École Centrale de Nantes Université de Nantes École des Mines de Nantes

Master Automatique Robotique et Informatique Appliquée
Spécialité : Temps Réels
Année 2014 / 2015

THÈSE DE MASTER

Complétion des Réseaux de Régulation –
une Contribution pour le Process Hitting

Présentée et soutenue par :
Xinwei CHAI

Le
28 août 2015

Président :	Olivier H ROUX	Professeur
Examinateurs :	Olivier ROUX	Professeur
	Morgan MAGNIN	Maître de conférence
	Tony RIBEIRO	Docteur

Directeurs de thèse : Olivier ROUX & Morgan MAGNIN

Laboratoire : IRCCyN UMR CNRS 6597

Table de Matières

Résumé

L'étude des réseaux de régulation est l'un des sujets les plus importants de la bioinformatique. Concernant la modélisation, il y a trois différents formalismes principaux : le réseau booléen, le modèle de Thomas et le Process Hitting. Ce dernier, introduit que récemment, effectue une analyse plus efficace sur les réseaux de régulation par rapport à deux modèles traditionnels, qui peinent à faire face au problème de l'explosion de l'espaces des états issue de gros calcul. Le Process Hitting est capable de résoudre ces problèmes en précisant les actions au lieu des espaces d'états à l'aide de structure abstraites. Dans ce rapport, ces méthodes seront introduites et comparées et quelques remarques seront faites sur les approches existantes. À l'aide du formalisme du Process Hitting, les nouvelles approches de complétion seront expliquées de façon intuitive. La complétion sert à traiter les réseaux incomplets et retrouver la partie inconnue de ces réseaux selon les critères de régulation qui viennent des expériences effectuées. Cette procédure enrichit les raisonnements du Process Hitting.

Remerciements

Je tiens à remercier mes deux encadrants, M. Olivier ROUX et M. Morgan MAGNIN, pour leur soutien continu et leurs conseils tout au long de cette thèse. Je leur remercie pour tout ce qu'ils m'ont apporté, pour leur présence, leur patience, pour m'avoir fait confiance et m'avoir laissé la liberté nécessaire à l'accomplissement de mes recherches, tout en gardant un oeil critique et avisé. Je les remercie également pour plusieurs relectures de ce rapport. Je remercie beaucoup M. Olivier H.ROUX, responsable de Master II ARIA à l'École Centrale de Nantes. J'adresse tous mes reconnaissances à tous mes professeurs pendant mes études de Master, notamment de 3e année cycle ingénieur informatique. Je tiens également à remercier tout personnel et professeur de l'École Centrale de Nantes ayant contribué de près ou de loin au bon déroulement de ce séminaire bibliographique. Je remercie également les camarades du laboratoire qui m'ont aidé pendant ce dur semestre. Finalement, un grand merci chaleureux et de tout mon cœur à mes parents, sans qui je ne serais absolument pas où j'en suis aujourd'hui. Je les remercie pour leur encouragements et soutiens constants.

Introduction

Cette thèse de Master a été réalisée à l'IRCCyN, au sein de l'équipe MeForBio. Elle s'inscrit dans une démarche d'application de méthodes et d'outils formels au domaine de la bioinformatique.

Contexte scientifique

Le terme bioinformatique est très générique : il inclut aujourd'hui tous les domaines de recherche utilisant les technologies de l'information dans le but d'étudier les systèmes biologiques. Parmi ces applications nous présentons le réseau de régulation. La régulation est un aspect clef des systèmes biologiques, dont l'échelle va de moléculaire à écologique. Acquérir une compréhension précise de la régulation est l'un des objectifs principaux de la biologie des systèmes. Il y a plusieurs méthodes de modélisation, parmi lesquelles le Process Hitting qui décrit le plus précisément le système mais rencontre des difficultés de complexité au niveau calcul, ce qui exige le recours à une méthode efficace, non exhaustive pour le calcul. Dans cet article, nous introduisons le graphe de causalité pour raisonner l'accessibilité et pour compléter le réseau biologique donné.

Bien qu'il y ait beaucoup de modèles qui donnent de bons résultats sur les propriétés de systèmes biologiques, dans les réseaux de grande échelle il existe souvent des parties de système qui restent inconnues, ce qui gênent l'exactitude de la modélisation. Pour faire face à cette difficulté, la complétion qui est le cœur de ce rapport, sert à retrouver ces parties manquantes. Cette complétion, s'applique également au systèmes représenté par le Process Hitting. Par conséquent, la recherche des algorithmes efficace de complétion devient cruciale.

Plan de l'article

La première partie consiste en un rappel sur les modèles booléen et une des méthodes de complétion liée à ce modèle, ensuite nous introduisons le modèle de Thomas qui est plus précis et multifonctionnel. Dans un troisième temps nous introduisons le Process Hitting, dont les bases s'inspire du modèle de Thomas. Dans cette partie nous présenterons également la causalité locale qui est une base incontournable de la détermination de l'accessibilité et la complétion. Ces deux parties constituent le noyau de ce rapport. Ensuite nous présentons une autre approche de complétion des réseaux de Process Hitting, inventée par notre camarade de l'IRCCyN *Emna Ben Abdallah*. Enfin une discussion courte sera dédiée à comparer les différentes approches de complétion.

Chapter 1

État de l'art

Quant à l'analyse de réseaux de régulation biologique, le réseau booléen est un formalisme fondamental et efficace, beaucoup d'études théoriques et pratiques ont été faites [Fox93], mais ce modèle est assez restreignant, comme l'analyse est statique en appliquant l'algèbre booléen.

Comme les valeurs sont souvent discrétisées, portant plus que deux niveaux de seuil, et l'analyse dynamique est souvent demandée, pour répondre à cette question, le nouveau formalisme Process Hitting est très compétant pour analyser les états discrets par rapport aux méthodes traditionnelles.

1.1 Réseaux de régulation

Un réseau de régulation biologique (RRB) décrit les interactions entre les entités biologiques, souvent des macromolécule ou des gènes d'un système donné. Il est statiquement représenté par un graphe d'interaction dont les sommets abstraient les entités biologiques et les arcs abstraient leurs interactions. Pour décrire l'évolution du système, le niveau de concentration de chaque entité est représenté par une valeur associée au sommet correspondant. L'évolution temporelle de ces niveaux constitue la dynamique du système [RCB06].

1.2 Réseaux booléens

Le réseau booléen est un cas particulier de RRB. Un réseau booléen $G(V, F)$ consiste d'un ensemble $V = \{v_1, \cdots, v_n\}$ des nœuds (d'entrée, de sortie et intérieur respectivement) et une liste $F = (f_1, \cdots, f_n)$ des fonctions booléennes, où les nœuds prennent des valeurs booléennes et les fonction booléennes $f_i(v_{i_1}, \cdots, v_{i_l})$ avec les données des nœuds spécifiés $v_{i_1}, \cdots, v_{i_l}$ est liées à tous les nœuds intérieurs et les nœuds de sortie v_i. Les valeurs des nœuds d'entrée et des nœuds de sortie sont connues et les valeurs des nœuds intérieurs sont partiellement ou complètement inconnues, cela reste donc à déterminer (compléter).

Un $(h + 1)$-dimension vecteur booléen $\mathbf{e}$ est appelé un exemple, où les premiers h éléments cor-

respondent aux nœuds d'entrée et le dernier élément correspond au nœud de sortie. Un exemple exprime les résultats des expériences ou les connaissances existantes. Un réseau booléen G est dit cohérent avec $\mathbf{e}$ si $\hat{v}_n = \mathbf{e}_{h+1} \wedge \forall \hat{v}_i = \mathbf{e}_i$ où $i = 1, \cdots, h$.

Complétion de réseau booléen

La complétion des réseaux booléens constitue des 2 problèmes suivants [ATH09] :

1. Étant donné un réseau booléen $G(V, F)$ incomplet, un ensemble d'exemples $\{\mathbf{e}^1, \cdots, \mathbf{e}^m\}$. Existe-t-elle une attribution des fonctions booléennes f_i telles que le réseau obtenu $G = (V, F')$ soit cohérent avec tous les exemples?

2. Étant donné un réseau booléen $G(V, F)$ complet, un ensemble d'exemples $\mathbf{e}^1, \cdots, \mathbf{e}^m$ et un entier positif L. Existe-t-elle existe une attribution des fonctions booléennes f_i sur au plus L nœuds telles que le réseau obtenu $G = (V, F')$ soit cohérent avec tous les exemples?

En fait, le problème 1 correspond à la complétion de modèle, et le problème 2 correspond à la modification de modèle, dont Atsuku et al. ont proposé des algorithmes. Comme il y a souvent des erreurs de connaissances qui sont donc la base du réseau, il vaut mieux de modifier un nombre minimum de nœuds (L) en gardant le plus que possible la de structure du réseau.

On peut chercher également les états stables à l'aide de l'analyse de réseau booléen, en plus, si on veut analyser la dynamique des systèmes, la sémantique booléenne n'est plus suffisante, à cause du non-déterminisme. E.g. Étant donné une réaction chimique $A + B \Longrightarrow C + D$, il y aura 4 états futurs possibles : A et B pourront être consommés partiellement ou complètement. A ce stade, le Process Hitting est très compétant de décrire le comportement non-déterministe. Mais le réseau ne représente que l'état stable et ne comprend que 2 valeurs, par conséquent, il est incapable de modéliser les cas ayant conflits ou l'évolution dans une période.

Dans [ATH09], la complétion est définie de la façon suivante : chercher toutes les combinatoires possibles des fonctions booléennes cohérentes avec les exemples donnés. Mais cette approche conduit à un calcul énorme, qui est aussi démontré dans [ATH09]. Ce problème nous fait penser à contraindre le modèle. (relation de régulation R, qui sera présentée après)

Pour mieux comprendre comment pratiquer la complétion, l'approche en utilisant le langage SBGN-AF est un bon exemple.

1.2.1 Langage SBGN-AF

Le SBGN-AF (System Biology Graphical Notation, Activity Flow en anglais) est un des formalismes standardisés principaux pour représenter les réseaux de régulation et les influences entre eux. Cette approche sert à trouver les bonnes fonctions booléennes selon les hypothèses. Le langage du SBGN-AF consiste d'un ensemble d'étiquettes, classé dans 5 groupes : *nœuds d'activité, unités auxiliaires, nœuds de container, arcs de modulation et opérateurs logiques.*

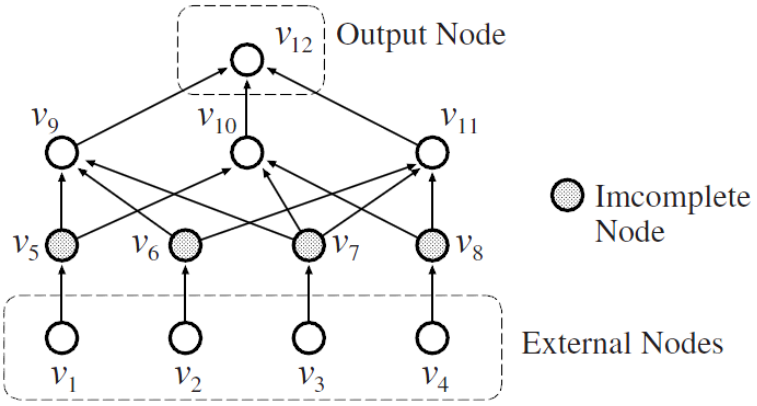

Figure 1.1: Un réseau booléen, selon les expériences, plusieurs exemples sont disponibles : $\mathbf{e}^i = \{v_1^i, v_2^i, v_3^i, v_4^i, v_{12}^i\}, i \in [1; n]$ avec n le total d'expériences, car les fonctions booléennes de $v_9 \sim v_{11}$ sont déjà connues, il reste donc à déterminer celles de $v_5 \sim v_8$

Exemple : dans le Figure 1.2,

Nœuds d'activité :

$$activity(A_1), activity(A_2), activity(A_3), activity(A_4)$$

opérateurs logiques :

$$or(lo_1), and(lo_2), not(lo_3)$$

arcs de modulation :

$$input(A_1, lo_1), input(A_2, lo_1), input(lo_1, lo_2), input(lo_3, lo_2)$$

$$stimulates(lo_2, A_4), inhibits(A_4, A_1)$$

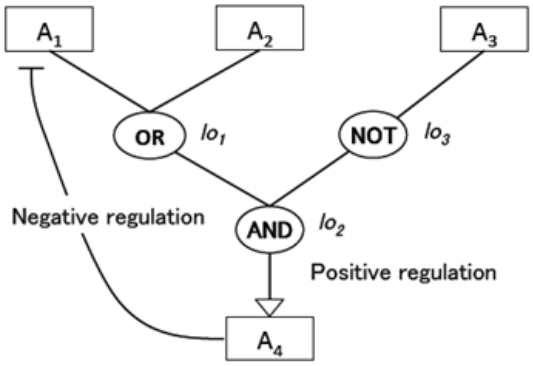

Figure 1.2: Exemple de réseau SBGN-AF qui signifie $A_4 = (A_1 \vee A_2) \wedge \neg A_3$ et $A_1 = \neg A_4$

Le raisonnement de complétion est comme suit :

Étant donné un exemple $\mathbf{e}$, on cherche une hypothèse H par rapport aux connaissances existantes B et $\mathbf{e}$, on en déduit $B \wedge \neg H \models \neg \mathbf{e}$ et $B \not\models \neg H$. En conséquence, $\neg H$ est considérée comme une

conséquence de $B \wedge \neg \mathbf{e}$. Avec l'outil SOLAR, les hypothèses sont trouvées. C'est pour ceci cette méthode est nommée "Hypothesis finding" [YRN+14, NIIR10].

C'est un raisonnement binaire appliquant sur les réseaux booléens, qui n'étudie pas les états des éléments dans le système, mais il nous inspire de renverser le lien causal entre les hypothèses et les observations. (recherche des hypothèses qui vérifient l'observation). Avant de parler du Process Hitting, une méthode plus précise possédant plusieurs caractéristiques, nous aimerions présenter sa base : modèle de Thomas.

1.3 Modèle de Thomas

Étant un modèle plus précis que le réseau booléen, le modèle de Thomas contient plusieurs valeurs à un nœud au lieu de deux, et il propose une description dynamique (attracteur) au lieu de fonctions booléennes, qui donne la possibilité de simuler les transitions des réseaux biologiques.

Nous utilisons souvent les équations différentielles pour décrire la dynamique de chaque partie d'un RRB afin d'obtenir une série d'équations différentielles. A partir d'un RRB, on a une série d'équations différentielles linéaires par morceaux [Fil60]. Pour simplifier l'analyse, ces équations sont considérées équivalentes à un ensemble des éléments discrets, qui change le problème en l'analyse d'un espace d'états [GK73].

Selon Thomas, un RRB peut être représenté formellement et qualitativement par un graphe. (e.g. La concentration d'un certain matière est classée par plusieurs niveaux, mais les écarts entre niveaux ne sont pas forcément égaux)

Definition 1.1 (Graphe de régulation)**.** Un graphe de régulation est un graphe orienté étiqueté, noté $G = (V, E)$, chaque sommet $v \in V$ est une variable discrète avec son seuil $b_v \in \mathbb{N}$ inférieur à son degré sortant. Chaque arc est étiqueté par un couple $(t_{\mathrm{uv}}, \alpha_{\mathrm{uv}})$, dont t_{uv} est un entier compris entre 1 et b_u, appelé le seuil qualitatif et où $\alpha_{uv} \in \{+, -\}$ est la signe de régulation.

Toutes régulations sont considérées comme effectives, i.e. lorsque $\alpha_{uv} = +, u \geq t_{uv}$ veut dire u active v et $u < t_{uv}$ veut dire u inhibe v, vice versa pour le cas $\alpha_{uv} = -$.

Definition 1.2 (État qualitatif)**.** Étant donné $G = (V, E)$ un graphe de régulation, un état qualitatif de G est un vecteur $q = (q_v)_{v \in V}$ tel que pour tout $v \in V$, $q_v \in \{0, 1, ..., b_v\}$. L'ensemble Q des états de G est alors défini comme $Q = \prod_{v \in V} \{0, 1, ..., b_v\}$.

Definition 1.3 (Ressource)**.** $G = (V, E)$ est un graphe de régulation, $v \in V$et $q \in Q$. L'ensemble $\omega_v(q)$ de ressources de v à l'état q est le sous-ensemble de $G^-(v)$:

$$\omega_v(q) = \{u \in G^-(v) \mid (q_u \geq t_{uv} \; et \; \alpha_{uv} = +) \; ou \; (q_u \leq t_{uv} \; et \; \alpha_{uv} = -)\}$$

À l'état q, l'évolution de la variable v dépend de ses ressources $\omega_v(q)$. Il reste à définir la direction d'évolution. Le paramètre $K_{v,\omega_v(q)}$ est appelé l'attracteur de v lorsque l'ensemble de ressources est $\omega_v(q)$. Il donne la tendance d'évolution de v:

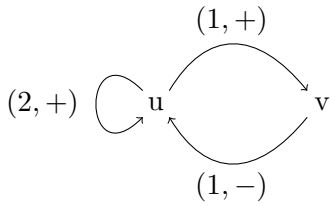

Figure 1.3: (a)

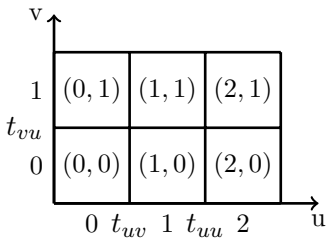

Figure 1.4: (b)

Figure 1.5: (a) Un graphe de régulation (b) L'espace d'états de ce graphe de régulation

- si $q_v < K_{v,\omega_v(q)}$, alors v va augmenter
- si $q_v = K_{v,\omega_v(q)}$, alors v va rester invariant
- si $q_v > K_{v,\omega_v(q)}$, alors v va diminuer

Les valeurs différentes des attracteurs sont possibles, les combinaisons d'attracteurs sont appelées *modèles*.

Definition 1.4 (Modèle)**.** $G = (V, E)$ est un graphe de régulation. Un modèle de G, noté par $M(G)$, est un groupe de nombres naturels $K_{v,\omega}$ indexés par la par l'ensemble des couple (v, ω) tel que :

- $v \in V$
- $\omega \subseteq G^-(v)$
- $K_{v,\omega} \leq b_v$

Il exige souvent que : $K_{v,\omega} \leq K_{v,\omega'}$ pour tous les $v \in V$ et pour tous les $\omega, \omega' \subseteq G$ tels que $\omega \subseteq \omega'$ (Les contraintes de Snoussi) [RCB06].

Modèle	u	v	Attracteurs		Tendances	
$K_{u,\{\}} = 0$	0	0	$K_{u,\{v\}} = 2$	$K_{v,\{\}} = 0$	$\nearrow$	$\rightarrow$
$K_{u,\{u\}} = 2$	0	1	$K_{u,\{\}} = 0$	$K_{v,\{\}} = 0$	$\rightarrow$	$\searrow$
$K_{u,\{v\}} = 0$	1	0	$K_{u,\{v\}} = 2$	$K_{v,\{u\}} = 1$	$\nearrow$	$\nearrow$
$K_{u,\{u,v\}} = 2$	1	1	$K_{u,\{\}} = 0$	$K_{v,\{u\}} = 1$	$\nearrow$	$\rightarrow$
$K_{v,\{\}} = 0$	2	0	$K_{u,\{u,v\}} = 2$	$K_{v,\{u\}} = 1$	$\rightarrow$	$\nearrow$
$K_{u,\{u\}} = 1$	2	1	$K_{u,\{u\}} = 2$	$K_{v,\{u\}} = 1$	$\rightarrow$	$\rightarrow$

Table 1.1: Un modèle possible du graphe de régulation du Figure 1.5. Ce tableau montre les attracteurs correspondant à chaque état qui est déduit du modèle.

1.3.1 Boucles de RRB

Lorsqu'on analyse un RRB, on fait souvent attention aux boucles positives et boucles négatives. Une boucle est positive (resp. négative) si dans son circuit le nombre des signes de régulation négative

est pair (resp. impair). Une boucle positive active ou inhibe toujours les variables dans le circuit, qui atteint finalement son seuil ou le minimum permis afin de rester toujours disponible. Un boucle négative active (resp. inhibe) toutes les variables lorsque cette variable atteint un assez bas (resp. haut) niveau, afin de stabiliser le système en comportant comme l'oscillation ou la multistationnarité [Tho78].

1.3.2 Dynamique des modèles

L'approche classique pour décrire la dynamique de modèles est de définir l'état du système au moment t+1 à partir de l'état du moment t. L'approche synchrone est de considérer que l'état suivant est l'attracteur de l'état actuel : si q est l'état actuel alors $q' = (K_{v,\omega_v(q)})_{v\in V}$ est le prochain état, c'est-à-dire il y a une transition de q à q'. Cette approche entraîne des problèmes graves, principalement à cause des propriétés déterministes [RCB06].

Alors ces problèmes nous mène à une nouvelle approche :

Definition 1.5 (graph d'état asynchrone)**.** $G = (V, E)$ est un graphe de régulation et $M(G)$ est un modèle de G. Le graphe d'état asynchrone est un graphe orienté dont l'ensemble des sommets est l'ensemble Q des états de G, tel qu'il existe un arc de q à q' si :

- Pour tous les variables $v \in V$, $q_v = q'_v = K_{v,\omega_v(q)}$ ou
- Il existe une variable $v \in V$ telle que :
  - Pour toute variable $u \neq v$, $q_u = q'_u$ et
  - $q_v < K_{v,\omega_v(q)}$ et $q'_v = q_v + 1$ ou $q_v > K_{v,\omega_v(q)}$ et $q'_v = q_v - 1$

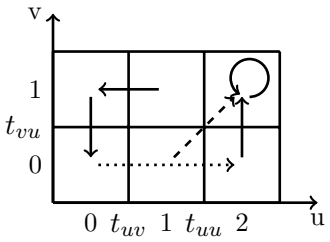

Figure 1.6: (a) synchrone

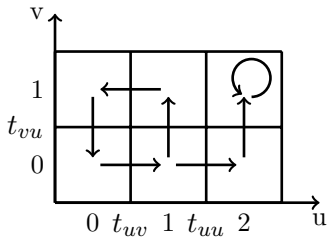

Figure 1.7: (b) asynchrone

Figure 1.8: (a) et (b) montre respectivement la dynamiques synchrone et la dynamique asynchrone du modèle de tableau 1.1. Les attracteurs sont pareils mais les chemins sont différents : le graphe d'état asynchrone contient un circuit $(0,0) \to (1,0) \to (1,1) \to (0,1) \to (0,0)$, qui est absent dans l'approche synchrone. Nous pouvons voir que dans le figure (a) il y a au plus une seule possibilité dans tous les états, c'est bien la définition de déterminisme, et que dans le figure (b) il y a 2 chemins dans l'état $(u, v) = (1, 0)$.

1.4 Process Hitting

Inspiré par le graphe d'état asynchrone, le Process Hitting donne la possibilité de simuler le système biologique pas à pas et de façon encore plus précise, possédant des caractéristiques stochastiques et temporisées [PMR11].

1.4.1 Concepts fondamentaux

Definition 1.6 (Process Hitting)**.** Un Process Hitting PH est constitué d'un triplet $(\Sigma, L, \mathscr{H})$ avec:

–$\Sigma = \{a, b, ...\}$ est l'ensemble fini des sortes

–$L = \prod_{a\in\Sigma} L_a$ est un ensemble d'états avec $L_a = \{a_0, ...a_{l_a}\}$ l'ensemble fini des processus de la sorte $a \in \Sigma$ et l_a est un entier positif satisfaisant:

$$a \neq b \rightarrow \forall (a_i, b_j) \in L_a \times L_b, a_i \neq b_j$$

L'ensemble fini d'actions est défini comme suit :

$$\mathscr{H} = \{a_i \rightarrow b_j \Rsh b_k \mid (a, b) \in \Sigma^2, (a_i, b_j, b_k) \in L_a \times L_b \times L_b, b_j \neq b_k, a = b \rightarrow a_i = b_j\}$$

a_i, b_j, b_k sont notés respectivement $hitter(h)$, $target(h)$ et $bounce(h)$ d'une action h.

On note l'ensemble de tous les processus $\mathbf{Proc} = \{a_i \mid a \in \Sigma \wedge a_i \in L_a\}$.

On note la sorte d'un processus a_i par $\Sigma(a_i) = a$ et $\Sigma(h) = \{\Sigma(hitter(h)), \Sigma(target(h))\}$ pour noter l'ensemble des sortes présentes dans une action $h \in H$. Étant donné un état $s \in L$, le processus de la sorte $a \in \Sigma$ présent dans s est noté par $s[a]$, qui est la i-ème coordonnée de l'état s. On a également les notations suivantes:

$$\text{si } a_i \in L_a, a_i \in s \overset{\Delta}{\Longleftrightarrow} [a] = a_j \text{ et si } ps \in p(\mathbf{Proc}), ps \subseteq s \overset{\Delta}{\Longleftrightarrow} \forall a_i \in ps, a_i \in s$$

Une action $h = a_i \rightarrow b_j \Rsh b_k \in \mathscr{H}$ est dite jouable dans $s \in L$ si et seulement si $s[a] = a_i$ et $s[b] = b_j$. Dans ce cas-là, $(s \cdot h)$ représente l'état issu de l'exécution de l'action h dans s, i.e. $c[b] = b_k$ et évidemment que $\forall c \in \Sigma, c \neq b, (s \cdot h)[c] = s[c]$, et que $((s \cdot h) \cdot h') = (s \cdot h \cdot h')$.

Definition 1.7 (Contexte)**.** Un contexte ς associe chaque sorte dans Σ à un sous-ensemble non-vide de ses propres processus : $\forall a \in \Sigma, c[a] \subseteq L_a \wedge c[a] \neq \varnothing$.

On note l'ensemble des contextes par **Ctx**.

Definition 1.8 (Scénario)**.** Etant donné un Process Hitting $PH = (\Sigma, L, \mathscr{H})$, un scénario $\delta = h_1 :: ... :: h_n$ est une séquence des actions dans H qui peuvent être tirées l'une par l'autre.

L'ensemble de tous les scénarios est noté par **Sce**.

Definition 1.9 (Action liée)**.** Pour un certain processus a_i, une action avec son target a_i est dite liée à a_i, qui prend la forme $b_k \to a_j \Rsh a_i$.

L'ensemble des actions liées du processus p est noté $\mathbf{LA}(p)$.

Étant donné une séquence des actions, on utilise $fst_a(A)$ pour noter le premier processus de la sorte a présent dans la séquence, et $last_a(A)$ pour noter le dernier processus.

$$fst_a(A) \triangleq \begin{cases} \varnothing & \text{si } a \notin \Sigma(A) \\ hitter(A_m) & \text{si } m = min\{n \in \mathrm{II}^A \mid a \in \Sigma(A_n) \wedge \Sigma(hitter(A_m)) = a\} \\ target(A_m) & \text{si } m = min\{n \in \mathrm{II}^A \mid a \in \Sigma(A_n) \wedge \Sigma(target(A_m)) = a\} \end{cases}$$

$$last_a(A) \triangleq \begin{cases} \varnothing & \text{si } a \notin \Sigma(A) \\ bounce(A_m) & \text{si } m = max\{n \in \mathrm{II}^A \mid a \in \Sigma(A_n) \wedge \Sigma(bounce(A_m)) = a\} \\ hitter(A_m) & \text{si } m = max\{n \in \mathrm{II}^A \mid a \in \Sigma(A_n) \wedge \Sigma(hitter(A_m)) = a\} \end{cases}$$

Où $\mathrm{II}^A \triangleq \{1, \ldots, |A|\}$ est l'ensemble des indexes de A.

Un scénario $\delta \in \mathbf{Sce}$ est dit jouable dans le contexte ς si et seulement si $support(\delta) \subseteq \varsigma$. L'exécution de δ dans ς est noté par ς où $\varsigma \cdot \delta = \varsigma \cap end(\delta)$.

Les fonctions $support(\delta)$ et $end(\delta)$ donnent respectivement le premier et le dernier processus de chaque sorte pendant la simulation:

$$support(\delta) \triangleq \{p \in \mathbf{Proc} \mid \Sigma(p) \in \Sigma(\delta) \wedge p = fst_{\Sigma(p)}(\delta)\}$$
$$end(\delta) \triangleq \{p \in \mathbf{Proc} \mid \Sigma(p) \in \Sigma(\delta) \wedge p = last_{\Sigma(p)}(\delta)\}$$

Exemple : Le Figure représente un Process Hitting $(\Sigma, L, \mathscr{H})$ où

$$\Sigma = \{a, b, c, d\}$$

$$L = \{a_0, a_1\} \times \{b_0, b_1, b_2\} \times \{c_0, c_1\} \times \{d_0, d_1, d_2\}$$

$$\mathscr{H} = \{a_0 \to c_0 \Rsh c_1, a_1 \to b_1 \Rsh b_0, c_1 \to b_0 \Rsh b_1, b_1 \to a_0 \Rsh a_1, b_0 \to d_0 \Rsh d_1,$$
$$b_1 \to d_1 \Rsh d_2, d_1 \to b_0 \Rsh b_2, c_1 \to d_1 \Rsh d_0, b_2 \to d_0 \Rsh d_2\}$$

$$\mathbf{LA}(d_2) = \{b_1 \to d_1 \Rsh d_2, b_2 \to d_0 \Rsh d_2\},\ \mathbf{LA}(a_0) = \varnothing$$

Étant donné le contexte

$$\varsigma = \langle a_0, b_0, b_1, c_0, d_1 \rangle$$

Avec

$$support(\delta) = \langle a_0, b_1, c_0, d_1 \rangle$$

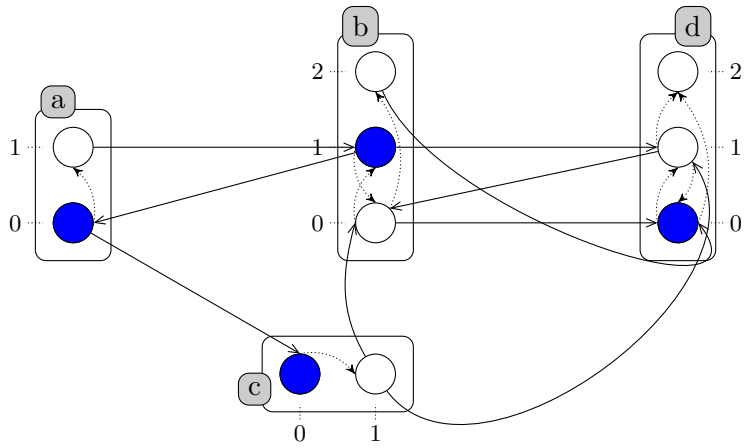

Figure 1.9: Exemple de Process Hitting

Et le scénario

$$\delta = a_0 \to c_0 \Rsh c_1 :: b_1 \to a_0 \Rsh a_1 :: b_1 \to d_1 \Rsh d_2$$

Si δ est jouable dans ς, on a :

$$end(\delta) = c \cdot \delta\langle a_0, b_0, b_1, c_0, d_1\rangle \cap \{a_1, b_1, c_1, d_2\} = \langle a_1, b_1, c_1, d_2\rangle$$

1.4.2 Outil Pint

PINT est un ensemble de commandes en ligne, qui implémente les analyses variées de Process Hitting. Parmi les commande en ligne, `ph-reach` sert à l'accessibilité et `ph-exec` sert à la simulation d'évolution d'un réseau Process Hitting, qui sont importants dans cette recherche. Les données de réseaux Process Hitting sont enregistrées dans un fichier `.ph`. Voici l'exemple d'un fichier `.ph` [PCF+14] :

```
(* Declaration of sorts *)
process a 1 process b 2 process c 1 process d 2

(*Definition of actions*)
a 0 -> c 0 1 a 1 -> b 1 0 c 1 -> b 0 1
b 1 -> a 0 1 b 0 -> d 0 1 b 1 -> d 1 2
d 1 -> b 0 2 c 1 -> d 1 0 b 2 -> d 0 2

initial_state
a 1, b 0, c 0,d 1
```

En exécutant `ph-reach -i <filename> d 2`, on a une valeur retournée `false` impliquant que le processus d_2 n'est pas accessible.

En exécutant `ph-exec -i <filename> 10`, on obtient les fichiers des informations de la stimulation par étape dans une période de 10 unités de temps.

Chapter 2

Causalité locale

Il arrive souvent que nous nous intéressons à l'accessibilité d'un certain processus, mais la complexité d'un raisonnement exhaustive est assez importante (comme montré dans la Section 1.4.3, l'explosion d'espace d'états). C'est pour ceci nous avons besoin d'une structure pour raisonner de façon non-exhaustive et assez précise.

Selon *Loc Paulevé* [PAK13], en analysant séparément(e.g. localement) et statiquement les processus, il est possible de réduire la complexité de calcul par rapport au calcul exhaustif.

Étant donné un Process Hitting $(\Sigma, L, \mathscr{H})$, pour tout $a \in \Sigma$, $S(a) = [1; l_a]$, est l'ensemble fini des états locaux de l'automate a; $S = \prod_{a\in\Sigma}[1; l_a]$ est l'ensemble fini des états globaux.

On note l'ensemble des états locaux $\mathbf{LS} \triangleq \{a_i \mid a \in \Sigma \wedge i \in [1; l_a]\}$

Definition 2.1 (Accessibilité de processus)**.** Étant donné un Process Hitting $(\Sigma, L, \mathscr{H})$ et un contexte ς, l'état local $a_j \in \mathbf{LS}$ est accessible à partir de ς si et seulement si $\exists s_0, \cdots, s_m \in S$ tel que $\forall a \in \Sigma, s_0(a) \in \varsigma(a)$, et $s_0 \to \cdots \to s_m$, et $s_m(a) = j$.

Dans une sorte a, l'accessibilité globale de a_j à partir de a_i peut être représentée par celle de a_j à partir d'un état local $a_i \in \varsigma(a)$. Cette accessibilité locale se réfère à un objectif noté $a_i \Rsh^* a_j$.

Definition 2.2 (Objectif)**.** L'accessibilité d'un état local a_j depuis a_i est appelé un objectif noté $a_i \to a_j$. L'ensemble de tous les objectifs est défini comme $\mathbf{Obj} \triangleq \{a_i \to a_j \mid (a_i, a_j) \in \mathbf{LS} \times \mathbf{LS}\}$

Étant donné un objectif $P = a_i \to^* a_j \in \mathbf{Obj}$, on définit $\mathsf{sol}(P)$ comme la causalité locale de P : chaque élément $ls \in \mathsf{sol}(P)$ est un sous-ensemble d'états locaux, référé à une solution (locale) de P, qui sont des fois plus faciles à obtenir que l'accessibilité de a_j. $\mathsf{sol}(P)$ est solide si la désactivation d'au moins un état local dans chaque solution rend l'accessibilité de a_j impossible depuis tout état global contenant a_i. Remarquons que si $\mathsf{sol}(P) = \{ls^1, \ldots, ls^m\}$ est solide, alors $\mathsf{sol}'(P) = \{ls^1, \ldots, ls^m\}$ est aussi solide. $\mathsf{sol}(a_i \to^* a_j) = \varnothing$ implique que a_j ne peut pas être atteint et $\forall a_i \in \mathbf{LS}, \mathsf{sol}(a_i \to^* a_j) \triangleq \{\varnothing\}$

Definition 2.3 (Solution)**.** $\mathbf{Obj} \to \mathcal{P}(\mathcal{P}(\mathbf{LS}))$ est un mappage d'objectifs à ensembles d'ensembles

des états locaux tels que $\forall P \in \mathbf{Obj}, \forall ls \in \mathsf{sol}(P), \nexists ls' \in \mathsf{sol}(P), ls \neq ls'$ tel que $ls' \subset ls$. L'ensemble de ces mappages est noté $\mathbf{Sol} \triangleq \{\langle P, ls\rangle \mid ls \in \mathsf{sol}(P)\}$.

2.1 Graphe de causalité locale

Avec les définitions préliminaires, nous pouvons maintenant construire une structure abstraite sans toucher les états globaux afin de réduire la complexité du calcul de l'accessibilité d'un certain processus [PAK13].

Definition 2.4 (GLC)**.** Étant donné un contexte ς et un ensemble d'états locaux $\omega \subseteq \mathbf{LS}$, le Graphe de Causalité Locale (Graph of Local Causality en anglais) $\mathcal{A}_\varsigma^\omega \triangleq (V_\varsigma^\omega, E_\varsigma^\omega)$, avec $V_\varsigma^\omega \subseteq \mathbf{LS} \cup \mathbf{Obj} \cup \mathbf{Sol}$ et $E_\varsigma^\omega \subseteq V_\varsigma^\omega \times V_\varsigma^\omega$ est la plus petite structure satisfaisant :

$$\begin{aligned}
\omega &\subseteq V_\varsigma^\omega \\
a_i \in V_\varsigma^\omega \cap \mathbf{LS} &\Leftrightarrow \{(a_i, a_j \rightarrow^* a_i) \mid a_j \in \varsigma\} \subseteq E_\varsigma^\omega \\
a_i \rightarrow^* a_j \in V_\varsigma^\omega \cap \mathbf{Obj} &\Leftrightarrow \{(a_i \rightarrow^* a_j, \langle a_i \rightarrow^* a_j, ls\rangle) \mid \langle a_i \rightarrow^* a_j, ls\rangle \in \mathbf{Sol}\} \subseteq E_\varsigma^\omega \\
\langle P, ls\rangle \in V_\varsigma^\omega \cap \mathbf{Sol} &\Leftrightarrow \{(\langle P, ls\rangle, a_i) \mid a_i \in ls\} \subseteq E_\varsigma^\omega
\end{aligned}$$

Le graphe de causalité locale (GLC, graph of local causality en anglais) réalise cette raisonnement récursif à partir d'un ensemble donné des états locaux $\omega \in \mathbf{LS}$ en reliant chaque état local a_j à tous les objectifs $a_i \nearrow^* a_j$, $a_i \in \varsigma(a)$, et en reliant chaque objectif P à leurs solutions $\langle P, ls\rangle \in \mathbf{Sol}$, et en reliant toutes les solutions $\langle P, ls\rangle$ à leurs états locaux $b_k \in ls$. Un GLC est dit valide si sol est solide pour tous les objectifs apparus.

2.2 Sur-approximation

Avant d'entrer dans cette partie, il y a quelques notions importantes :

Definition 2.5 (Séquence de bounce(**BS**))**.** La séquence de bounces ζ est une séquence des actions telle que $\forall n \in \mathrm{II}^\zeta, n < |\zeta|, bounce(\zeta_n) = target(\zeta_{n+1})$. On l'utilise pour décrire l'ensemble des séquence de bounces, et **BS**(P) pour noter l'ensemble de séquence de bounces qui résolvent l'objectif P :

$$\begin{aligned}
\mathbf{BS}(a_i \rightarrow^* a_j)\{\zeta \in \mathbf{BS} \mid\ & target(\zeta_1) = a_i \wedge bounce(\zeta_{|\zeta|}) = a_j \\
& \wedge \forall m, n \in \mathrm{II}^\zeta, n > m, bounce(\zeta_n) \neq target(\zeta_m)\}
\end{aligned}$$

Bien entendu, on a

$$\mathbf{BS}(a_i \rightarrow^* a_i) = \{\varepsilon\},$$

et

$$\mathbf{BS}(a_i \rightarrow^* a_j) = \varnothing$$

s'il est impossible d'accéder de a_i à a_j.

Avec la définition de séquence de bounce, on peut maintenant définir $\mathbf{BS}^\wedge$ qui représente la liste de processus nécessaire pour que cet objectif soit réalisable.

Definition 2.6 ($\mathbf{BS}^\wedge : \mathbf{Obj} \mapsto \mathcal{P}(\mathbf{Proc})$)**.**

$$\mathbf{BS}^\wedge(P) \triangleq \{\zeta^\wedge \mid \zeta \in \mathbf{BS}(P) \wedge \nexists \zeta' \in \mathbf{BS}(P), \zeta'^\wedge \subsetneq \zeta^\wedge\}$$

où $\zeta^\wedge \triangleq \{hitter(\zeta_n) \mid n \in \Pi^\zeta \wedge \Sigma(hitter(\zeta_n)) \neq \Sigma(P)\}$

Exemple :

Dans le Figure 1.9, pour réaliser l'objectif $b_1 \rightarrow^* b_2$, il faut tirer successivement les actions $a_1 \rightarrow b_1 \not\rightarrow b_0$ et $d_1 \rightarrow b_0 \not\rightarrow b_2$, c'est exactement $\mathbf{BS}^\wedge(b_1 \rightarrow^* b_2) = \{\{a_1, d_1\}\}$.

La sur-approximation s'applique le GLC qui est une condition suffisante de l'accessibilité d'un certain processus. Sa structure abstraite est notée $\mathcal{A}^\omega_\varsigma$ [PMR12] :

Definition 2.7 ($\mathcal{A}^\omega_\varsigma$)**.** Étant donné un contexte ς et une séquence d'objectifs,

$$\mathcal{A}^\omega_\varsigma = (Req^\omega_\varsigma, Sol^\omega_\varsigma, Cont^\omega_\varsigma)$$

où Req^ω_ς, Sol^ω_ς et $Cont^\omega_\varsigma$ sont définis comme suit :

$$Req^\omega_\varsigma \triangleq \{(a_i, a_j \rightarrow^* a_i) \in \mathbf{Proc} \times \mathbf{Obj} \mid a_j \in \varsigma[a] \wedge (\exists (P, ps) \in Sol^\omega_\varsigma, a_i \in ps \vee \exists n \in \Pi^\omega, bounce(\omega) = a_i)\}$$

$$Sol^\omega_\varsigma \triangleq \{(P, ps) \in \mathbf{Obj} \times \mathcal{P}(\mathbf{Proc}) \mid \exists (a_i, P) \in Req^\omega_\varsigma \wedge ps \in \mathbf{BS}^\wedge(P)\}$$

$$Cont^\omega_\varsigma \triangleq \{(P, Q) \in \mathbf{Obj} \times \mathbf{Obj} \mid \exists (P, ps) \in Sol^\omega_\varsigma \wedge Q \in minCont_\varsigma(P)\}$$

où

Definition 2.8 ($minCont_\varsigma : \mathbf{Obj} \mapsto \mathcal{P}(\mathbf{Obj})$)**.**

$$minCont_\varsigma(\overset{\bigstar}{\rightarrow}^* a_j) \triangleq \{a_k \rightarrow^* a_j \mid a_k \neq a_j \wedge \forall a_i \in \varsigma[a], a_k \in minCont^{Obj}_\varsigma(a, a_i \rightarrow^* a_j)\}$$

$minCont^{Obj}_\varsigma : \Sigma \times \mathbf{Obj} \mapsto \mathcal{P}(\mathbf{Proc})$

$$minCont^{Obj}_\varsigma(a, P) \triangleq \begin{cases} \varnothing & \text{si } \mathbf{BS}^\wedge(P) = \varnothing \\ \{p \in \mathbf{Proc} \mid \forall ps \in \mathbf{BS}^\wedge(P), & \\ \exists q \in ps, p \in minCont^{Proc}_\varsigma(a, q)\} & \text{sinon} \end{cases}$$

$minCont^{Obj}_\varsigma : \Sigma \times \mathbf{Obj} \mapsto \mathcal{P}(\mathbf{Proc})$

$$minCont_{\varsigma}^{Proc}(a, b_i) \triangleq \begin{cases} \{b_1\} & \text{si } a = b \\ \{p \in \mathbf{Proc} \mid \forall b_j \in \varsigma[b], & \\ p \in ps, p \in minCont_{\varsigma}^{Obj}(a, b_j \rightarrow^* b_i)\} & \text{sinon} \end{cases}$$

où ★ signifie un processus de certaine sorte à niveau quelconque.

Dans le Figure 2.1, la structure abstraite $\mathcal{A}_{\varsigma}^{\omega}$ commence par un nœud de processus (carré), il relie les objectifs associés et chaque objectif relie un nœud de solution (cercle), enfin, les nœuds de solutions relient ses processus composants. Les $\perp$ signifient l'inaccessibilité de cette branche [PMR12].

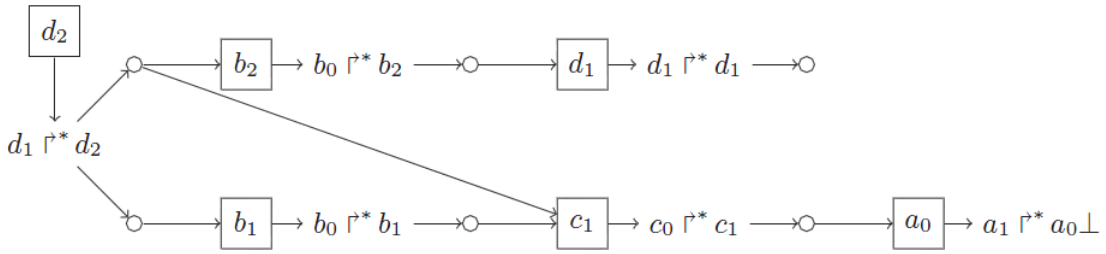

Figure 2.1: La structure abstraite $\mathcal{A}_{\varsigma}^{\omega}$ de l'exemple de Process Hitting dans le Figure 1.9 avec $\omega = d_1 \Rsh^* d_2$ et $\varsigma = \langle a_1, b_0, c_0, d_1 \rangle$, cet objectif n'est pas réalisable.

2.3 Sous-approximation

De façon analogue, la sous-approximation est basée sur la structure abstraite $\lceil \mathcal{B}_{\varsigma}^{\omega} \rceil$:

Definition 2.9 ($maxCont_{\varsigma} : \Sigma \times \mathbf{Obj} \mapsto \mathcal{P}(\mathbf{Proc})$)**.**

$$maxCont_{\varsigma}(a, P) \triangleq \{p \in \mathbf{Proc} \mid \exists ps \in \mathbf{BS}^{\wedge}(P), \exists b_i \in ps, b = a \wedge p = b_i \\ \vee\, b \neq a \wedge p \in maxCont_{\varsigma}(a, b_j \rightarrow^* b_i) \wedge b_j \in \varsigma[b]\}$$

Definition 2.10 ($\lceil \mathcal{B}_{\varsigma}^{\omega} \rceil$)**.** La structure abstraite

$$\lceil \mathcal{B}_{\varsigma}^{\omega} \rceil = (\lceil Req_{\varsigma}^{\omega} \rceil, \lceil Sol_{\varsigma}^{\omega} \rceil, \lceil Cont_{\varsigma}^{\omega} \rceil)$$

est définie comme :

$$\lceil \mathcal{B}_{\varsigma}^{\omega} \rceil \triangleq pppf\{\mathcal{B}_{\varsigma}^{\omega}\}^1$$

avec

$$\mathcal{B}_{\varsigma}^{\omega} = (\overline{Req_{\varsigma}^{\omega}}, \overline{Sol_{\varsigma}^{\omega}}, \overline{Cont_{\varsigma}^{\omega}})$$

avec

$$\overline{Req_{\varsigma}^{\omega}} \triangleq \{(a_i, a_j \rightarrow^* a_i) \in \mathbf{Proc} \times \mathbf{Obj} \mid a_j \in \varsigma[a] \wedge (\exists (P, ps) \in \overline{Sol_{\varsigma}^{\omega}}, a_i \in ps \\ \vee\, \exists n \in \Pi^{\omega}, bounce(\omega) = a_i)\}$$

[1] pppf : le plus petit point fixe

$$\overline{Sol^{\omega}_{\varsigma}} \triangleq \{(a_i, a_j \rightarrow^* a_i) \in \mathbf{Obj} \times \mathcal{P}(\mathbf{Proc}) \mid \exists (a_i, P) \in Req^{\omega}_{\varsigma} \wedge ps \in \mathbf{BS}^{\wedge}(\mathbf{P})\}$$

$$\overline{Cont^{\omega}_{\varsigma}} \triangleq \{(a_i, a_j \rightarrow^* a_i) \in \mathbf{Obj} \times \mathbf{Obj} \mid \exists (P, ps) \in Sol^{\omega}_{\varsigma} \wedge q \in maxCont_{\varsigma}(\Sigma(P), P)\}$$

Avec ces 2 approximations, l'accessibilité peut être résolue dans la plupart de cas sans gros calcul (non-exhaustive), comme cette approche ne touche ni la simulation ni les états globaux.

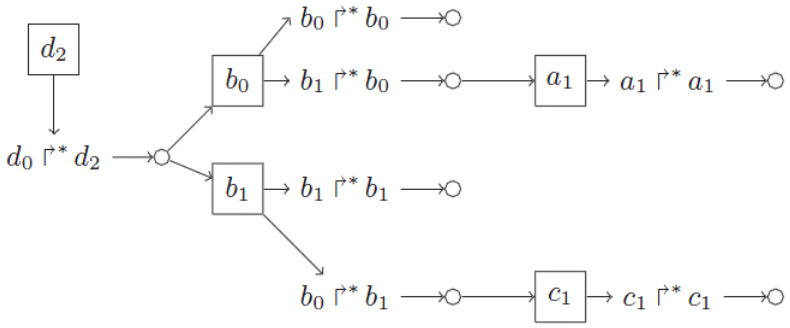

Figure 2.2: La structure abstraite $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$ de l'exemple de Process Hitting dans le Figure 1.9 avec $\omega = d_0 \Rsh^* d_2$ et $\varsigma = \langle a_1, b_1, c_1, d_0 \rangle$, cet objectif est réalisable.

2.4 Cut set

Le cut set une approche contraire à la complétion, qui sert à empêcher définitivement l'accessibilité de certain processus. Cette méthode calcule les cut sets directement d'une structure abstraite dérivée du modèle, qui rend l'analyse des grands réseaux traitables [PAK13].

Définition préliminaires

Supposons un GLC solide global $\mathcal{A}^{\omega}_{\varsigma} = (V^{\omega}_{\varsigma}, E^{\omega}_{\varsigma})$, avec les accesseurs habituels pour les relations directes de nœuds :

$$\mathsf{children} : V^{\omega}_{\varsigma} \mapsto \mathcal{P}(V^{\omega}_{\varsigma}) \qquad \mathsf{parents} : V^{\omega}_{\varsigma} \mapsto \mathcal{P}(V^{\omega}_{\varsigma})$$
$$\mathsf{children}(n) \triangleq \{m \in V^{\omega}_{\varsigma} \mid (n, m) \in E^{\omega}_{\varsigma}\} \qquad \mathsf{parents}(n) \triangleq \{m \in V^{\omega}_{\varsigma} \mid (m, n) \in E^{\omega}_{\varsigma}\}$$

En partant d'une valuation $\mathbb{V}$, nous associons chaque nœud à un ensemble de cut N-ensemble, l'ensemble des cut N-ensembles peut être raffiné par $\mathsf{update}(\mathbb{V}, n)$:

- si n est une solution $\langle P, ls \rangle \in \mathbf{Sol}$, il est suffisant d'empêcher l'accessibilité de tout état local ls; par conséquent, le cut N-ensemble est issu de l'union des ensembles des $\mathsf{children}$ de n.

- si n est un objectif $P \in \mathbf{Obj}$, tous ses solutions (dans P) doivent être enlevées afin que P ne soit pas réalisable : alors les cut N-ensembles sont issus des cut N-ensembles des $\mathsf{children}$.
- si n est un état local a_i, il est suffisant d'empêcher tous ses $\mathsf{children}$(objectifs) pour empêcher l'accessibilité de a_i à partir de tout état dans le contexte ς. Au fait, si $a_i \in \mathcal{O}$, $\{a_i\}$ sera rajouté dans l'ensemble de ses cut N-ensembles.

Definition 2.11 (Valuation $\mathbb{V}$)**.** Une valuation $\mathbb{V} : V_\varsigma^\omega \mapsto \mathcal{P}(\mathcal{P}^{\leq N}(\mathcal{O}))$ est un mappage de chaque nœud de $\mathcal{A}_\varsigma^\omega$ à un ensemble de N-ensemble d'états locaux. **Val** est l'ensemble de toutes les valuations. $\mathbb{V}_0 \in \mathbf{Val}$ se réfère à la valuation telle que $\forall n \in V_\varsigma^\omega, \mathbb{V}_0(n) = \varnothing$.

Definition 2.12 ($\mathsf{update} : \mathbf{Val} \times V_\varsigma^\omega \mapsto \mathbf{Val}$)**.**

$$\mathsf{update}(\mathbb{V}, n) \triangleq \begin{cases} \mathbb{V}\{n \mapsto \zeta^N(\bigcup_{m \in \mathsf{children(n)}} \mathbb{V}(m))\} & \text{si } n \in \mathbf{Sol} \\ \mathbb{V}\{n \mapsto \zeta^N(\prod_{m \in \mathsf{children(n)}} \mathbb{V}(m))\} & \text{si } n \in \mathbf{Obj} \\ \mathbb{V}\{n \mapsto \zeta^N(\prod_{m \in \mathsf{children(n)}} \mathbb{V}(m))\} & \text{si } n \in \mathbf{LS} \setminus \mathcal{O} \\ \mathbb{V}\{n \mapsto \zeta^N(\{\{a\}\} \cup \prod_{m \in \mathsf{children(n)}} \mathbb{V}(m))\} & \text{si } n \in \mathbf{LS} \cap \mathcal{O} \end{cases}$$

avec $\zeta^N(\{e_1, \ldots, e_n\}) \triangleq \{e_i \mid i \in [1; n] \wedge \#e_i \leq N \wedge \nexists j \in [1; n], j \neq i, e_j \subset e_i\}$ où e_i sont des ensembles, et $\forall i \in [1; n]$.

À partir de $\mathbb{V}_0$, nous pouvons appliquer à maintes reprises update pour tout nœud de $\mathcal{A}_\varsigma^\omega$ pour raffiner sa valuation. Seuls les nœuds où l'un de leurs $\mathsf{children}$ ont changé de valeur doivent être prise en compte pour update.

Étant donné un ensemble d'états locaux $\mathcal{O} \subseteq \mathbf{LS}$, cette section dédie à un algorithme du calcul de l'ensemble $\mathbb{V}(a_i)$ à partir de $\mathcal{A}_\varsigma^\omega$.
où $\mathsf{rank}(n)$ se réfère au rang topologique de n.

$\mathcal{M} \leftarrow V_\varsigma^\omega$
$\mathbb{V} \leftarrow \mathbb{V}_0$
tant que $\mathcal{M} \neq \varnothing$ **faire**
  $n \leftarrow \text{argmin}_{m \in \mathcal{M}}\{\mathsf{rang}(m)\}$
  $\mathcal{M} \leftarrow \mathcal{M} \setminus \{n\}$
  $\mathbb{V}' \leftarrow \mathsf{update}(\mathbb{V}, n)$
  si $\mathbb{V}'(n) \neq \mathbb{V}(n)$ **alors**
    $\mathcal{M} \leftarrow \mathcal{M} \cup \mathsf{parents}(n)$
  fin
  $\mathbb{V} \leftarrow \mathbb{V}'$
fin
retourner $\mathbb{V}$

Une fois le GLC est établi, il est possible de calculer le cut set, de complexité moins importante que le parcours global. Sous l'aspect des actions, il est suffisant d'empêcher certain processus en enlevant les actions liées d'un des cut sets.

Exemple:

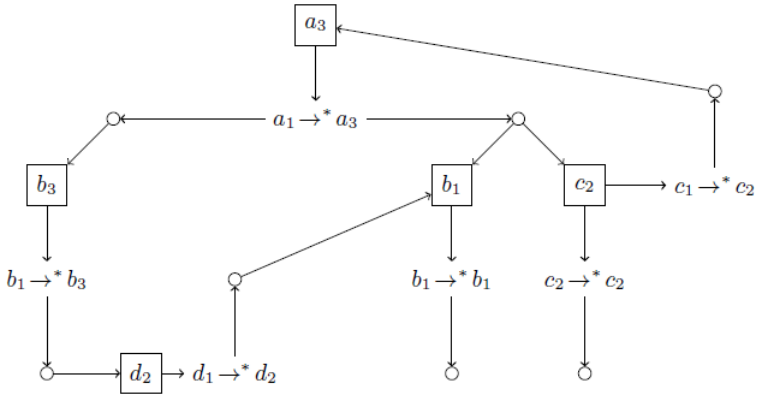

Figure 2.3: Exemple d'un GLC avec son objectif $a_1 \rightarrow^* a_3$

Nœud	rang	$\mathbb{V}$
$\langle b_1 \rightarrow^* b_1, \varnothing \rangle$	1	$\varnothing$
$b_1 \rightarrow^* b_1$	2	$\varnothing$
b_1	3	$\{\{b_1\}\}$
$\langle d_1 \rightarrow^* d_2, \{b_1\} \rangle$	4	$\{\{b_1\}\}$
$d_1 \rightarrow^* d_2$	5	$\{\{b_1\}\}$
d_2	6	$\{\{b_1\}, \{d_2\}\}$
$\langle b_1 \rightarrow^* b_3, \{d_2\} \rangle$	7	$\{\{b_1\}, \{d_2\}\}$
$b_1 \rightarrow^* b_3$	8	$\{\{b_1\}, \{d_2\}\}$
b_3	9	$\{\{b_1\}, \{b_3\}, \{d_2\}\}$
$\langle a_1 \rightarrow^* a_3, \{b_3\} \rangle$	10	$\{\{b_1\}, \{b_3\}, \{d_2\}\}$
$\langle c_2 \rightarrow^* c_2, \varnothing \rangle$	11	$\varnothing$
$c_2 \rightarrow^* c_2$	12	$\varnothing$
c_2	13	$\{\{c_2\}\}$
$\langle a_1 \rightarrow^* a_3, \{b_1, c_2\} \rangle$	13	$\{\{b_1\}, \{c_2\}\}$
$a_1 \rightarrow^* a_3$	13	$\{\{b_1\}, \{b_3, c_2\}, \{c_2, d_2\}\}$
a_3	13	$\{\{a_3\}, \{b_1\}, \{b_3, c_2\}, \{c_2, d_2\}\}$
$\langle c_2 \rightarrow^* c_2, \{a_3\} \rangle$	13	$\{\{a_3\}, \{b_1\}, \{b_3, c_2\}, \{c_2, d_2\}\}$

Table 2.1: Résultat de l'exécution de l'Algorithme 1 sur le GLC du Figure 2.4, dans la colonne de $\mathbb{V}$, il est suffisant d'empêcher un des ensemble de processus pour rendre certain objectif non-réalisable, e.g.$\{\{a_3\}, \{b_1\}, \{b_3, c_2\}, \{c_2, d_2\}\}$ veut dire réalisabilité de $(a_1 \rightarrow^* a_3) = \neg(a_3 \vee b_1 \vee (b_3 \wedge c_2) \vee (c_2 \wedge d_2))$

Chapter 3

Préparation avant complétion

Avant de nous engager la complétion, il est important de préciser quelle partie à compléter : les différences entre l'observation et la simulation. Voici les deux cas possibles,

1. l'absence de certain processus dans l'observation et la présence de ce processus dans la simulation
2. la présence de certain processus dans l'observation et l'absence de ce processus dans la simulation

Dans le cas 1, il existe des actions en trop dans le modèle, nous pouvons appliquer le cut set afin de trouver les actions erronées. Dans le cas 2, il manque quelques actions. Pour les rajouter, les démarches seront présentées dans le chapitre 4 Complétion.

Quant à l'accessibilité, on tient compte de 2 aspects : l'efficacité et l'exactitude, mais ces 2 aspects se contredisent souvent l'un l'autre. Dans la section suivante, deux méthodes seront présentées.

3.1 Méthode non-exhaustive

À l'aide de la commande `ph-reach` de PINT (Analyse de sur-approximation et sous-approximation), on a 3 valeurs possibles de l'accessibilité : non-accessible (non pour sur-approximation), inconclusive (oui pour sur-approximation et non pour sous-approximation, ce qui nous ne mène pas à un résultat définitif), accessible (oui pour sous-approximation). En fait, il arrive rarement que les résultats soient inconclusifs, c'est-à-dire nous pouvons déterminer dans la plupart de cas l'accessibilité de certain processus [PMR12].

3.2 Méthode exhaustive

Nous fournissons également une autre approche en cas d'inconclusivité et elle est capable d'analyser non seulement l'accessibilité d'un processus mais la réalisabilité d'une séquence stricte. Cette ap-

Sur-approximation	Sous-approximation	Accessibilité
vrai	vrai	vrai
vrai	faux	inconclusive
faux	vrai	N/A
faux	faux	faux

Table 3.1: Table de vérité de l'accessibilité

proche est de parcourir toutes les branches de transitions de Process Hitting qui donne définitivement l'accessibilité, mais avec une complexité importante théorique. Intuitivement, grâce aux nombreux conditions de sortir, la complexité dans la plupart de cas deviendrait moins importante.

3.2.1 Définitions Préliminaires

Definition 3.1 (Observabilité)**.** Une sorte est considérée observable si ses données temporisées sont disponibles pendant l'expérience, et toutes les sortes sont attribuées inobservable par défaut.

Definition 3.2 (Séquence)**.** Une séquence est composée de processus dans l'ordre d'occurrence. Une séquence est dite réalisable s'il y a un scénario dans lequel les processus sont de même ordre d'occurrence que la séquence.

Definition 3.3 (Séquence stricte)**.** Une séquence est stricte si elle contient toutes les évaluations des sortes observables, et les sortes inobservables ne doivent pas être dans une séquence stricte.

Definition 3.4 (Ensemble d'actions valables)**.** Cet ensemble désigne toutes les actions tirables dans l'état actuel, noté $A = \{a_i \to b_j \Rsh b_k \mid s[a] = a_i \wedge s[b] = b_j\}$

Definition 3.5 (Opérateur override $\Cap$: $\mathbf{Ctx} \times \mathcal{P}(\mathbf{Proc}) \mapsto \mathbf{Ctx}$)**.** Étant donné un contexte $\varsigma \in \mathbf{Ctx}$ et $ps \in \mathcal{P}(\mathbf{Proc})$, l'override de ς par ps est noté $\varsigma \Cap ps$:

$$\forall a \in \Sigma, (\varsigma \Cap ps)[a] \triangleq \begin{cases} \{p \in ps \mid \Sigma(p) = a\} & si\ \exists p \in ps, \Sigma(p) = a \\ \varsigma[a] & sinon \end{cases}$$

Exemple de l'opérateur $\Cap$: $\langle a_1, a_2, b_1, c_1\rangle \Cap \{a_3, b_2, b_3\} = \langle a_3, b_2, b_3, c_1\rangle$.

3.2.2 Description du problème

Étant donné un Process Hitting $PH = (\Sigma, L, \mathscr{H})$, un contexte ς (On tient compte des états initiaux de toutes les sortes (y compris ceux des sortes inobservables)), une séquence stricte probablement non réalisable pour l'instant : $S = p_1 :: p_2 :: \ldots :: p_{|S|}$, nous voulons classer le(les) processus inaccessible(s) pour la prochaine démarche (la complétion).

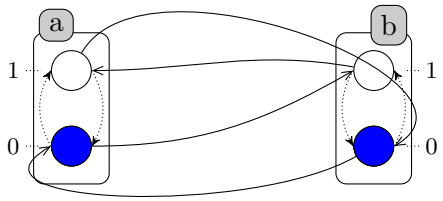

Figure 3.1: Exemple de séquence stricte: Il y a 4 actions: $\mathscr{H} = \{a_0 \to b_1 \Rsh b_0,\ a_1 \to b_0 \Rsh b_1,\ b_0 \to a_0 \Rsh a_1,\ b_1 \to a_1 \Rsh a_0\}$, avec l'état initial $\varsigma = \langle a_0, b_0 \rangle$, supposons les sortes a et b sont observables, la sequence $S_1 = a_1 :: b_1$ est stricte alor que la sequence $S_2 = b_1$ n'est pas stricte, comme le seul scénario est $\delta = b_0 \to a_0 \Rsh a_1 :: a_1 \to b_0 \Rsh b_1$...

3.2.3 Dynamique

Pour noter l'évolution de modèle, $s_0 = \varsigma$, $s_i = s_{i-1} \Cap p_i$, où i est l'indexe des étapes et les s_i sont les états globaux et les p_i sont les processus changés après transitions et A_i les actions valables à l'étape i.

- Si $Card(A_i) = 0$, il n'y a aucune action valable, le système atteint un état stable;
- Si $Card(A_i) = 1$, il n'y a qu'une seule action valable, le système va dans un seul sens;
- Si $Card(A_i) > 1$, il y a des branches dans l'évolution, et les overrides possibles du prochain instant seront enregistrés dans `futureActions`.

  Où $Card(A)$ signifie le cardinal de l'ensemble A.

3.2.4 Classification des cas

Chaque fois après tirer une action (transition), l'analyse du premier processus dans la séquence stricte S est faite comme suit:

1 Processus inaccessible (processus sans actions liées): retourner 1 dans `realisablility` puis ajouter ce processus dans `unreachableProcess1`;

2 Processus inaccessible (processus avec actions liées mais il n'y a aucune action valable ou l'état actuel est pareil que celui qui est apparu avant (boucle)): retourner 1 dans `realisablility` puis ajouter ce processus dans `unreachableProcess2`;

3 Accessible mais dans mauvais ordre : retourner 2 dans `realisablility`, puis ajouter ce processus dans `unreachableProcess2`, il y a alors 2 cas de mauvais ordre :

  I Un dernier processus dans la séquence apparaît plus tôt que ceux des anciens;

  II Un processus observable apparaît mais il n'est pas dans la séquence.

4 Processus accessible : enlever ce processus dans la séquence; lorsque la séquence devient vide, retourner 0 dans `realisablility` (cette séquence est bien réalisable);

5 S'il y a des branches dans la transition, on considère :

- I Ce processus est accessible s'il est accessible depuis au moins une des branches. (Classé dans le cas 4);
- II Il n'y a aucune branche valable pour que ce processus soit accessible :
  - i Toutes ses branches mènent au cas 3, retourner 2 dans `realisablility`
  - ii Il y a au moins une branche qui mène au cas 1 ou 2, retourner 1 dans `realisablility`

Dans la classification, il y a 3 valeurs de retour possibles, 0 indique la réalisabilité de la séquence, 1 indique que la séquence n'est pas réalisable à cause de l'inatteignabilité de certain processus, 2 indique que la séquence n'est pas réalisable non plus, mais à cause du mauvais ordre d'apparition.

Exemple de mauvais ordre : la séquence dans l'observation est $S_o = p_1 :: p_2 :: p_3$, mais selon la simulation, nous n'avons que $S_s = p_2 :: p_1 :: p_3$ comme résultat. Bien que les processus soient accessible, ils n'apparaissent pas dans un bon ordre.

Comme expliqué avant, cette approche peut être appliquée en cas d'inconclusivité de la méthode non-exhaustive. En prenant la séquence stricte avec un seul processus, L'Algorithme 4 réalise cette idée, bien que la complexité est importante, la séquence stricte la réduit efficacement en utilisant les données temporisées[1].

[1]Le code est disponible sur https://github.com/XinweiChai/Completion/tree/master/src/completion

Chapter 4

Complétion

4.1 Motivation

L'analyse des réseaux de régulation, qui comprend les réseaux génétiques, les réseaux d'interaction entre les protéines et les réseaux de signaux, est donc un des thèmes principaux en bioinformatique.

Le but est souvent : avec les informations d'expression d'une série de gènes (une série d'états de tous les gènes dans un environnement varié), on peut en déduire les fonctions avec des gènes d'entrée qui régule chaque gène, avec lesquelles un réseau génétique est formé, mais dans le cas d'un réseau de régulation, il existe toujours des données inconnues (niveaux d'activation et les relations entre des nœuds) pour certains gènes, protéines, etc, et qu'il existe aussi des donnée erronées [ATH09]. Il est indispensable de chercher les méthodes pour compléter les réseaux et pour les modifier avec afin d'avoir la dynamique complète du système. Souvent, la complétion est assez importante. Dans cet article, trois méthodes de complétion seront présentées, dont l'une est à l'aide d'un seul processus, et les deux sont à partir d'une séquence de processus.

Si on dit le cut set "coupe" la possibilité d'atteindre certain état local, alors la complétion est l'opération inverse de cut set, car elle donne la possibilité d'accéder les processus inatteignable, mais elle donne souvent de nouvelles branches qui ne sont pas issus du système original.

Exemple :

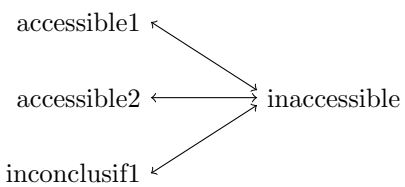

Figure 4.1: La surjection de cas accessible au cas inaccessible (cutset $\leftrightarrows$ complétion)

Prenons l'exemple de Figure 1.9, supposons $\varsigma = \langle a_0, b_0, c_0, d_1 \rangle$ et

$$\mathscr{H} = \{a_0 \to c_0 \Rsh c_1, a_1 \to b_1 \Rsh b_0, c_1 \to b_0 \Rsh b_1,$$

$$b_1 \to a_0 \Rsh a_1, b_0 \to d_0 \Rsh d_1, d_1 \to b_0 \Rsh b_2, c_1 \to d_1 \Rsh d_0\}$$

avec $b_2 \to d_0 \Rsh d_2$ et $b_1 \to d_1 \Rsh d_2$ relevées, maintenant d_2 n'est pas accessible. Pour que d_2 soit accessible, il y a une alternative entre ces deux actions, mais il n'y a pas d'informations en plus pour déterminer.

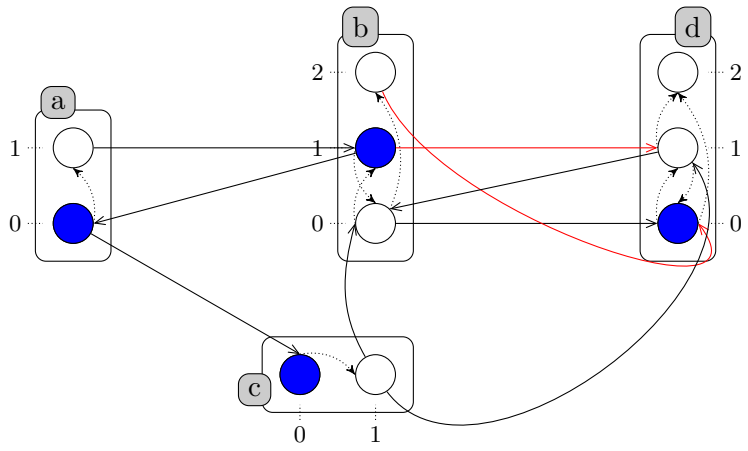

Figure 4.2: Complétion du Process Hitting en rajoutant l'action $b_2 \to d_0 \Rsh d_2$ ou $b_1 \to d_1 \Rsh d_2$

À cause de cette surjectivité, le choix définitif exige plus d'expériences et d'analyses.(Dans cet exemple, l'analyse sur b_1, b_2 ou c_1 est possible)

Les résultats de complétion diffèrent à cause de l'approche de détermination de l'accessibilité de processus (ou bien la réalisabilité de séquence stricte).

- En utilisant PINT : non-accessible → non conclusive/accessible
- En utilisant la méthode exhaustive : non-accessible → accessible

Par la suite, nous proposons deux méthodes de complétion d'un réseau Process Hitting, qui servent à ces deux cas.

4.2 Définitions préliminaires

Comme il y a des données inconnues dans le réseau de régulation, on fait des hypothèses sur les relations de régulation et les vérifie si elles sont cohérentes avec l'expérience.

Definition 4.1 (Relation de régulation)**.** L'activation ou l'inhibition de la sorte a sur la sorte b est notée $r = (a, b, \alpha_{ab})$, l'ensemble de toutes les relations de régulation est défini comme : $R = \{(a, b, \alpha_{ab}) \mid (a, b) \in \Sigma^2 \wedge \alpha_{ab} \in \{+, -\}\}$.

Definition 4.2 (Cohérence)**.** Une action $a = a_i \to b_j \Rsh b_k$ est dite *incohérente* avec R si

$$\begin{aligned}\exists r \in R, &(r = (a, b, +) \wedge ((i = l_a \wedge j > k) \vee (i = 0 \wedge j < k))) \vee \\ &(r = (a, b, -) \wedge ((i = l_a \wedge j < k) \vee (i = 0 \wedge j > k)))\end{aligned}$$

sinon, elle est considérée cohérente avec R.

Exemple :

Le Figure 1.5 (a), si les relations de régulation entre u et v restent à vérifier, c'est-à-dire $R = \{(u, v, +), (v, u, -), (u, u, +)\}$, et si on prend un formalisme de Process Hitting, l'action $v_1 \to u_1 \Rsh u_2$ n'est pas cohérente avec R.

4.3 Complétion basé sur la séquence stricte

Description du problème

Étant donné un Process Hitting PH, un ensemble de relations de régulation R et une séquence stricte S, il est suffisant d'analyser la réalisabilité de S en utilisant la méthode exhaustive. Au début, la séquence observée est souvent irréalisable pour le réseau original, nous voulons le compléter en rajoutant des actions pour que le modèle ait les mêmes comportements.

L'idée est : dans la séquence stricte $S = p_1 :: \ldots p_n$, supposons que après l'analyse de réalisabilité, le premier processus inaccessible p_i avec $1 \leq i \leq n$ a été trouvé. Ensuite, appliquer la complétion (rajouter des actions) pour que p_i soit accessible. Puis reprendre l'analyse de réalisabilité de S, un autre processus inaccessible p_j avec $i \leq j \leq n$ est trouvé. Refaire les démarches avant, jusqu'à ce que la séquence est réalisable.

Voici l'algorithme répétitif qui réalise cette idée.

Données : Un fichier `.ph` en ajoutant une séquence stricte `sequence` et un ensemble de relations de régulation `relation`
Résultat : Un Process Hitting complété
répéter
  Algorithme 3 Transition
  Algorithme 4 Realisability
  Algorithme 5 Classification
  Algorithme 6 **ou** Algorithme 7 Complétion selon `relation`
jusqu'à *`Realisability=0`*;
Algorithme 1 : Structure de programme (Algorithmes mentionnés sont dans l'annexe)

Nous aimerons détailler l'idée de Algorithme 6 et Algorithme 7, la complétion : après avoir appliqué

la méthode exhaustive et la classification des processus inaccessibles, on a plusieurs valeurs de retour possibles, l'idée de complétion est comme suit[1],

Si la valeur de retour (dans la section 3.2.4) est :

0. la séquence est bien réalisable, pas de nécessité de faire la complétion;
1. différent types de inaccessibilité de processus :
   - I `unreachableProcess1`: supposons a_j est un (ou le seul) de ses élément(s), ajouter action : $b_k \to a_j \Rsh a_i$ s'il y a une relation de régulation de forme $(b, a, \pm)$ dans R;
   - II `unreachableProcess2`: supposons a_j est un (ou le seul) de ses élément(s), et $b_k \to a_j \Rsh a_i$ est une de ses actions liées, vérifier si a_j et b_k sont dans les états actuels:
     - i si oui, il est sûr que b_k n'est pas accessible, sinon a_i sera accessible. Classer b_k (dans la Section 3.2.4) et refaire le traitement que l'on a fais à a_i ... c'est une tâche récursive sans garantie de terminaison à cause de la boucle[2], par conséquent il est nécessaire de noter tous les bounces liés à ce calcul, si on trouve la répétition de certain bounce, alors l'action liée ($b_k \to a_j \Rsh a_i$ dans ce cas-là) sera enlevée de **LA**;
     - ii si b_k est accessible, alors a_i n'est pas accessible, traiter de même façon comme i;
     - iii si ni a_i ni b_k n'est accessible, enlever $b_k \to a_j \Rsh a_i$ des actions liées de a_i, puis:
       s'il y a d'autres actions liées, essayer la complétion avec une prochaine action;
       s'il n'y a pas d'autres action liées, ajouter a_j à `unreachableProcess1` et enlever a_j de `unreachableProcess2`.
2. il y a peut-être des erreurs dans le réseau original, mais on peut quand même compléter ce réseau comme dans le cas 1;

Exemple : Un Process Hitting $PH = (\Sigma, L, \mathscr{H})$ avec $\Sigma = \{a, b\}$, $L = \{a_0, a_1\} \times \{b_0, b_1\}$, $\mathscr{H} = \{b_0 \to a_0 \Rsh a_1, a_1 \to b_0 \Rsh b_1\}$, son contexte $\varsigma = \{a_0, b_0\}$, l'ensemble de relations de régulation $R = \{(a, b, +), (b, a, -)\}$, et une séquence stricte $S = a_1 :: b_1 :: a_0 :: b_0$.

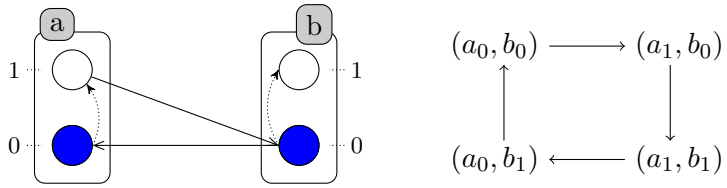

Figure 4.3: Process Hitting original et son graphe de transition déduit par la séquence strict

D'abord, $S' = a_1 :: b_1$ est bien réalisable, mais a_0 n'est pas accessible dans l'état $\{a_1, b_1\}$, selon $(b, a, -)$, $b_1 \to a_1 \Rsh a_0$ est ajoutée (cohérente). On relance la simulation et on trouve que pour l'instant, a_0 est accessible, mais b_0 n'est pas accessible dans l'état $\{a_0, b_1\}$, on rajoute $a_0 \to b_1 \Rsh b_0$ selon $(a, b, +)$, enfin, $S = a_1 :: b_1 :: a_0 :: b_0$ est réalisable après complétion.

[1] Le code est disponible sur https://github.com/XinweiChai/Completion/tree/master/src/completion

[2] par exemple les actions $b_k \to a_j \Rsh a_i$ et $a_i \to b_m \Rsh b_k$ forme une boucle, le raisonnement n'arrête jamais

En plus, pour enregistrer les nouvelles données, le fichier `.ph` est modifié en rajoutant une séquence et des relations de régulation. Les données rajoutées sont sous la forme:

```
sequence a 1 b 1 a 0 b 0
relation a b + b a -
```

4.4 Complétion basée sur le processus

Complétion basée sur le processus est un problème plus simple que celle avant, car un processus est une séquence contenant un seul élément. L'intérêt de cette section est de rendre la complexité moins importante. Dans l'algorithme de Section 4.3, la partie Algorithme 3 est exhaustive, qui parcourt toutes les branches pour vérifier leur accessibilité. Le problème de complétion sera peut-être intraitable à grande échelle.

Pour analyser l'accessibilité d'un certain processus, `ph-reach` peut remplacer l'Algorithme 3, et on fais aussi la classification des processus inaccessibles et la partie de complétion ressemble à Section 4.3, sauf il n'y a pas de possibilité de la valeur de retour 2 dans le cas de la complétion basée sur la séquence stricte.

Supposons a_i est le processus inaccessible.[3]

1 `unreachableProcess1`: ajouter action $b_k \rightarrow a_j \not\Rsh a_i$ s'il existe une relation de régulation de forme $(b, a, \pm)$ dans R;

2 `unreachableProcess2`: supposons $b_k \rightarrow a_j \not\Rsh a_i$ est une de ses actions liées, vérifier si a_j est accessible;

- i si a_j est accessible, il est sûr que b_k n'est pas accessible, sinon a_i sera accessible. Classer b_k (dans la Section 1) et refaire le traitement que l'on a fais pour a_i, c'est une tâche récursive sans garantie de terminaison de même raisons que la section précédente, par conséquent il est nécessaire de noter tous les bounces liés à ce calcul, si on trouve la répétition de certain bounce, alors l'action liée ($b_k \rightarrow a_j \not\Rsh a_i$ dans ce cas-là) sera enlevée de **LA**;
- ii si b_k est accessible, traiter de même façon comme i
- iii si ni a_j ni b_k n'est accessible, enlever $b_k \rightarrow a_j \not\Rsh a_i$ des actions liées de a_i, puis:

  s'il y a d'autres actions liées, essayer la complétion avec une prochaine action;

  s'il n'y a pas d'autres action liées, ajouter a_j à `unreachableProcess1` et supprimer a_j de `unreachableProcess2`.

L'algorithme global deviendra :

[3]Le code est disponible sur https://github.com/XinweiChai/Completion/tree/master/src/completion

Données : Un fichier `.ph`, un processus inatteignable `p` et l'ensemble de relations de régulation `relation`
Résultat : Un Process Hitting complété
répéter
  Appliquer `ph-reach` sur `p`
  Algorithme 5 Classification
  Algorithme 6 **ou** Algorithme 7 Complétion selon `relation`
jusqu'à *`ph-reach`*$=$ *true* **ou** *inconclusive*;

Algorithme 2 : Structure de programme

En plus, cette méthode peut être encore simplifiée en appliquant seulement la sur-approximation, comme l'accessibilité cherchée ne dépend pas de la sous-approximation (la logique indiqué dans le tableau 3.1).

Exemple :
Étant donné un Process Hitting $PH = (\Sigma, L, \mathscr{H})$ avec $\Sigma = \{a, b, c\}$, $L = \{a_0, a_1\} \times \{b_0, b_1\} \times \{c_0, c_1\}$, $\mathscr{H} = \{a_1 \to b_0 \not\Rightarrow b_1\}$, son contexte $\varsigma = \{a_0, b_0, c_0\}$, et la relation de régulation $R = \{(a, b, +), (c, a, -)\}$.

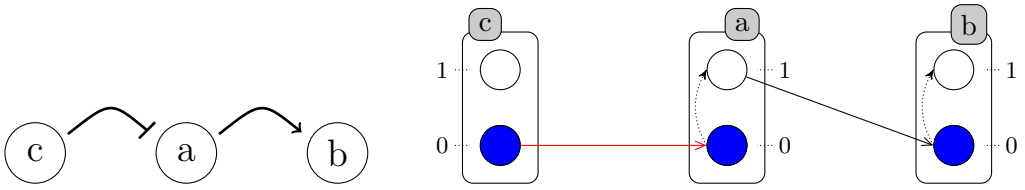

Figure 4.4: Au début, b_1 n'est pas accessible, comme il a une action liée $a_1 \to b_0 \not\Rightarrow b_1$, b_1 est classé dans `unreachableProcess2`. a_0 est évidemment inaccessible car b_0 est accessible (état initial). Selon $(c, a, -) \in R$, $c_0 \to a_0 \not\Rightarrow a_1$ est rajoutée, alors b_1 devient accessible.

Nous pouvons aussi simplifier le calcul de sur-approximation, comme dans l'article de Paulevé *et al.* [PMR12], la structure abstraite $\mathcal{A}^{\omega}_{\varsigma}$ est de forme $processus \to objectif \to solution \to processus \cdots$, mais il n'y a qu'un seul objectif suivi d'un processus. Alors le GLC peut être ainsi simplifié : $processus \to solution \to processus \cdots$.

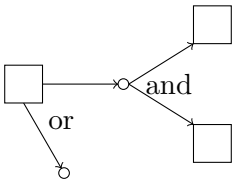

Figure 4.5: Le raisonnement local avec les nœuds dans la sur-approximation : les carrés représentent les processus , et les cercles représentent les solutions

4.5 Complétion basée sur le système retardé

Dans cette section, nous allons introduire une autre méthode de complétion de Process Hitting basée sur le système retardé proposée par Ben Abdallah *et al.* [AFRM15, ARM$^+$16]. Cette approche fonctionne de façon exhaustive mais limitée, en prenant les données d'observation de forme d'un chronogramme. Par le calcul de délai de chaque action, on décide enfin les actions rajoutées. Ce problème est résolue à l'aide de l'ASP [Bar03], un langage très concis et efficace.

4.5.1 Système biologique retardé

Definition 4.3 (Action plurielle temporisée)**.** Étant donné un Process Hitting $PH = (\Sigma, L, \mathcal{H})$. Une action plurielle est notée : $h = A \xrightarrow{D} b_j \Rsh b_k$ avec $A \in \mathcal{L}^\diamond \wedge b_j \neq b_k \wedge$ si $b_i \in A \Rightarrow A = \{b_j\}$. Où $\mathcal{L}^\diamond$ est l'ensemble de tous les sous-ensembles de $\mathcal{L}$ et D le délai de l'action.

Avec ces nouvelles actions, un système retardé modelé en Process Hitting peut être construit.

Definition 4.4 (Process Hitting avec actions plurielles temporisées)**.** Étant donné un Process Hitting $PH = (\Sigma, L, \mathscr{H})$, en changeant $\mathscr{H}$ en $\mathscr{H}_{tp} = \{A \xrightarrow{D} b_j \Rsh b_k \mid A \in \mathcal{L}^\diamond, b_j \neq b_k, b_i \in A \Rightarrow A = \{b_j\}\}$ qui est l'ensemble fini des actions plurielles temporisées, le nouveau Process Hitting $PH = (\Sigma, L, \mathscr{H}_{tp})$ est appelé Process Hitting avec actions plurielles temporisées.

4.5.2 Algorithme

Cette approche (Algorithme 8) prend en compte toutes les possibilités d'ajout d'actions du réseau, mais à cause de la sémantique retardée, il n'est pas possible de décider si une action rajoutée est correcte ou non. On considère que le résultat avec le minimum de modifications tel que l'observation et le modèle soient cohérents est le plus "correct".

Comme les contraintes de l'algorithme sont assez compliquées, il est conseillé d'utiliser l'ASP pour réaliser, qui est apte de décrire les contraintes en quelques phrases au lieu de concevoir plusieurs boucles.

Si l'observation n'est pas parfaite (mauvaise précision de mesure de temps), nous pourrons fusionner les actions en prenant la valeur moyenne du délai actuel et celui de l'observation.

L'exemple est Figure B.3 dans l'annexe.

Chapter 5

Discussion

Étant un modèle automate, le Process Hitting donne un aspect précis sur l'interaction entre les éléments dans les réseaux biologiques par rapport aux réseaux booléens. Il possède une approche efficace (en évitant le calcul exhaustif) pour calculer l'accessibilité de certain processus (sur- et sous-approximation), mais pas définitivement. Et nous pouvons aussi appliquer le cut set pour empêcher certain processus. À l'aide des approches de complétion, il est possible de trouver les actions à rajouter. Avec nos tâches, les fonctions de Process Hitting sont enrichies : les hypothèses sous forme de l'ensemble de relations de régulation R peuvent être vérifiées selon les expériences.

5.1 Commentaires de nouvelles approches

Avec le Process Hitting, la démarche de complétion s'effectue d'une façon dynamique et temporisée, qui est beaucoup plus riche que celle des réseaux booléens, mais à cause de la surjectivité de complétion (Figure 4.1), il existe souvent plusieurs résultats de complétion, qui rendent la vérification plus complexe, c'est-à-dire si les hypothèses (R) sont nombreuses, c'est pour cela qu'il est difficile de trouver le réseau réel dans un grande échelle.

En plus, la complétion basé sur le système retardé nécessite les données temporisées, qui est plus contraignante mais elle donne plus d'informations (action plurielle temporisée) que le Process Hitting original. La complétion basée sur la séquence stricte nécessite une séquence stricte, qui concentre plutôt sur l'ordre d'occurrence des processus. Ces deux approches sont basés sur les algorithmes exhaustifs, et il y a encore des améliorations possibles de la complexité.

5.2 Travail futur

Le travail futur consiste de rajouter des contraintes hors les relations de régulation, pour réduire encore le nombre de résultat et la complexité de calcul, et en plus, enrichir la sémantique de la complétion (e.g. l'action plurielle, la loi de probabilité d'action, etc) pour que la méthode complétion puisse être appliquée dans les cas variés.

Références

[AFRM15] Emna Ben Abdallah, Maxime Folschette, Olivier Roux, and Morgan Magnin. Exhaustive analysis of dynamical properties of biological regulatory networks with answer set programming. In *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, pages 281–285. IEEE, 2015.

[ARM+16] Emna Ben Abdallah, Tony Ribeiro, Morgan Magnin, Olivier Roux, and Katsumi Inoue. Inference of delayed biological regulatory networks from time series data. In *International Conference on Computational Methods in Systems Biology*, pages 30–48. Springer, 2016.

[ATH09] Tatsuya Akutsu, Takeyuki Tamura, and Katsuhisa Horimoto. Completing networks using observed data. In *International Conference on Algorithmic Learning Theory*, pages 126–140. Springer, 2009.

[Bar03] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, 2003.

[Fil60] Aleksei Fedorovich Filippov. Differential equations with discontinuous right-hand side. *Matematicheskii sbornik*, 93(1):99–128, 1960.

[Fox93] Ronald F Fox. Review of stuart kauffman, the origins of order: Self-organization and selection in evolution. *Biophysical journal*, 65(6):2698, 1993.

[GK73] Leon Glass and Stuart A Kauffman. The logical analysis of continuous, non-linear biochemical control networks. *Journal of theoretical Biology*, 39(1):103–129, 1973.

[NIIR10] Hidetomo Nabeshima, Koji Iwanuma, Katsumi Inoue, and Oliver Ray. Solar: An automated deduction system for consequence finding. *AI communications*, 23(2-3):183–203, 2010.

[PAK13] Loïc Paulevé, Geoffroy Andrieux, and Heinz Koeppl. Under-approximating cut sets for reachability in large scale automata networks. In *International Conference on Computer Aided Verification*, pages 69–84. Springer, 2013.

[PCF+14] Loïc Paulevé, Courtney Chancellor, Maxime Folschette, Morgan Magnin, and Olivier Roux. Analyzing large network dynamics with process hitting, 2014.

[PMR11] Loïc Paulevé, Morgan Magnin, and Olivier Roux. Refining dynamics of gene regulatory networks in a stochastic π-calculus framework. In *Transactions on computational systems biology xiii*, pages 171–191. Springer, 2011.

[PMR12] Loïc Paulevé, Morgan Magnin, and Olivier Roux. Static analysis of biological regulatory networks dynamics using abstract interpretation. *Mathematical Structures in Computer Science*, 22(4):651–685, 2012.

[RCB06] Adrien Richard, Jean-Paul Comet, and Gilles Bernot. Formal methods for modeling biological regulatory networks. In *Modern formal methods and applications*, pages 83–122. Springer, 2006.

[Tho78] René Thomas. Logical analysis of systems comprising feedback loops. *Journal of Theoretical Biology*, 73(4):631–656, 1978.

[YRN+14] Yoshitaka Yamamoto, Adrien Rougny, Hidetomo Nabeshima, Katsumi Inoue, Hisao Moriya, Christine Froidevaux, and Koji Iwanuma. Completing sbgn-af networks by logic-based hypothesis finding. In *International Conference on Formal Methods in Macro-Biology*, pages 165–179. Springer, 2014.

Appendix A

Algorithme de complétion

Comment simuler le réseau pas à pas:

Données : Un Process Hitting et un état actuel
Résultat : L'ensemble de processus futurs possibles après un tir
pour chaque $a \in A$ **faire**
  si *hitter*(a) *est dans l'état actuel* **alors**
    `Result` $\leftarrow$ `Result` $+$ *bounce*(a)
  fin
fin
retourner `Result`

Algorithme 3 : Transition

```
Données : Un Process Hitting, une séquence sequence, Result de Transition
Résultat : realisability, ensemble unreachableProcess
début Fonction realisability(state, sequence, steps)
    si sequence est vide alors
        retourner 0
    sinon
        si Le 1er élément de sequence est dans l'état actuel alors
            Supprimer le 1er élément de sequence
            retourner realisability(state, sequence, steps)
        sinon
            pour tous e dans sequence sauf le 1er faire
                si e est dans state et e ≠ sequence[1]) alors
                    unreachableProcess ← unreachableProcess + sequence[1]
                    retourner 2
                fin
            fin
        fin
        si Result = φ ou steps = 0 alors
            unreachableProcess ← unreachableProcess + sequence[1]
            retourner 1
        sinon
            si Card(Result)=1 alors
                si la sorte de Result[1] est observable et Result[1] ≠ sequence[1] alors
                    unreachableProcess ← unreachableProcess + sequence[1]
                    retourner 2
                sinon
                    retourner realisability(state, sequence, steps-1)
                fin
            sinon
                pour tous Branches dans Result faire
                    si realisability(state, sequence, steps-1)≠ 0 alors
                        unreachableProcess ← unreachableProcess + sequence[1]
                    fin
                    Val ← Val+realisability(state, sequence, steps-1)
                    si Il y a un 0 dans Val alors
                        retourner 0
                    sinon
                        si Il y a un 1 dans Val alors
                            retourner 1
                        sinon
                            retourner 2
                        fin
                    fin
                fin
            fin
        fin
    fin
fin
```

Algorithme 4 : Réalisabilité

Données : Un Process Hitting, `unreachableProcess`
Résultat : `unreachableProcess1` et `unreachableProcess2`
pour *p dans* `unReachableProcess` **faire**
  pour *a dans Actions* **faire**
    si *bounce(a)=p* **alors**
      `unreachableProcess2` $\leftarrow$ `unreachableProcess2` $+ p$
    fin
  fin
  si *p n'a pas été ajouté à* `unreachableProcess2` **alors**
    `unreachableProcess1` $\leftarrow$ `unreachableProcess1` $+ p$
  fin
fin

Algorithme 5 : Classification des processus inaccessibles

Données : Un Process Hitting ,un ensemble de relations de régulation, un processus b_k dans `unreachableProcess1`(avec $\mathbf{LA}(b_k) = \varnothing$)
Résultat : Au plus une action ajoutée
début Fonction `addAction1`(b_k)
  pour *r dans* `relation` **faire**
    si *Action* $a_i \rightarrow b_j \Rsh b_k$ *est cohérent avec r* **alors**
      addActions $\leftarrow$ *addActions* $+ a_i \rightarrow b_j \Rsh b_k$
    fin
  fin
  Supprimer les éléments dans addActions égaux aux actions existantes.
  retourner *addActions*
fin

Algorithme 6 : Ajout d'action pour `unreachableProcess1`

Données : Un Process Hitting ,`sequence`, un processus b_k dans `unreachableProcess2`(avec $\mathbf{LA}(b_k) = \mathtt{a} \neq \varnothing$)
Résultat : Au plus une action ajoutée
Initialisation: iterations $\leftarrow 0$,
début Fonction `addAction2(`b_k`, a)`,
  si *iteration* > 10 **alors**
    retourner addAction1(b_k)
  fin
  iterations $\leftarrow$ iterations + 1
  si $a = \phi$ **alors**
    retourner addAction1(b_k)
  fin
  firstAction $\leftarrow$ a[0] ; `// Supposons firstAction =` $a_i \rightarrow b_j \not\Vdash b_k$
  a $\leftarrow$ a-a[0]
  prec $\leftarrow$ false ; `// Marque si le cible apparaît avant le bond`
  pour *i dans* `sequence` **faire**
    si $i = target(firstAction)$ **alors**
      prec $\leftarrow$ true
    fin
    si $i \neq b_j$ **et** $i \neq b_k$ **et** *i est de sorte b* **alors**
      prec $\leftarrow$ false
    fin
    `/*` b_k `apparaît forcément dans sequence */`
    si $i = b_k$ **alors**
      si *prec = true* **alors**
        si a_i *a des actions liées* **alors**
          retourner addAction2(a_i, actions liées de a_i)
        sinon
          retourner addAction1(a_i)
        fin
      sinon
        retourner addAction2(b_k, a)
      fin
    fin
  fin
fin

Algorithme 7 : Ajout d'action pour `unreachableProcess2`

Données : un Process Hitting PH, un chronogramme C d'entrée de taille T : les niveaux de toutes les sortes sont donnés pour à tout instant $0 \leq t \leq T$, un ensemble de relations de régulation R et un degré entrant maximal d'action i.

$A \leftarrow \varnothing$

pour $\forall g \in G$ **faire**

  Générer toutes les actions cohérente avec R de taille maximale i en portant un délai D, puis les ajouter dans A

fin

$S' = \mathcal{P}(A)$

$S = \{s \in S' \mid C \text{ est réalisable}\}$

pour $\forall S$ **faire**

  pour $A \in S$ **faire**

    Choisir A_{min} tel que $\forall A \in S, \nexists A \subset A_{min}$

    et que $\forall h \in A$, avec $h =\xrightarrow{D} b_j \Rsh b_k, \nexists h' =\xrightarrow{D'} b_j \Rsh b_k, D \neq D'$

  fin

fin

Résultat : S l'ensemble de Process Hitting complété qui réalise ce chronogramme.

Algorithme 8 : Algorithme de complétion basée sur le système retardé

Appendix B

Exemple de la complétion basée sur le système retardé

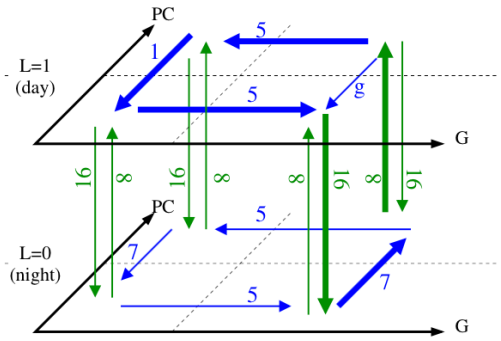

Figure B.1: *

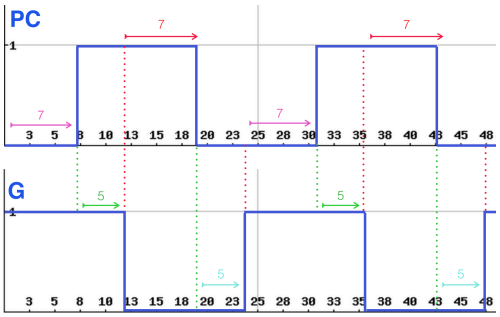

Figure B.2: *

(a) (b)

Figure B.3: (a) le modèle qualitatif de cycle circadien mammalien pendant l'été (b) le chronogramme correspondant à la discrétisation des données d'observation des processus pendant la nuit ($L = 0$).

Selon Algorithme 8, 10 actions sont générées :

$\{L_0\} \xrightarrow{8} L_0 \Rsh L_1$	$\{L_1\} \xrightarrow{16} L_1 \Rsh L_0$
$\{L_0, G_0\} \xrightarrow{7} PC_1 \Rsh PC_0$	$\{L_0, PC_0\} \xrightarrow{5} G_0 \Rsh G_1$
$\{L_0, G_1\} \xrightarrow{7} PC_0 \Rsh PC_1$	$\{L_0, PC_1\} \xrightarrow{5} G_1 \Rsh G_0$
$\{L_1, G_0\} \xrightarrow{7} PC_1 \Rsh PC_0$	$\{L_1, PC_0\} \xrightarrow{5} G_0 \Rsh G_1$
$\{L_1, G_1\} \xrightarrow{7} PC_1 \Rsh PC_0$	$\{L_1, PC_1\} \xrightarrow{5} G_1 \Rsh G_0$

qui vérifie et enrichit la sémantique du réseau de régulation dans Figure B.4 :

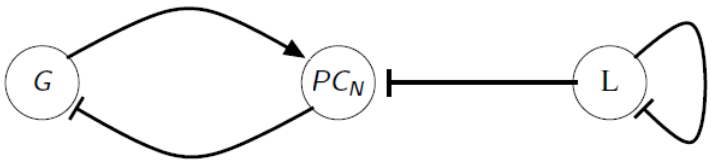

Figure B.4: Le graphe de régulation simplifié du modèle de cycle circadien mammalien