.

Ainsi, notre objectif est de proposer des pistes permettant d'avancer dans l'adaptation des méthodes de visualisation booléennes au cadre discret.

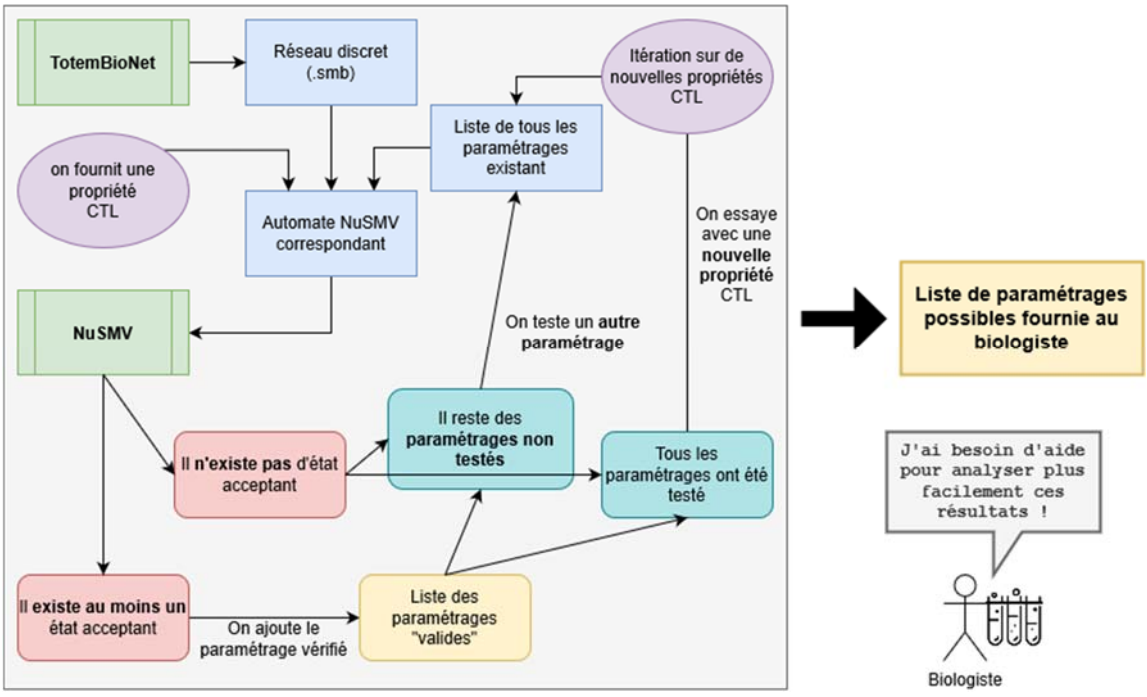

Figure 7. Illustration schématique du processus actuel. A partir d'un réseau discret, générer par TotemBioNet [2], des paramètres vont être énumérés. A l'aide de ces derniers, l'automate NuSMV correspondant est affiné. On lui applique une première formule CTL donnée. S'il existe un état acceptant, ce paramétrage est ajouté aux paramétrages valides, s'il n'y en a pas on essaye avec un autre paramétrage. Une fois que tous les paramétrages ont été testés, on itère avec une autre formule CTL.

[2] http://gmolines.github.io/pfe/oqp/al/ihm/2018/08/23/Y1819-S002/

2 - EXECUTION

2.1 - DEMARCHE DE L'ARTICLE

Afin d'atteindre notre objectif, nous nous sommes intéressés à l'article « *Basins of Attraction, Commitment Sets and Phenotypes of Boolean Networks* » [KLA18].

L'objectif de cet article est de définir de nouveaux outils formels de visualisation à l'aide de *PyBoolNet*[3]. C'est un package Python, qu'ils ont créé, permettant la génération, la modification et l'analyse de réseaux booléens. Le but premier de cette librairie est de permettre le calcul et la visualisation des « bassins d'attractions » et des « *commitment sets* ». Pour chacun de ces concepts, l'article propose des formules CTL sur les états acceptants.

En effet, les auteurs développent une utilisation du CTL *model-cheking* visant à calculer l'ensemble des états qui satisfont une requête.

Ces « états acceptants » sont définis comme suit :

The accepting states of a CTL formula φ and STG $(S, \to)$ are defined by

$$Accept(S, \to, \varphi) := \{x \in S \mid x \models \varphi\}$$

where $x \models \varphi$ means that φ holds in state x of the transition system $(S, \to)$.

Figure 8. Tiré de [KLA 18]

A partir de cette définition, le formalisme suivant sera utilisé :

$Enc(Accept(S, \to, \varphi))$ is an expression that represents the accepting states of φ in $(S, \to)$.

Figure 9. Tiré de [KLA 18]

Intéressons-nous maintenant aux différents types de bassins d'attractions.

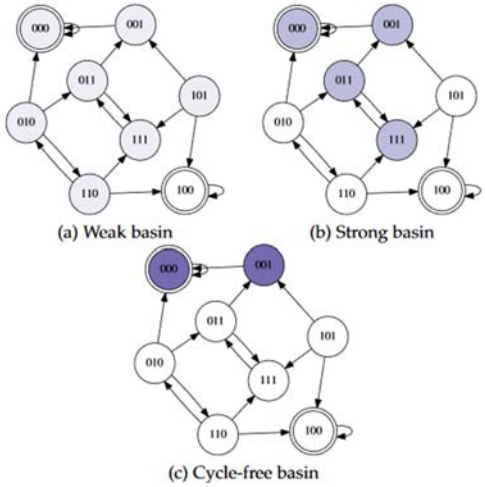

Figure 10. Représentations des différents types de bassins d'attraction d'un ATG. Les nœuds entourés de 2 cercles (000 et 100) correspondent aux états stables. En (a) les états grisés constituent le weak basin de l'attracteur 000 qui contient tous les états pouvant l'atteindre. En (b) le strong basin de l'attracteur 000 (constitués des états bleus) contient tous les états qui ne peuvent atteindre uniquement l'attracteur. En (c) les états violets constituent le cycle-free basin qui comme son nom l'indique ne contient que les états du strong basin sans cycle possible. Exemple tiré de [KLA 18]

[3] https://github.com/hklarner/PyBoolNet

Avant de présenter les définitions formelles de ces bassins, il est important de faire un rappel sur la sémantique des opérateurs CTL dont nous nous servirons.

- EF : *Exists Finally* : « Un état x satisfait EF(M) pour un ensemble d'état M, s'il existe un chemin partant de x arrivant à un état de M. »
- AG : *All Globally* : « Un état x satisfait la combinaison AG(EF(M)) si pour chaque état atteignable à partir de x il existe un chemin menant à un état dans M. »
- AF : *All Finally* : « Un état x satisfait AF(M) pour un ensemble d'état M tel que tous les chemins partant de x mènent forcément à M. »
- EX : *Exists Next* : « Un état x satisfait EX(M) si la formule donnée est dans un état accessible à partir de x par une seule transition. »

On remarque que ces définitions (exceptée celle de EX) correspondent aux états acceptants des différents types de bassins.

Formellement, un *weak basin* se définit comme suit :

$$Accept(S, \rightarrow, \alpha), \quad \text{where} \quad \alpha := \mathbf{EF}(Enc(a))$$

Figure 11. Tiré de [KLA 18]

La définition d'un *strong basin* se notera :

$$Accept(S, \rightarrow, \gamma), \quad \text{where} \quad \gamma := \mathbf{AG}(\mathbf{EF}(Enc(a)))$$

Figure 12. Tiré de [KLA 18]

Et celle d'un *cycle-free* sera :

$$Accept(S, \rightarrow, \beta), \quad \text{where} \quad \beta := \mathbf{AF}(Enc(A))$$

Figure 13. Tiré de [KLA 18]

Une fois les différents éléments définis, nous avons choisi de prendre un exemple afin de suivre au mieux la démarche de l'article.

$$\begin{aligned} Erk, \quad & Erk \ \& \ Mek \mid Mek \ \& \ Raf \\ Mek, \quad & Erk \mid Mek \ \& \ Raf \\ Raf, \quad & !\,Erk \mid !\,Raf \end{aligned}$$

Afin de le visualiser, nous avons généré le graphe de transition d'états correspondant.

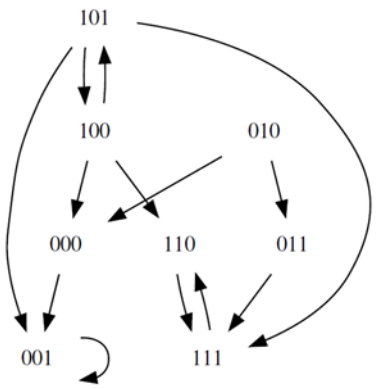

Figure 14. Graphe de transition d'états de l'exemple.

Sur cet exemple, *PyBoolNet* obtient les résultats suivants :

Attracteur simple :

001 → ! $\boldsymbol{Erk}$&! $\boldsymbol{Mek}$&$\boldsymbol{Raf}$

$week\_basin$ → ! (Erk & (Mek) | ! Erk & (Mek & (Raf)))

$strong\_basin$ → ! (Erk | (Mek))

Attracteur double :

11− → $\boldsymbol{Erk}$&$\boldsymbol{Mek}$&! $\boldsymbol{Raf}$

$week\_basin$ → Erk | (Mek)

$strong\_basin$ → Erk & (Mek) | ! Erk & (Mek & (Raf))

Nous pouvons maintenant visualiser les bassins d'attractions.

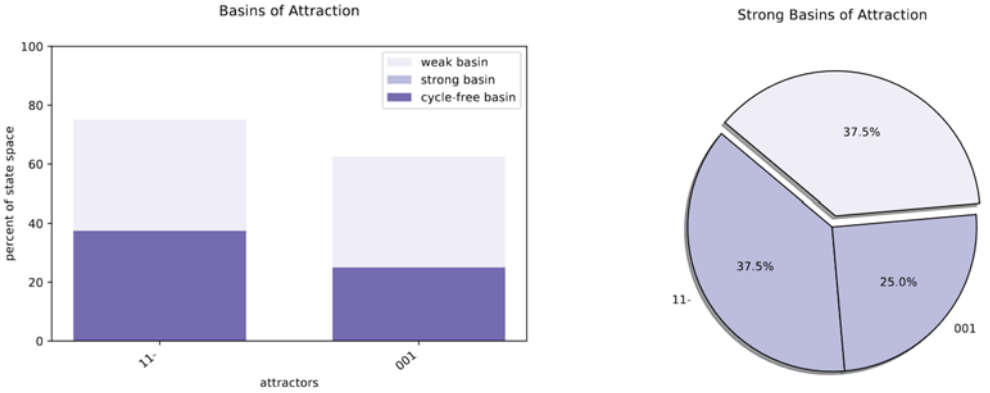

Figure 15. Histogramme et diagramme circulaire des bassins d'attractions.

L'histogramme nous montre la répartition des différents bassins d'attraction en fonction de leur type, et le diagramme circulaire nous montre le nombre d'états qui appartiennent à chaque bassin d'attraction « fort » (violet foncé) et le nombre qui ne leur appartiennent pas (violet clair).

Lors des calculs effectués, *PyBoolNet* génère 5 fichiers à l'aider de *NuSMV-a* [4]. Ces fichiers représentent les différentes itérations sur les CTL qu'effectue la librairie pour affiner ses calculs (présentées dans la Figure 7).

1) INIT TRUE
   $CTLSPEC$ EF(! Erk&! Mek&Raf) & AG(EF(! Erk&! Mek&Raf))

2) INIT TRUE
   $CTLSPEC$ EF(Erk&Mek) & AG(EF(Erk&Mek))

3) INIT TRUE
   $CTLSPEC$ EF(! Erk&! Mek&Raf) & EF(Erk&Mek) & AG(EF(! Erk&! Mek&Raf) | EF(Erk&Mek))

4) INIT ! (Erk & (Mek) | ! Erk & ((Raf) | ! Mek))
   $CTLSPEC$ EX(! (Erk | (Mek)))

5) INIT ! (Erk & (Mek) | ! Erk & ((Raf) | ! Mek))

[4] https://github.com/hklarner/NuSMV-a

$CTLSPEC$ EX(Erk & (Mek) | ! Erk & (Mek & (Raf)))

On remarque que la première CTL correspond à l'intersection du *weak* et du *strong* bassin de l'attracteur simple, quant à la seconde, elle correspond à cette intersection pour l'attracteur double.

La troisième CTL représente l'intersection des deux précédentes :

$$\mathbf{EF}(Enc(a_1)) \wedge \mathbf{EF}(Enc(a_2)) \wedge \mathbf{AG}(\mathbf{EF}(Enc(a_1) \vee Enc(a_2)))$$

Figure 13. Tiré de [KLA 18]

Cette CTL correspond parfaitement à la définition formelle des « *commitment sets* » :

$$\delta := \bigwedge_{i \in I} WeakBasin(a_i) \wedge StrongBasin(\{a_j \mid j \in I\}).$$

Figure 14. Tiré de [KLA 18]

On peut également déterminer s'il existe une transition entre les différents « *commitment sets* » :

L'arc $ComSet(I) \rightarrow ComSet(J)$ existe si $ComSet(I) \cap Accept(S, \rightarrow, \zeta) \neq \emptyset$

Ainsi, nous obtenons les diagrammes suivants :

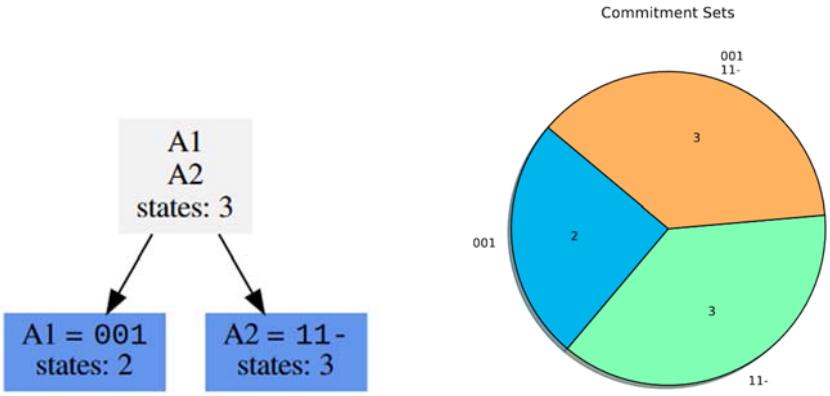

Figure 16. « Commitment diagram » et diagramme circulaire des commitment sets

Le diagramme circulaire nous intéresse tout particulièrement car il permet de visualiser les tailles relatives des différents « *commitment sets* ».

2.2 – ETUDE DES LIBRAIRIES EXISTANTES

Toujours dans le but de pouvoir appliquer la méthode déroulée dans l'article [KLA18] au cas discret, nous avons étudié plus en détail l'implémentation de ces librairies.

2.2.1 – PYBOOLNET

Nous avons naturellement commencé par étudier *PyBoolNet*. Cette librairie prend en entrée un réseau booléen, à partir duquel elle calcule les *primes implicants* du réseau (on dit que P est un *implicant* d'une fonction booléenne F, si P implique F, et on dit que P est un *prime implicant* s'il est minimal).

A partir de ces derniers, elle offre en sortie un grand nombre de propriétés de ce réseau. Une grande partie de ces propriétés est détectée à l'aide des attracteurs du réseau. En effet, *PyBoolNet* réussit à détecter ces états particuliers en utilisant l'algorithme de Nuutila [NUU94].

Cet algorithme, qui permet de détecter les composantes fortement connexes, est une amélioration de l'algorithme de Tarjan. La principale différence entre ces deux algorithmes réside dans l'amélioration de la fermeture transitive. En effet, la détection des composantes fortement connexes utilise l'algorithme de recherche en profondeur (DFS). Cependant, au lieu de construire partiellement les ensembles de successeurs des composantes puis de les combiner ou de les construire après avoir détecter complétement la composante, dans l'algorithme de Nuutila, ces derniers sont créés pendant la recherche. Cela évite des opérations inutiles et notamment de scanner le graphe plusieurs fois.

L'algorithme de fermeture transitive utilisé dans cette modification génère un unique ensemble de successeur pour chaque composante pendant la recherche en profondeur. Pour ce faire l'algorithme de Nuutila utilise une pile auxiliaire qui lui permet de collecter les composantes adjacentes qui seront nécessaires plus tard pour la construction de l'ensemble. Ainsi le graphe n'est scanné qu'une seule fois.

Outre le fait d'éviter de scanner 2 fois les arrêtes partants des sommets de la composante parcourue, elle permet aussi de stocker les composantes de manière consécutive dans la pile et ainsi en cas de problème de mémoire, le parcours de la pile nécessite moins d'opérations de pagination que l'accès à la liste d'adjacence contenant les sommets des composants.

Une fois le graphe des composantes fortement connexes obtenu, *PyBoolNet* en fait un graphe condensé, et regarde chaque nœud de ce nouveau graphe. Si un nœud n'a pas de successeur, on sait alors que c'est un attracteur. Et en regardant si ce nœud est composé d'un seul ou de plusieurs nœuds du graphe des composantes fortement connexes, on sait alors si cette attracteur est « stable » (i.e. il est composé d'un unique état) ou s'il est cyclique (i.e. composé de plusieurs états).

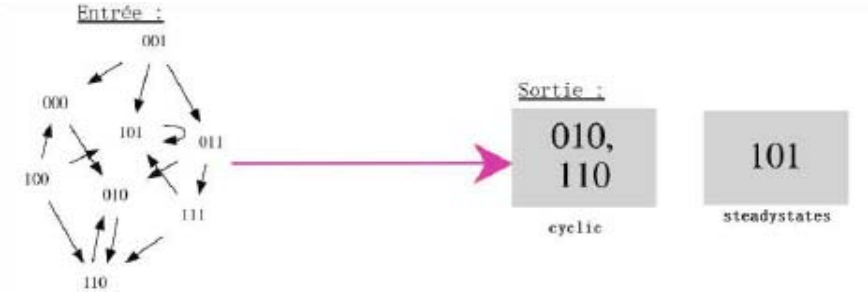

Figure 17. Exemple de la détection des attracteurs par PyBoolNet.

2.2.2 – NuSMV-a

En étudiant cette librairie en profondeur, nous avons découvert qu'elle avait principalement un rôle de visualisation. En effet, les calculs sont majoritairement réalisés par la librairie *NuSMV-a* qui est une extension de *NuSMV*[5] permettant d'afficher et de sauvegarder les états acceptant (i.e. les états qui valide une propriété CTL) d'un réseau.

[5] http://nusmv.fbk.eu/

```
Test sur le fichier : semaphoreCTL1.smv

(True, {'INIT': '!(semaphore | (proc1.state.1 | (proc1.state.0 | (proc2.state.1
| (proc2.state.0))))))', 'INIT_SIZE': 1, 'ACCEPTING': '!(semaphore & (proc1.state
.0 & (proc2.state.0)) | !semaphore & (proc1.state.0 | (proc2.state.0)))', 'ACCEP
TING_SIZE': 16, 'INITACCEPTING': '!(semaphore | (proc1.state.1 | (proc1.state.0
| (proc2.state.1 | (proc2.state.0))))))', 'INITACCEPTING_SIZE': 1})
```

Figure 18. Exemple de réponse de NuSMV-a sur un réseau booléen.

Mais lorsque l'on appelle la commande *NuSMV-a* sur un réseau multivalué on obtient une réponse incohérente.

```
CTLSPEC:              (operon = 0 -> AG !(operon = 2))
INIT:                 TRUE
INIT_SIZE:            8
ACCEPTING:            TRUE
ACCEPTING_SIZE:       8
INITACCEPTING:        TRUE
INITACCEPTING_SIZE:   8
ANSWER:               TRUE

CTLSPEC:              (operon = 2 -> AG !(operon = 0))
INIT:                 TRUE
INIT_SIZE:            8
ACCEPTING:            TRUE
ACCEPTING_SIZE:       8
INITACCEPTING:        TRUE
INITACCEPTING_SIZE:   8
ANSWER:               TRUE
```

Figure 19. Exemple de réponse de NuSMV-a sur un réseau multivalué.

Pour comprendre cette réponse, nous avons contacté Hannes Klarner, l'un des développeurs de *NuSMV-a*. Ce dernier nous a alors expliqué que cette extension avait seulement pour but d'analyser la réponse de *NuSMV*. L'extension ne modifie en rien cette réponse, elle ne fait aucun calcul, elle permet de récupérer des informations contenues dans *NuSMV* qui ne sont habituellement pas accessible à l'utilisateur.

Ainsi, afin de comprendre ce qui se passe dans le cas d'un réseau multivalué nous avons étudié *NuSMV*.

2.2.3 - NuSMV

NuSMV est un *model-checker* symbolique basé sur des diagrammes de décision binaire (BDD).

Afin de procéder à l'analyse du modèle, *NuSMV* commence par analyser le fichier d'entrée et par construire une représentation « plate et synchrone » du modèle. S'en suit un encodage en booléen de la représentation précédemment construite. Puis, *NuSMV* construit le BDD correspondant au problème et procède au *model-checking* [CIM02].

Lors de l'étude de cette librairie, nous avons notamment découvert que ses BDD étaient basés sur la librairie *Cudd*[6] qui permet leur construction et leur gestion.

[6] https://github.com/ivmai/cudd

Nous avons également trouvé une fonction permettant d'extraire des *primes implicants* de ces derniers. Cette méthode se base sur la théorie développée dans l'article [COU 93]. Elle nous a permis d'imaginer une nouvelle solution.

2.3 – EMERGENCE D'UNE NOUVELLE SOLUTION

L'étude de *NuSMV* nous a permis de comprendre la réponse incohérente que nous apportait *NuSMV-a* sur un réseau multivalué.

En effet, dans le cas d'un réseau booléen, l'encodage effectué par *NuSMV* ne change rien au modèle. Ainsi, lorsqu'il construit le BDD, ce dernier contient exactement les états du modèle d'origine. De cette manière, lorsque *NuSMV-a* analyse ce diagramme, il peut retrouver les états acceptants.

Mais, dans le cas d'un réseau multivalué, *NuSMV* construit un BDD calculé à partir de l'encodage booléen qu'il a effectué. Par conséquent, lorsque *NuSMV-a* analyse le diagramme, les états qu'il récupère ne correspondent en rien aux états du modèle initial.

Etant donné que *NuSMV* calcule un BDD correspondant au réseau multivalué et qu'il existe une méthode pour extraire les *primes implicants* d'un BDD [COU93] nous avons pensé qu'il serait intéressant d'extraire ces *primes*.

En effet, comme *PyBoolNet* effectue tous ses calculs à partir d'eux, en les obtenant, nous serions en mesure d'utiliser certaines méthodes de visualisation implémentée dans *PyBoolNet* sur un réseau multivalué.

3 – RESULTATS

3.1 – LES DIFFERENTES TENTATIVES

Le schéma suivant (Figure 20) présente l'idée générale de nos différentes tentatives. Comme expliquer dans la partie précédente, nous souhaitions « brancher » un fichier .svm à la librairie *PyBoolNet*.

En considérant que la fonction permettant d'extraire les *primes implicants* d'un BDD est déjà présente dans *NuSMV*, nous avons naturellement souhaité nous en servir. Comme la librairie sur laquelle nous souhaitions nous « brancher » est codée en Python, nous avons commencé par chercher à réécrire cette fonction en Python.

Par la suite, nous avons cherché à utiliser directement la fonction présente dans *NuSMV.*

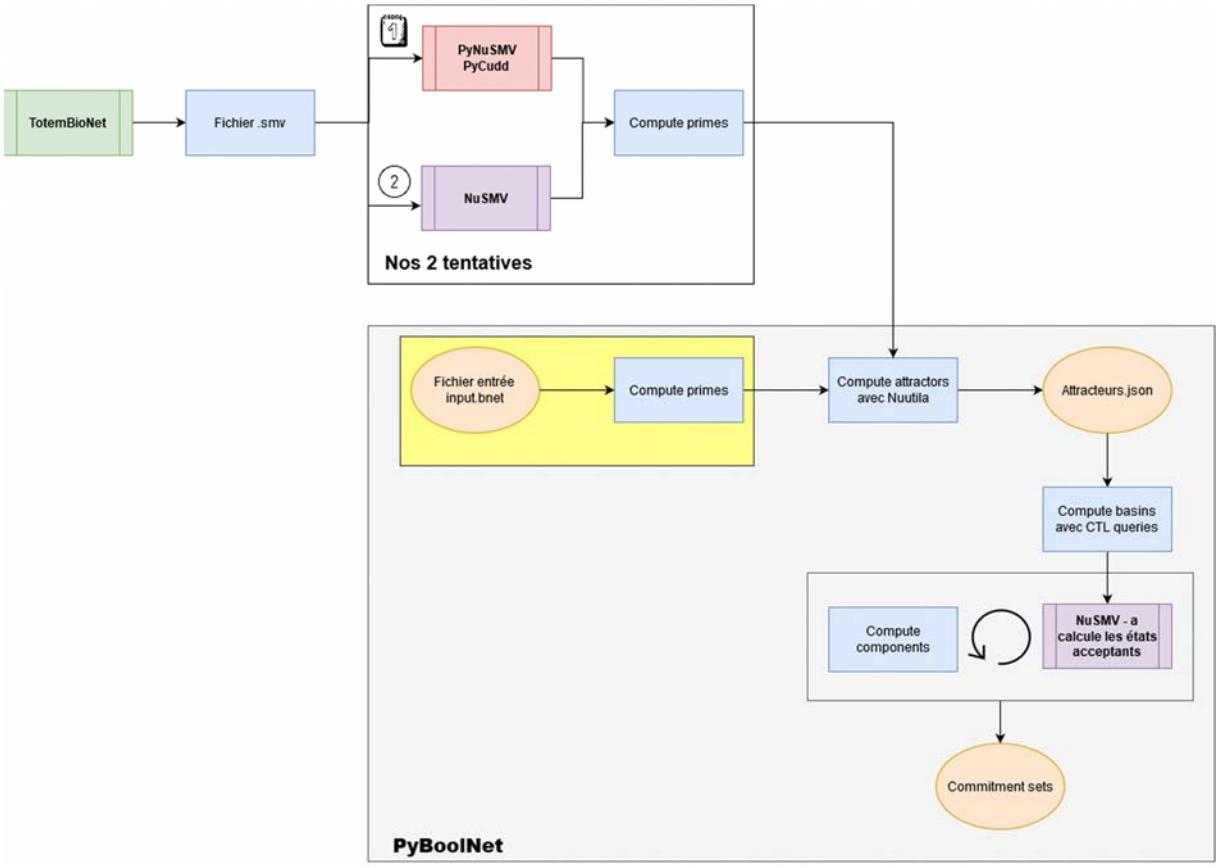

Figure 20. Résumé de nos tentatives. L'objectif est de substituer le chainon de départ de PyBoolNet (en jaune) par le nôtre (en blanc) afin de pouvoir connecter TotemBioNet à PyBoolNet

3.1.1 – L'AVENTURE PYTHON

1. *PyNuSMV*[7]

Au cours de notre étude de la librairie NuSMV nous nous sommes rendu compte qu'il existait une méthode de calcul de ces primes implicants déjà implémentée. Néanmoins cette dernière n'est pas accessible à travers les commandes du *shell*.

[7] https://github.com/sbusard/pynusmv

Nous avons donc cherché un moyen d'y accéder. Notre premier critère a été de chercher une bibliothèque python qui nous fournirait cet accès direct. Nous souhaitions garder le même langage de programmation pour programmer le chaînon manquant.

Nous avons ainsi découvert *PyNuSMV* dont la promesse était l'implémentation de NuSMV en python à l'aide de l'interface SWIG. Cependant, encore une fois, cette fameuse méthode n'a pas été implémentée. Nous avons ainsi tenté de compléter la librairie en y ajoutant cette dernière. Il semble que la raison de son absence soit due à un problème venant de la complexité de gérer des données complexes avec SWIG. Nous avons obtenu, en effet, un problème d'encodage rendant impossible la lecture de ces primes implicants.

Après cette tentative infructueuse nous avons tenté une nouvelle approche.

Comme énoncé précédemment, NuSMV utilise la librairie *Cudd*, une librairie C permettant la manipulation des diagrammes de décisions binaires (BDD) et algébriques (ADD). *PyNuSMV* s'en sert pour implémenter ces méthodes via *PyCudd*[8], un autre package python, qui recrée les méthodes du package *Cudd* en python. Nous nous sommes donc penchés dessus.

2. *PyCudd*

Ci-dessous (Figure 21) est représentée l'idée d'implémentation qui a émergée après l'étude de cette nouvelle librairie.

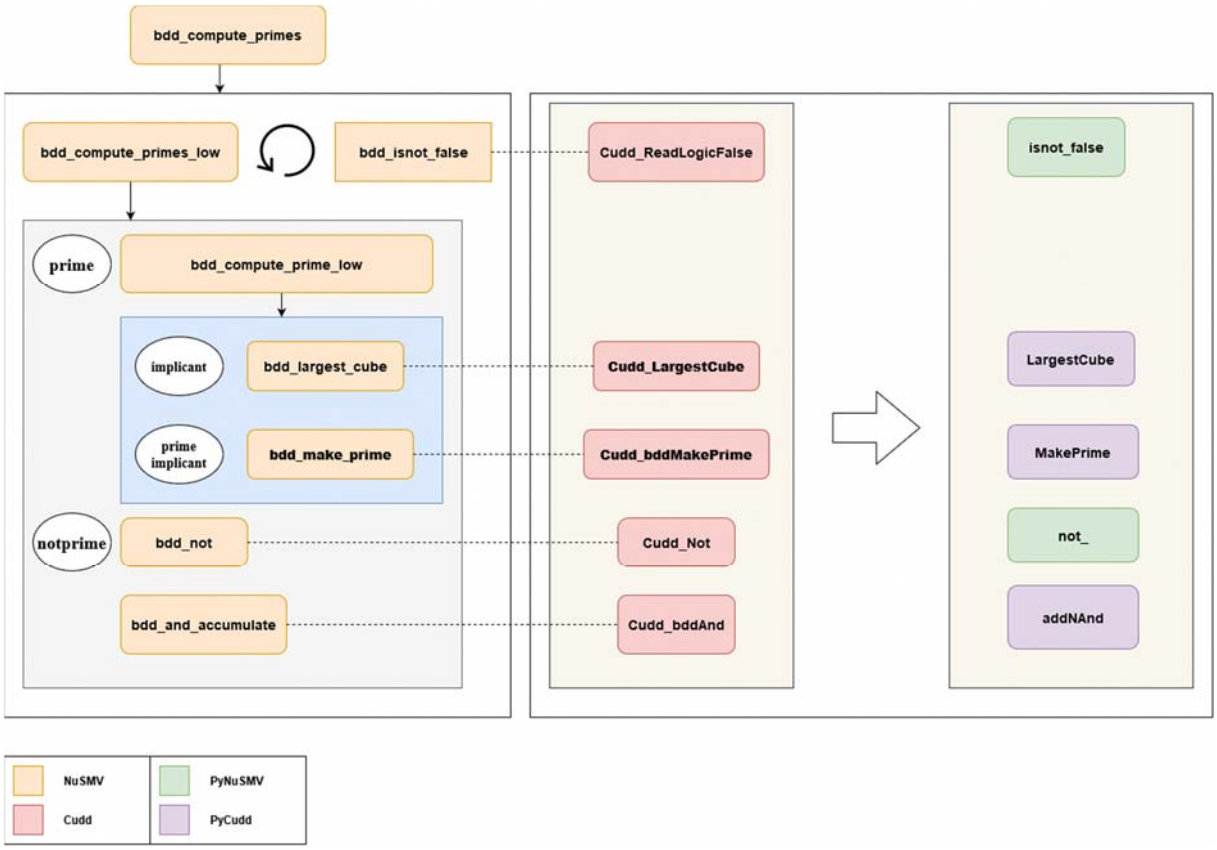

Figure 21. Résumé de notre approche pythonesque. La partie gauche du schéma montre la composition de la méthode bdd_compute_primes qui nous intéresse. En orange sont représentées les méthodes de NuSMV, en rouge sont les méthodes Cudd utilisées par les méthodes de NuSMV. La partie droite quant à elle montre l'équivalent de chacune des méthodes Cudd en python (présentes dans les librairies PyNuSMV et PyCudd) afin de les utiliser dans notre code.

[8] https://pypi.org/project/pycudd/

Le mélange d'implémentation de *PyNuSMV* et de *PyCudd*, bien que perturbant, s'explique par le fait que nous voulions tirer profit de cette nouvelle librairie tout en gardant les méthodes fonctionnelles de *PyNuSMV*.

Cette tentative s'est très vite soldée par un problème technique. Cette librairie n'étant pas disponible via pip nous avons tenté de l'installer à travers les fichiers « makefile » mis à disposition. Mais il nous a été impossible d'installer cette librairie sur notre machine à cause d'un problème lié à la distribution que nous utilisions.

3.1.2 – L'EPOPEE C

Nous avons donc tenté une approche différente en changeant d'angle et de langage.

Il semble ici important de souligner que *NuSMV* s'exécute sous forme de *shell* avec des paramètres définis. Mais il est aussi possible d'exécuter *NuSMV* de manière interactive dans le *shell* avec des commandes qui permettent un plus large choix d'exécution.

Notre première tentative a donc été d'intégrer la méthode manquante dans l'environnement *shell* de *NuSMV*. Là encore il ne nous a pas été possible de le faire car l'environnement fournissant des variables globales prédéfinies il n'en existait pas pour les paramètres de notre méthode.

Finalement nous avons opté pour une dernière approche. Elle consiste à créer un fichier C, externe à *NuSMV*, faisant appel à toutes les fonctions nécessaires à la génération des primes implicants dont voici la liste exhaustive : celle permettant de lire le modèle à partir du fichier, celle encodant les variables, celle créant le modèle et le diagramme de décision binaire et pour finir celle générant un fichier contenant les primes implicants du BDD.

3.2 – L'APPROCHE LA PLUS PROMETTEUSE

A ce stade de notre réflexion et de nos essais, nous avons choisi de nous inspirer des différentes librairies que nous avons étudiées.

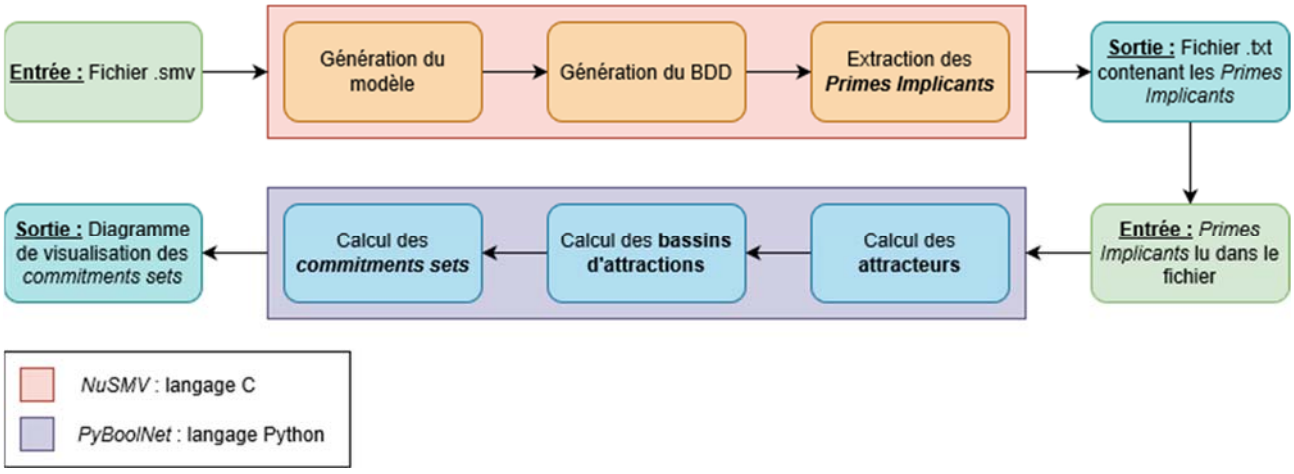

Figure 22. Illustration schématique de notre solution finale.

En prenant exemple sur *NuSMV-a*, nous avons choisi de mettre en place une extension de *NuSMV* permettant de générer le BDD et d'en récupérer les *primes implicants*. Cette extension les écrits ensuite dans un fichier texte qui sera lu par un programme Python.

Après avoir lu les primes, ce programme fait appel à *PyBoolNet* et applique la méthode de l'article [KLA 18] afin d'obtenir une visualisation des *commitments sets*.

A l'heure actuelle, cette approche n'est pas encore fonctionnelle. En effet, un certain nombre de problème se pose dans l'utilisation des fonctions de *NuSMV*. Ces dernières ont été créées dans le but d'être utilisées par l'intermédiaire du *shell*, elles ne peuvent donc pas nécessairement être appelées directement. En outre, la grande majorité d'entre elles utilise l'environnement spécifique de ce *shell* pour fonctionner.

4 – ETUDE CRITIQUE

4.1 – EVALUATION DE NOTRE SOLUTION

En s'insérant dans une vision à long terme, notre solution pourrait être perfectionnée.

En effet, notre extension de *NuSMV* se présenterait comme une fichier C, qu'il faudrait compiler, puis exécuter. Il aurait été plus efficace d'intégrer cette nouvelle fonctionnalité dans le *shell* de *NuSMV* sous la forme d'une option, comme l'ont fait les développeurs de *NuSMV-a*.

De plus, la chaîne d'exécution finale serait un script qui lance les différentes exécutions à la suite. Une solution plus aboutie pourrait se présenter sous la forme d'une librairie, permettant ainsi à un utilisateur de la modifier.

Mais, en dehors des améliorations au niveau de l'implémentation, il existe un problème de visualisation directement lié à la théorie des *primes implicants*. En effet, notre solution présenterait bien une visualisation, mais elle s'appuierait sur les états d'un BDD généré par *NuSMV*. Ces états ne sont donc pas facilement assimilables aux états du réseau initial.

Ainsi, pour rendre notre solution pleinement utilisable et véritablement efficace, il faudrait trouver une méthode permettant de remonter notre chaîne de production (i.e. de pouvoir revenir aux états initiaux). Et ainsi, proposer une « légende » (i.e. si un état généré « a » apparait dans la visualisation, une légende nous indique à quels états du réseau initial il correspond) à l'utilisateur, permettant de voir apparaître les états du modèle initial sur notre visualisation.

Pour mettre en œuvre cette solution, il faudrait ajouter une nouvelle fonctionnalité à *NuSMV* permettant de garder une « liste d'équivalence » entre les états initiaux et les états générés. Cette liste serait alors ajoutée au fichier texte. Et, en modifiant la fonction de visualisation de *PyBoolNet*, nous pourrions afficher ces équivalences sur le diagramme final.

4.2 – AUTRES APPROCHES POSSIBLES

D'autres solutions peuvent être envisagée. La première, bien que contre intuitive, consisterait à créer un traducteur de réseau discret en réseau booléen. En effet, cette dernière sera confrontée à une limite de taille dans la génération du réseau. Même si nous ne pouvons pas générer les graphes d'interactions ni les graphes d'états, nous serions en mesure de visualiser les « *commitments sets* ».

Une seconde approche de traduction aurait pu être envisagée. La transformation d'un automate *SMV* en un fichier *bnet* (le fichier d'entrée de *PyBoolNet*) serait une alternative aux primes implicants. Il nous permettrait ainsi de compléter la chaine d'une autre manière.

Ayant observé les limites de *NuSMV* à travers son utilisation des diagrammes de décisions binaires, nous pourrions, dans un cadre plus complet, développer une librairie symbolique de vérificateur de modèle basé sur les diagrammes de décision multiple (MDDs).

En effet, l'automate d'états finis du réseau discret serait automatiquement transformé en MDD ce qui faciliterait la récupération des primes implicants et éviterait la perte d'information résultant du passage en BDD. De plus, les opérations sur les MDDs pourraient être utiles à d'autres domaines de recherche et une autre librairie de visualisation basée sur ce nouveau package pourrait voir le jour.

CONCLUSION

Bien que notre solution ne soit, à l'heure actuelle, pas fonctionnelle et qu'elle puisse être améliorée, il semble évident qu'il soit possible d'utiliser les méthodes de visualisation des réseaux booléens pour visualiser les réseaux discrets.

En effet, tout au long de ce semestre, nous avons pu apporter et explorer de nombreuses pistes de recherches. Mais, avant d'arriver à une solution vraiment performante et efficace, il reste un grand nombre de problèmes à surmonter. Peut-être serait-il donc plus intéressant de mettre en place de nouveaux outils, permettant d'utiliser des MDD pour modéliser ces réseaux.

D'un point de vue plus personnel, ce Travail d'Etude et de Recherche nous a permis d'apprendre un grand nombre de méthodes et de théories, aussi bien en informatique, qu'en biologie. Nous avons également pu voir comment l'informatique peut entrer au service d'autres sciences (ici, la biologie), en permettant notamment une visualisation et une analyse plus efficace. Mais cet apport n'est pas à sens unique. En effet, l'informatique gagne également à ces échanges, par exemple, dans notre cas, la résolution de ce problème peut amener les informaticiens trouver des solutions innovantes à une problématique bien connue en informatique : l'explosion des états d'un graphe.

Nous avons également pu prendre conscience du milieu de la recherche. En effet, tout au long de ce semestre, nous avons eu la chance d'entrer en contact avec Hannes Klarner mais aussi avec l'équipe Bioinfo Formelle qui nous a guidée dans nos recherches en nous indiquant avec plus de précision leurs besoins.

BIBLIOGRAPHIE

[ABO16] ABOU-JAOUDE, W., TRAYNARD, P., MONTEIRO, P. T., SAEZ-RODRIGUEZ, J., HELIKAE, T., THIEFFRY, D., & CHAOUIYA, C. (2016). Logical Modeling and Dynamical Analysis of Cellular Networks. *Frontiers in Genetics vol.7*, 94.

[BER XX] BERNOT, G., COLLAVIZZA, H., & COMET, J.-P. (n.d.). Formal Methods for Model in Biology.

[BER14] BERNOT, G., COMET, J.-P., & SNOUSSI, E. (2014). Formal methods applied to gene network modelling. In L. FARINAS DEL CERRO, & K. INOUE, *Logical Modeling of Biological Systems* (pp. 245-289). ISTE & Wiley.

[CIM02] CIMATTI, A., GIUNCHIGLIA, E., PISTORE, M., ROVERI, M., SEBASTIANI, R., & TACCHELLA, A. (2002). *Integrating BDD-based and SAT-based Symbolic Model Checking.*

[COU93] COUDERT, O., & MADRE, J.-C. (1993). A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions.

[GUE04] GUESPIN, J., BERNOT, G., & COMET, J.-P. (2004). Les réseaux de régulation biologique : rencontre entre biologie et informatique. Production de mucus chez P.Aeruginosa. *Technique et Science Informatiques (RSTI série TSI), numéro spécial sur articles sélectionnés de AFADL'03, vol.23, Num.7*, 939-946.

[KHO17] KHOODEERAM, R., BERNOT, G., & TROSSET, J.-Y. (2017). An Ockham Razor model of energy metabolism. In V. NORRIS, F. KEPES, & P. AMAR, *Proc. of the Thematic Research School on Advances in Systems and Synthetic Biology* (pp. 81-101). EDP Science publisher.

[KLA18] KLARNER, H., HEINITZ, F., NEE, S., & SIEBERT, H. (2018). Basins of Attraction, Commitment Sets and Phenotypes of Boolean Networks. *IEEE/ACM transactions on computational biology and bioinformatics*.

[NUU94] NUUTILA, E. (1994). An Efficient Transitive Closure Algorithm for Cyclic Digraphs. *Information Processing Letters vol. 52*, 207-213.

[THO01] THOMAS, R., & KAUFMAN, M. (2001). Multistationarity, the basis of cell differentiation and memory. I. & II. *Chaos; vol. 11*, 170-195.

[WAN12] WANG, R.-S., SAADATPOUR, A., & ALBERT, R. (2012). Boolean modeling in systems biology: an overview of methodology and applications. *Physicals Biology 95*, 055001.

TABLE DES ILLUSTRATIONS

CODES

Ici, nous vous proposons de découvrir les GitHub sur lesquels nous avons travaillé tout au long du semestre.

- https://github.com/GameDisplayer/TER : ce GitHub nous a accompagné tout au long du TER, ainsi, vous pourrez y trouver les différents rapports intermédiaires, plus centrés sur le code, que nous avons rédigé, mais également des exemples d'utilisation des différentes librairies cités.
- https://github.com/nsgln/NuSMV-p : sur ce GitHub vous pourrez trouver la solution finale que nous proposons.